Ding-Zhu Du
Xiaodong Hu
Panos M. Pardalos (Eds.)

Combinatorial Optimization and Applications

LNCS 5573

Third International Conference, COCOA 2009
Huangshan, China, June 2009
Proceedings

Springer

Lecture Notes in Computer Science 5573

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Ding-Zhu Du Xiaodong Hu
Panos M. Pardalos (Eds.)

Combinatorial Optimization and Applications

Third International Conference, COCOA 2009
Huangshan, China, June 10-12, 2009
Proceedings

Springer

Volume Editors

Ding-Zhu Du
University of Texas at Dallas, Department of Computer Science
800 West Campbell Road, Richardson, TX 75080-3021, USA
E-mail: dzdu@utdallas.edu

Xiaodong Hu
Chinese Academy of Sciences, Institute of Applied Mathematics
Zhong Guan Cun Dong Lu 55, Beijing 100190, P. R. China
E-mail: xdhu@amss.ac.cn

Panos M. Pardalos
University of Florida, Department of Industrial and Systems Engineering
303 Weil Hall, P.O. Box 116595, Gainesville, FL 32611-6595, USA
E-mail: pardalos@ufl.edu

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.2, G.1.6, G.2, H.1, I.3.5

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-02025-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-02025-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12684359 06/3180 5 4 3 2 1 0

Preface

The Third Annual International Conference on Combinatorial Optimization and Applications, COCOA 2009, took place in Huangshan, China, June 10–12, 2009. Past COCOA conferences were held in Xi'an, China (2007) and Newfoundland, Canada (2008).

COCOA 2009 provided a forum for researchers working in the areas of combinatorial optimization and its applications. In addition to theoretical results, the conference is particularly focused on recent works on experimental and applied research of general algorithmic interest. The Program Committee received 103 submissions from 17 countries and regions: Brazil, Canada, China, Denmark, France, Germany, Hong Kong, India, Italy, Japan, Korea, Malaysia, Poland, Switzerland, Taiwan, UK, and USA.

Among the 103 submissions, 50 papers were selected for presentation at the conference and are included in this volume. Some of these will be selected for publication in a special issue of *Journal of Combinatorial Optimization*, a special issue of *Theoretical Computer Science*, and a special issue of *Discrete Mathematics, Algorithms and Applications* under the standard refereeing procedure. In addition to the selected papers, the conference also included one invited presentation by Panos M. Pardalos (University of Florida, USA).

We thank the authors for submitting their papers to the conference. We are grateful to the members of the Program Committee and the external referees for their work within demanding time constraints. We thank the Organizing Committee for their contribution to make the conference a success. We also thank Donghyun Kim for helping us create and update the conference website and maintain the Springer Online Conference Service system.

Finally, we thank the conference sponsors and supporting organizations for their support and assistance. COCOA 2009 was supported in part by the National Natural Science Foundation of China under Grant No. 10531070, 10771209, 10721101 and the Chinese Academy of Sciences under Grant No. kjcx-yw-s7. COCOA 2009 was held at Huangshan International Hotel, Huangshan, China.

June 2009

Ding-Zhu Du
Xiaodong Hu
Panos M. Pardalos

Organization

Executive Committee

Conference Chair: Ding-Zhu Du University of Texas at Dallas, USA

Program Committee

Vladimir L. Boginski	University of Florida, USA
Sergiy Butenko	Texas A&M University, USA
Zhipeng Cai	University of Alberta, Canada
Gerard Jennhwa Chang	National Taiwan University, Taiwan
Maggie Cheng	Missouri University of Science and Technology, USA
Wanpracha Art Chaovalitwongse	The State University of New Jersey, USA
Bo Chen	University of Warwick, UK
Ding-Zhu Du	University of Texas at Dallas, USA; Co-chair
Rudolf Fleischer	Fudan University, China
Christodoulos A. Floudas	Princeton University, USA
Xiaodong Hu	Chinese Academy of Sciences, China; Co-chair
Hejiao Huang	Harbin Institute of Technology, China
Marek Karpinski	University of Bonn, Germany
Naoki Katoh	Kyoto University, Japan
Ilias S. Kotsireas	Wilfrid Laurier University, Canada
Der-Tsai Lee	Acdemia Sinica, Taiwan
Guohui Lin	University of Alberta, Canada
Carlos Oliveira	Princeton Consultants, Inc., USA
Panos M. Pardalos	University of Florida, USA; Co-chair
Leonidas Pitsoulis	Aristotle University of Thessaloniki, Greece
Oleg Prokopyev	University of Pittsburgh, USA
Yuehua Pu	Zhejiang Normal University, China
Liqun Qi	Hong Kong Polytechnic University, Hong Kong
Michel Raynal	Université de Rennes, France
Franz Rendl	Universität Klagenfurt, Austria
Lu Ruan	Iowa State University, USA
Onur Seref	Virginia Tech, USA
Alexander Schliep	Max Planck Institute for Molecular Genetics, Germany
Guochun Tang	Shanghai University, China
My T. Thai	University of Florida, USA

Doreen Anne Thomas	University of Melbourne, Austrilia
Feng Wang	Arizona State University, USA
Jie Wang	University of Massachusetts Lowell, USA
Lusheng Wang	City University of Hong Kong, Hong Kong
Weili Wu	University of Texas at Dallas, USA
Yinfeng Xu	Xi'an Jiaotong University, China
Boting Yang	University of Regina, Canada
Jianzhong Zhang	Chinese University of Hong Kong, Hong Kong
Zhao Zhang	Xinjiang University, China
Constantin Zopounidis	Technical University of Crete, Greece

Organizing Committee

Xujin Chen	Chinese Academy of Sciences, China
Jie Hu	Operations Research Society of China, China
Xudong Hu	Chinese Academy of Sciences, China; Chair
Aimin Jiang	Chinese Academy of Sciences, China
Degang Liu	Operations Research Society of China, China

Sponsoring Institutions

Academy of Math and System Science, Chinese Academy of Sciences, China
Operations Research Society of China, China
University of Texas at Dallas, USA

Table of Contents

Algorithms for Network Design

Bioinformatics

Combinatorics and Its Applications

Computational Geometry

Game Theory

Graph Algorithms

Graph Theory

Network Models and Problems

On-line Algorithms

Size-Problems

Scheduling

Wireless and Optical Networks

Polynomial Approximation Schemes for the Max-Min Allocation Problem under a Grade of Service Provision*

Jianping Li[1,**], Weidong Li[1], and Jianbo Li[2]

[1] Department of Mathematics, Yunnan University, Kunming 650091, China
jianping@ynu.edu.cn, liweidong@mail.ynu.edu.cn
[2] School of Management and Economics, Kunming Uni. of Sci. and Tech., China
lijianbo@public.km.yn.cn

Abstract. The max-min allocation problem under a grade of service provision is defined in the following model: given a set $\mathcal{M}$ of m parallel machines and a set $\mathcal{J}$ of n jobs, where machines and jobs are all entitled to different levels of grade of service (GoS), each job $J_j \in \mathcal{J}$ has its processing time p_j and it is only allocated to a machine M_i whose GoS level is no more than the GoS level the job J_j has. The goal is to allocate all jobs to m machines to maximize the *minimum machine load*, where the machine load of machine M_i is the sum of the precessing times of jobs executed on M_i. The best approximation algorithm [4] to solve this problem produces an allocation in which the minimum machine completion time is at least $\Omega(loglogm/loglogm)$ of the optimal value.

In this paper, we respectively present four approximation schemes to solve this problem and its two special versions: (1) a polynomial time approximation scheme (PTAS) with running time $O(mn^{O(1/\epsilon^2)})$ for the general version, where $\epsilon > 0$; (2) a PTAS and an fully polynomial time approximation scheme (FPTAS) with running time $O(n)$ for the version where the number m of machines is fixed; (3) a PTAS with running time $O(n)$ for the version where the number of GoS levels is bounded by k.

Keywords: Scheduling; allocation; grade of service; polynomial time approximation scheme; fully polynomial time approximation scheme.

1 Introduction

The max-min allocation problem, treated as the Santa Claus problem [2,4] and the fair division [3,6,8], has been studied widely in many domains. This problem is formulated in the following model. There are m players (or machines) and n items (or jobs). Let p_{ij} (≥ 0) denote the value (or processing time) of item j to

* The work is fully supported by the National Natural Science Foundation of China [No.10861012,10561009], Natural Science Foundation of Yunnan Province [No.2006F0016M] and Foundation of Younger Scholar in Science and Technology of Yunnan Province [No.2007PY01-21].

** Correspondence author.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 1–13, 2009.

player i. The goal is to partition n items into m disjoint bundles, $S_1, \ldots, S_m$, such that $\min\{ \sum_{j \in S_i} p_{ij} \mid 1 \leq i \leq m\}$ is maximized.

The max-min allocation problem is viewed as a "dual" problem to the famous problem of makespan minimization on unrelated parallel machines. Lenstra *et al.* [12] designed a 2-approximation algorithm based on linear programming for the makespan problem and then proved that approximating it with ratio better than 1.5 is *NP*-hard. The problem of finding a better than 2-approximation algorithm (or improving the lower bound) for the makespan minimization problem is one of the ten open challenging problems in the area of approximation algorithms for machine scheduling [14].

The max-min allocation problem was first studied as a machine scheduling problem, where the goal is to maximize the minimum machine completion time. Alon *et al.* [1] gave a polynomial time approximation scheme (PTAS) with running time $O(n)$ for the problem $P||C_{min}$ in which each job has an intrinsic value p_j and $p_{ij} = p_j$ for all machines M_i. Epstein and Sgall [7] presented a PTAS for the version where the machine M_i has a speed s_i and $p_{ij} = p_j/s_i$ for machine M_i and job J_j, here $1 \leq i \leq m$ and $1 \leq j \leq n$.

For the max-min allocation problem, Bezakova and Dani [5] first provided a simple approximation algorithm with the worst-case performance ratio $n - m + 1$ and showed that this problem is *NP*-hard to be approximated within $2 - \epsilon$, for any $\epsilon > 0$. Asadpour and Saberi [3] presented an $O(\sqrt{m}log^3 m)$-approximation algorithm for this problem, by using the LP relaxation method. Recently, Chakrabarty *et al.* [6] designed an $\widetilde{O}(n^{\epsilon})$ approximation algorithm for this problem in running time $n^{O(1/\epsilon)}$, where $\epsilon = \Omega(loglogn/logn)$. Woeginger [15] gave an FPTAS for this problem in the version where the number of machines is a constant, but its running time is very high.

Bansal and Sviridenko [4] studied a restricted version of the max-min allocation problem, treated as the Santa Claus problem [2,4], where $p_{ij} \in \{p_j, 0\}$, and they proposed a nontrivial $O(loglogm/logloglogm)$-approximation algorithm for the problem, by using the configuration of LP relaxation. Feige [8] showed a constant upper bound on the integrality gap of configuration LP for the Santa Claus problem, however, his proof is not constructive. Asadpour *et al.* [2] presented an alternative non-constructive proof of a factor-5 upper bound on the integrality gap of the configuration LP.

In this paper, we consider the max-min allocation problem under a grade of service provision, defined by: given a set $\mathcal{M} = \{M_1, \ldots, M_m\}$ of machines and a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of jobs, each job J_j has the processing time p_j and is labelled with the GoS level $g(J_j)$, and each machine M_i is also labelled with the GoS level $g(M_i)$, job J_j is allowed to be allocated to machine M_i only when $g(J_j) \geq g(M_i)$. The goal is to partition the set $\mathcal{J}$ into m disjoint bundles, $S_1, \ldots, S_m$, such that $\min\{ \sum_{J_j \in S_i} p_j \mid 1 \leq i \leq m\}$ is maximized, where $J_j \in S_i$ only if $g(J_j) \geq g(M_i)$. We denote this model as the problem $P|GoS|C_{min}$, by the three-field notation of Graham *et al.* [9]. For convenience, we denote a notation, by $I = (\mathcal{J}, \mathcal{M}; p, g)$, to represent an instance of the problem $P|GoS|C_{min}$. We also consider its two basic variants: the problem $P_m|GoS|C_{min}$, where the

number m of machines is a fixed constant, and the problem $P|GoS_k|C_{min}$, where $g(M_i)$, $g(J_j) \in \{1, 2, \ldots, k\}$ and k is a fixed constant.

Our max-min allocation problem under a grade of service provision is closely related to the problem of parallel machine scheduling under a grade of service provision [10], but the goal in [10] is to minimize $\max\{\sum_{J_j \in S_i} p_j \mid 1 \leq i \leq m\}$, and our goal is to maximize $\min\{\sum_{J_j \in S_i} p_j \mid 1 \leq i \leq m\}$. Ou *et al.* [13] designed a PTAS for the problem of parallel machine scheduling under a grade of service provision.

It is easy to verify that the difficulty to solve the problem $P|GoS|C_{min}$ is somewhere between the parallel machine problem $P||C_{min}$ and the Santa Claus problem. To our knowledge by now, the best approximation algorithm to solve the problem $P|GoS|C_{min}$ is the $O(loglogm/logloglogm)$ approximation algorithm designed for the Santa Claus problem [4], which is based on rounding a certain natural exponentially large linear programming relaxation usually referred to as the configuration LP. In this paper, we first presented a PTAS for the problem $P|GoS|C_{min}$ in running time $O(mn^{O(1/\epsilon^2)})$, for any $\epsilon > 0$. In addition, we notice that Woeginger [15] provided an FPTAS for the problem $P_m|GoS|C_{min}$, however, the running time in [15] is high, which leads an interest that we present a new PTAS and a new FPTAS for the problem $P_m|GoS|C_{min}$ in linear time. Afterwards, we also design a PTAS for the problem $P|GoS_k|C_{min}$ in linear time.

This paper is divided into the following sections. In Section 2, we construct a transformed instance from any instance of the problem $P|GoS|C_{min}$, and then prove some important lemmas. In Section 3, we present a PTAS with running time $O(mn^{O(1/\epsilon^2)})$ for the problem $P|GoS|C_{min}$. In Section 4, we present a PTAS and an FPTAS with running time $O(n)$ for the problem $P_m|GoS|C_{min}$, where the number m of machines is a fixed constant. In Section 5, we design a PTAS with running time $O(n)$ for the problem $P|GoS_k|C_{min}$. We provide our conclusions and future work in the last section.

2 Original Instance and Transformed Instance

For any given instance $I = (\mathcal{J}, \mathcal{M}; p, g)$ of the problem $P|GoS|C_{min}$, sort the machines in nonincreasing order of their GoS levels, *i.e.*, $g(M_1) \geq g(M_2) \geq \cdots \geq g(M_m)$. Let $v_j = \min\{i \mid g(M_i) \leq g(J_j)\}$ denote the machine index of job J_j, and $J_{[i]} = \{J_j \mid v_j = i\}$ denote the set of jobs with machine index i $(i = 1, 2, \ldots, m)$. Let

$$L = \min\left\{\frac{\sum_{i=1}^{t} p(J_{[i]})}{t} \;\middle|\; t = 1, 2, \ldots, m\right\}$$

denote the minimum average load of the first t machines with $p(J_{[i]}) = \sum_{J_j \in J_{[i]}} p_j$. Clearly, $L \geq OPT$, where OPT denotes the optimal value for the instance $I = (\mathcal{J}, \mathcal{M}; p, g)$.

For the problem $P|GoS|C_{min}$, a feasible allocation must allocate all jobs to the machines such that each job is assigned to exactly one machine. In fact,

we call that an allocation $\sum$ is *feasible* if each job is allocated at most one machine, because we can allocate the remaining jobs to arbitrary machines to get a solution such that each job is allocated to exactly one machine.

The following two observations are similar to the two observations in Alon et al. [1].

Observation 1. *If a job J_j has processing time $p_j \geq L$, then there exists an optimum allocation $\sum^* = (S_1^*, S_2^*, \ldots, S_m^*)$ in which job J_j is the only job assigned to its machine.*

Proof. Suppose that in some optimal allocation $\sum^*$, job J_j is assigned together with some other jobs to a machine M_i. As L is an upper bound on OPT, removing all jobs except J_j from the machine M_i cannot decrease the objective function. □

Hence, if there is a job J_j with processing time $\geq L$, we assign it to the machine M_{v_j}. If $v_j < m$, remove both the job J_j and the machine M_{v_j} from the setting, and reset $J_{[v_j+1]} = J_{[v_j+1]} \cup J_{[v_j]} - \{J_j\}$; otherwise, remove both the jobs in $J_{[m]}$ and the machine M_m from the setting. Then, we can handle a smaller instance without changing the optimal value. Thus, we may assume that no jobs has processing time $\geq L$.

Observation 2. *If no job has processing time $\geq L$, then there exists an optimum allocation $\sum^* = (S_1^*, S_2^*, \ldots, S_m^*)$ in which all machine completion times C_i satisfy $L/2 \leq C_i \leq 2L$, where $C_i = \sum_{J_j \in S_i^*} p_j$ for $1 \leq i \leq m$.*

Proof. For convenience, given an instance I of the problem $P|GoS|C_{min}$, we refer a job J_j as *large job* if J_j has processing time $\geq L/2$ and otherwise as *small job*. First, suppose that $C_i > 2L$ holds for some machine M_i, discarding jobs in S_i^* until $C_i \leq 2L$ cannot decrease the goal value.

Next, we will prove that $L/2$ is a lower bound on OPT by presenting the following algorithm: For $i = 1$ to m, if there remains a large job J_j in $J_{[1]} \cup \cdots \cup J_{[i]}$, we assign job J_j to the machine M_i, then stop assigning more jobs to M_i; otherwise, we assign small jobs in $J_{[1]} \cup \cdots \cup J_{[i]}$ to M_i until the completion time of M_i first exceeds $L/2$. In the end, every machine completion time must exceed $L/2$. Suppose, to the contrary, that the completion time of machine M_τ is less than $L/2$, which implies no jobs in $J_{[1]} \cup \cdots \cup J_{[\tau]}$ are remained after assigning jobs to machine M_τ.

According to our algorithm, each machine completion time C_i is no greater than L. Hence, combining the definition of L, we have

$$\sum_{i=1}^{\tau} p(J_{[i]}) = \sum_{i=1}^{\tau} C_i < \tau L \leq \tau \frac{\sum_{i=1}^{\tau} p(J_{[i]})}{\tau} = \sum_{i=1}^{\tau} p(J_{[i]}),$$

a contradiction. Hence, the observation 2 holds. □

In the sequel, similar to the strategy in Alon et al. [1], we introduce a so-called transformed instance that is a kind of rounded version of the original instance

$I = (\mathcal{J}, \mathcal{M}; p, g)$ of the problem $P|GoS|C_{min}$, and that can easily be solved in polynomial time.

For any instance $I = (\mathcal{J}, \mathcal{M}; p; g)$ of the problem $P|GoS|C_{min}$ and an integer constant λ, we construct a *transformed instance* $I^\sharp(\lambda)$ from the instance $I = (\mathcal{J}, \mathcal{M}; p, g)$ as follows:

• Assume that the processing time of any job is less than L. Partition the jobs into two subsets J^B and J^S, where $J^B = \{J_j \mid p_j > L/\lambda\}$ and $J^S = \{J_j \mid p_j \leq L/\lambda\}$.

• For every job J_j in J^B, the transformed instance $I^\sharp(\lambda)$ contains a *corresponding job* $J_j^\sharp$ with processing time

$$p_j^\sharp = \left\lceil \frac{p_j}{L/\lambda^2} \right\rceil \frac{L}{\lambda^2}.$$

It is easy to verify that

$$p_j^\sharp \leq p_j + \frac{L}{\lambda^2} \leq p_j + \frac{p_j}{\lambda} \leq \frac{\lambda+1}{\lambda} p_j.$$

The GoS level of job $J_j^\sharp$ is indicated as the original GoS level $g(J_j)$ of job J_j.

• For $i = 1, 2, \ldots, m$, let $J_{[i]}^S = J^S \cap J_{[i]}$. The instance $I^\sharp(\lambda)$ contains a number of *auxiliary jobs*, each of processing time L/λ. Specifically, for $i = 1, 2, \ldots, m$, let $J_{[i]}^A$ be a set of k_i auxiliary jobs with machine index i, where

$$k_i = \left\lceil \frac{\sum_{J_j \in J_{[i]}^S} p_j}{L/\lambda} \right\rceil.$$

Let $p_j^\sharp = L/\lambda$ denote the processing time of the auxiliary job J_j^A. Clearly,

$$\sum_{J_j \in J_{[i]}^S} p_j \leq \sum_{J_j^A \in J_{[i]}^A} p_j^\sharp \leq \sum_{J_j \in J_{[i]}^S} p_j + \frac{L}{\lambda}.$$

• The machines remain the same as in the instance I.

We notice that in the instance $I^\sharp(\lambda)$, the processing times of all jobs are integer multiples of L/λ^2. Let

$$L^\sharp = \min \left\{ \frac{\sum_{i=1}^t \left(p^\sharp(J_{[i]}^B) + p^\sharp(J_{[i]}^A)\right)}{t} \;\middle|\; t = 1, 2, \ldots, m \right\}$$

denote the minimum average load of the first t machines in the instance $I^\sharp(\lambda)$.

By the fact

$$\sum_{i=1}^t p(J_{[i]}) = \sum_{i=1}^t \left(\sum_{J_j \in J_{[i]}^B} p_j + \sum_{J_j \in J_{[i]}^S} p_j \right)$$

$$\leq \sum_{i=1}^{t}\left(\sum_{J_j^\sharp \in J_{[i]}^B} p_j^\sharp + \left\lceil \frac{\sum_{J_j\in J_{[i]}^S} p_j}{\frac{L}{\lambda}} \right\rceil \frac{L}{\lambda}\right)$$
$$= \sum_{i=1}^{t}\left(\sum_{J_j^\sharp \in J_{[i]}^B} p_j^\sharp + \sum_{J_j^A \in J_{[i]}^A} p_j^\sharp\right)$$
$$= \sum_{i=1}^{t}\left(p^\sharp(J_{[i]}^B) + p^\sharp(J_{[i]}^A)\right)$$

we have

$$L = \min\left\{\frac{\sum_{i=1}^{t} p(J_{[i]})}{t} \;\middle|\; t = 1,\ldots,m\right\}$$
$$\leq \min\left\{\frac{\sum_{i=1}^{t}(p^\sharp(J_{[i]}^B) + p^\sharp(J_{[i]}^A))}{t} \;\middle|\; t = 1,\ldots,m\right\} = L^\sharp.$$

And by the fact

$$\sum_{i=1}^{t}\left(p^\sharp(J_{[i]}^B) + p^\sharp(J_{[i]}^A)\right) = \sum_{i=1}^{t}\left(\sum_{J_j^\sharp \in J_{[i]}^B} p_j^\sharp + \sum_{J_j^A \in J_{[i]}^A} p_j^\sharp\right)$$
$$= \sum_{i=1}^{t}\left(\sum_{J_j\in J_{[i]}^B} \left\lceil \frac{p_j}{\frac{L}{\lambda^2}} \right\rceil \frac{L}{\lambda^2} + \left\lceil \frac{\sum_{J_j\in J_{[i]}^S} p_j}{\frac{L}{\lambda}} \right\rceil \frac{L}{\lambda}\right)$$
$$\leq \sum_{i=1}^{t}\left(\sum_{J_j\in J_{[i]}^B} p_j + \frac{L}{\lambda^2} + \sum_{J_j\in J_{[i]}^S} p_j + \frac{L}{\lambda}\right)$$
$$\leq \sum_{i=1}^{t} p(J_{[i]}) + t\frac{2L}{\lambda}$$

we have $L \leq L^\sharp \leq (1 + 2/\lambda)L$.

Similar to Observation 2, we may assume that there is no job has processing time $\geq L^\sharp$ in the instance $I^\sharp(\lambda)$.

Observation 3. *If no job has processing time $\geq L^\sharp$ in the instance $I^\sharp(\lambda)$, then there exists an optimum allocation in which all machine completion times C_i satisfy $L^\sharp/2 \leq C_i \leq 2L^\sharp$, where $1 \leq i \leq m$.*

From now on, for convenience, for a job J_j in the instance I or in the transformed instance $I^\sharp(\lambda)$, we refer job J_j as a *large job* if it has the processing time $> L/\lambda$ and otherwise as a *small job*, *i.e.*, having the processing time $\leq L/\lambda$.

The two following lemmas about the relationships between allocations for the instance I and allocations for the instance $I^\sharp(\lambda)$ are very important to our approximation schemes.

Lemma 1. *If there exists an allocation $\sum$ for the instance I with minimum machine completion time at least C, then there exists an allocation $\sum^\sharp$ for the instance $I^\sharp(\lambda)$ with minimum machine completion time at least $C - L/\lambda$.*

Proof. In the allocation $\sum$, replace every large job J_j in the instance I by its corresponding job $J_j^\sharp$ in the instance $I^\sharp(\lambda)$. This can never decrease a machine completion time. Next, rearrange on every machine M_i and the jobs in such a way that the small jobs are processed in the end, and let s_i denote their total size. Clearly, we have $\sum_{i=1}^{t} s_i \le \sum_{i=1}^{t} p(J_{[i]}^S)$ for all $t \ge 1$. For $i = 1, 2, \ldots, m$, we assign $\lfloor \frac{s_i}{L/\lambda} \rfloor$ remaining auxiliary jobs in $J_{[1]}^A \cup \cdots \cup J_{[i]}^A$ to M_i. This may decrease a machine completion time at most L/λ. Suppose that there exists a machine M_τ which can not be assigned $\lfloor \frac{s_i}{L/\lambda} \rfloor$ auxiliary jobs, we have

$$\sum_{i=1}^{\tau} \left\lfloor \frac{s_i}{L/\lambda} \right\rfloor \frac{L}{\lambda} > \sum_{i=1}^{\tau} \left\lceil \frac{p(J_{[i]}^S)}{L/\lambda} \right\rceil \frac{L}{\lambda} \ge \sum_{i=1}^{\tau} p(J_{[i]}^S) \ge \sum_{i=1}^{\tau} s_i \ge \sum_{i=1}^{\tau} \left\lfloor \frac{s_i}{L/\lambda} \right\rfloor \frac{L}{\lambda}$$

a contradiction. Hence, the lemma holds. □

Lemma 2. *Let $\sum^\sharp$ be an allocation for the instance $I^\sharp(\lambda)$ with minimum machine completion time at least $C^\sharp$. Then there exists an allocation $\sum$ for the instance I with minimum machine completion time at least $\lambda C^\sharp/(1+\lambda) - 2L/\lambda$.*

Proof. In the allocation $\sum^\sharp$, replace every large job $J_j^\sharp$ by its corresponding job J_j in the instance I. This may decrease a machine completion time at most a factor of $\lambda/(\lambda+1)$. Next, let $s_i^\sharp$ denote the number of auxiliary jobs of processing time L/λ that are processed on machine M_i in the allocation $\sum^\sharp$; rearrange on every machine M_i the jobs in such a way that these auxiliary jobs are processed in the end.

For $i = 1, 2, \ldots, m$, we assign the remaining small jobs in $J_{[1]}^S \cup J_{[2]}^S \cup \cdots \cup J_{[i]}^S$ to M_i until the total size of small jobs exceeds $(s_i^\sharp - 2)L/\lambda$. We claim that the total size of small jobs assigned to any machine is at least $(s_i^\sharp - 2)L/\lambda$ in the end. Suppose, to the contrary, that there exists a machine M_τ such that the total size of small jobs assigned to it is less than $(s_i^\sharp - 2)L/\lambda$, which implies that no small jobs is remaining after assigning small jobs to M_τ. Since the size of small job in I is at most L/λ, the total size of small jobs assigned to any machine is bounded by $(s_i^\sharp - 1)L/\lambda$. Thus, we have

$$\sum_{i=1}^{\tau} p(J_{[i]}^S) < \frac{L}{\lambda} \sum_{i=1}^{\tau} \left(s_i^\sharp - 1\right) = \frac{L}{\lambda} \sum_{i=1}^{\tau} s_i^\sharp - \tau \frac{L}{\lambda} \le \frac{L}{\lambda} \sum_{i=1}^{\tau} \left\lceil \frac{p(J_{[i]}^S)}{L/\lambda} \right\rceil - \tau \frac{L}{\lambda} \le \sum_{i=1}^{\tau} p(J_{[i]}^S)$$

a contradiction. Hence, the lemma holds. □

3 The Problem $P|GoS|C_{\min}$

Note that all jobs in the instance $I^\sharp(\lambda)$ have processing times of the form kL/λ^2, where $k \in \{\lambda, \lambda+1, \ldots, \lambda^2\}$. The jobs in the instance $I^\sharp(\lambda)$ can be represented

as a set $N = \{\overrightarrow{n^i} \mid \overrightarrow{n^i} = (n^i_\lambda, n^i_{\lambda+1}, \ldots, n^i_{\lambda^2}),\ i = 1, 2, \ldots, m\}$, where n^i_k denotes the number of jobs with machine index i whose processing times are equal to kL/λ^2. An assignment to a machine is a vector $\overrightarrow{v} = (v_\lambda, v_{\lambda+1}, \ldots, v_{\lambda^2})$, where v_k is the number of jobs of processing time kL/λ^2 assigned to that machine. The *processing time* of assignment $\overrightarrow{v}$, denoted as $l(\overrightarrow{v})$, is $\sum_{k=\lambda}^{\lambda^2}(v_k \cdot kL/\lambda^2)$. Let $\overrightarrow{n} = \sum_{i=1}^{m} \overrightarrow{n^i} = (n_\lambda, n_{\lambda+1}, \ldots, n_{\lambda^2})$. Denote by F the set of all possible assignment vectors with processing time between $L^\sharp/2$ and $2L^\sharp$, *i.e.*, $F = \{\overrightarrow{v} \mid L^\sharp/2 \le l(\overrightarrow{v}) \le 2L^\sharp \text{ and } \overrightarrow{v} \le \overrightarrow{n}\}$. By the fact that the processing times of all jobs are integer multiples of L/λ^2 and $L^\sharp \le (1 + 2/\lambda)L$, it is easy to see that $|F| \le (2\lambda^2 + 4\lambda)^{\lambda^2}$ holds. Denote by ψ_i the set of all feasible vectors that can be allocated to machine M_i, *i.e.*, $\psi_i = \{\overrightarrow{u} \mid \overrightarrow{u} \le \sum_{t=1}^{i} \overrightarrow{n^t}\}$. For every $\overrightarrow{u} \in \psi_i$, $1 \le i \le m$, we denote by $T(i, \overrightarrow{u})$ the value of optimum allocation of the jobs in $\overrightarrow{u}$ to first i machines that only uses assignments from the set F.

For each $\overrightarrow{u} \in \psi_i$, we compute $T(i, \overrightarrow{u})$ using the following recurrence:

Subroutine

Step 1 For $\overrightarrow{u} \in \psi_1$, originally set $T(1, \overrightarrow{u}) = l(\overrightarrow{u})$;
Step 2 For $\overrightarrow{u} \in \psi_i$, $i = 2, \ldots, m$, compute

$$T(i, \overrightarrow{u}) = \max\{\min\{l(\overrightarrow{v}), T(i-1, (\overrightarrow{u} - \overrightarrow{v}) \bigtriangledown \overrightarrow{n^i})\} \mid \overrightarrow{v} \in F \cap \psi_i\},$$

where $(\overrightarrow{u} - \overrightarrow{v}) \bigtriangledown \overrightarrow{n^i} = (\min\{u_\lambda - v_\lambda, n^i_\lambda\}, \min\{u_{\lambda+1} - v_{\lambda+1}, n^i_{\lambda+1}\}, \ldots, \min\{u_{\lambda^2} - v_{\lambda^2}, n^i_{\lambda^2}\})$.

Step 3 Output $T(m, \overrightarrow{n})$.

It is easy to check that the total running time of the preceding dynamic programming is $O(|F| \sum_{i=1}^{m} |\psi_i|)$. By the facts

$$|\psi_i| \le \prod_{k=\lambda}^{\lambda^2} \big(n_k + 1\big) \le n^{\lambda^2 - \lambda + 1} \text{ and } |F| \le \big(2\lambda^2 + 4\lambda\big)^{\lambda^2},$$

we get the following result.

Theorem 1. *For any fixed integer λ, an optimal solution for the transformed instance $I^\sharp(\lambda)$ of the problem $P|GoS|C_{min}$ is computed in $O(m(2\lambda^2+4\lambda)^{\lambda^2} n^{\lambda^2-\lambda+1})$.*

Now, we are ready to present our first polynomial time approximation scheme for the problem $P|GoS|C_{min}$. For any instance I of the problem $P|GoS|C_{min}$ and a positive number $0 < \epsilon < 1$, our approximation scheme is constructed in the following structural form:

Algorithm PTAS-1

Step 1. Set $\lambda = \lceil 7/\epsilon \rceil$, and construct the instance $I^\sharp(\lambda)$ from the instance I.
Step 2. Solve the instance $I^\sharp(\lambda)$ optimally, and denote the optimal allocation $\sum^\sharp$ with goal value $OPT^\sharp$.
Step 3. Transform the schedule $\sum^\sharp$ of the instance $I^\sharp(\lambda)$ into an allocation $\sum$ with goal value OUT for the original instance I as described in the proof of Lemma 2.

Now, let OPT denote the goal value in the optimum schedule of the instance I, we get the following result.

Theorem 2. *For any $\epsilon > 0$, the algorithm PTAS-1 can produce a schedule $\sum$ for the problem $P|GoS|C_{min}$, whose output value OUT satisfies $OUT \geq (1-\epsilon)OPT$, and the running time of PTAS-1 is $O(mn^{O(1/\epsilon^2)})$, where the hidden constant depends exponentially on $1/\epsilon$.*

Proof. By Lemma 1, we have $OPT^\sharp \geq OPT - L/\lambda$; and by Lemma 2, we have

$$OUT \geq \frac{\lambda}{1+\lambda} OPT^\sharp - \frac{2}{\lambda} L.$$

By Observation 2, we have $L \leq 2OPT$. Thus, we obtain

$$\begin{aligned} OUT &\geq \frac{\lambda}{1+\lambda}(OPT - \frac{L}{\lambda}) - \frac{2}{\lambda} L \\ &\geq OPT - \frac{1}{1+\lambda} OPT - \frac{1}{1+\lambda} L - \frac{2}{\lambda} L \\ &\geq OPT - \frac{1}{\lambda} OPT - \frac{1}{\lambda} L - \frac{2}{\lambda} L \\ &\geq OPT - \frac{7}{\lambda} OPT \geq (1-\epsilon) OPT. \end{aligned}$$

We now analyze the running time of the algorithm PTAS-1. The running time of Step 1 is $O(n)$; By Theorem 1, Step 2 can be implemented in time $O(mn^{O(1/\epsilon^2)})$; By Lemma 2, Step 3 can be implemented in time $O(n)$. Thus, the total running time of the algorithm PTAS-1 is $O(mn^{O(1/\epsilon^2)})$. □

4 The Problem $P_m|GoS|C_{\min}$

Now, we study a special version $P_m|GoS|C_{min}$ of the problem $P|GoS|C_{min}$, where the number m of machines is fixed. In this version, an important observation is that the number of jobs needed to allocate in the instance $I^\sharp(\lambda)$ is bounded by a constant. As the processing time of any job in the instance $I^\sharp(\lambda)$ is at least L/λ, and by Observation 3 and the fact $L^\sharp \leq (1+2/\lambda)L$, the number of jobs needed in $\overrightarrow{n^i}$ is at most $(2\lambda+4)m$. Hence, the total number of jobs needed in the instance $I^\sharp(\lambda)$ is at most $(2\lambda+4)m^2$, which is a constant. By simply enumerating all possible assignments and selecting the maximum one, we can get the optimal solution for the transformed instance $I^\sharp(\lambda)$ in time $O(m^{(2\lambda+4)m^2}) = O(1)$, where m is a fixed constant. Therefore, for the problem $P_m|GoS|C_{min}$, Step 2 in the algorithm PTAS-1 can be executed in $O(1)$. With the arguments similar to the proof of Theorem 2, we can prove the following result.

Theorem 3. *For any $\epsilon > 0$, the algorithm PTAS-1 can produce a schedule $\sum$ for the problem $P_m|GoS|C_{min}$, whose output value satisfies $OUT \geq (1-\epsilon)OPT$, and the running time is $O(n)$, where the hidden constant depends exponentially on $1/\epsilon$ and m.*

To obtain an FPTAS for the problem $P_m|GoS|C_{min}$, we must avoid the exponential dependence on $1/\epsilon$. Now, we will present a dynamic programming method at Step 2 in the algorithm PTAS-1 to solve $I^\sharp(\lambda)$ optimally, in the running time $n(2\lambda^2+4\lambda)^m = O(n)$.

In the dynamic programming method at Step 2 in the algorithm PTAS-1, we will store certain information for certain schedules for the first k jobs in the instance $I^\sharp(\lambda)$. Every such schedule is encoded by an m-dimensional vector $(C_1, C_2, \ldots, C_m)$, where C_i specifies the overall processing times of all jobs assigned to machines M_i for $i = 1, 2, \ldots, m$. Originally, the state space ψ_0 contains only one element $(0, 0, \cdots, 0)$. For $k \geq 1$, by Observation 3, it is sufficient for the state space ψ_k only to contain all m-dimensional vectors with $C_i \leq 2L^\sharp$ $(i = 1, 2, \cdots, m)$ to the first k job, and then the state space ψ_k is derived from the state space ψ_{k-1}. Assume that $J_k^\sharp \in J_{[t]}$. For every vector $(C_1, C_2, \cdots, C_m) \in \psi_{k-1}$ and for every coordinate $i \geq t$, we put the vector $(C_1, C_2, \cdots, C_i + p_k^\sharp, \cdots, C_m)$ which satisfies $C_i + p_k^\sharp \leq 2L^\sharp$ into ψ_k. In the end, we run through all vectors in the final state space and output the schedule that maximize the minimum coordinate.

Note that the processing times of all jobs in the transformed instance $I^\sharp(\lambda)$ are integer multiples of L/λ^2. Hence, $\mid \psi_k \mid \leq (2\lambda^2+4\lambda)^m = O(1)$. Therefore, the running time of the dynamic programming algorithm is $O(n)$, which implies that Step 2 in the algorithm PTAS-1 can be executed in $O(n)$ for the problem $P_m|GoS|C_{min}$. With the arguments similar to the proof of Theorem 2, we can prove the following result.

Theorem 4. *For any $\epsilon > 0$, the algorithm PTAS-1 can produce a schedule $\sum$ for the problem $P_m|GoS|C_{min}$, whose output value satisfies $OUT \geq (1-\epsilon)OPT$, and the running time is $O(n)$, where the hidden constant depends exponentially on m.*

5 The Problem $P|GoS_k|C_{\min}$

In this section, we study the problem $P|GoS_k|C_{min}$ as a special version of the problem $P|GoS|C_{min}$, where the number of GoS levels is bounded by k. Without loss of generality, we assume that $g(M_i)$, $g(J_j) \in \{1, 2, \ldots, k\}$ for $1 \leq i \leq n$ and $1 \leq j \leq m$. For a given instance $I = (\mathcal{J}, \mathcal{M}; p, g)$ of the problem $P|GoS_k|C_{min}$, let m_i denote the number of machines with GoS level i. Clearly, $m = \sum_{i=1}^{k} m_i$. Let $J_{[i]} = \{J_j \mid g(J_j) = i\}$ denote the set of jobs with GoS level i. Obviously, $J = \cup_{i=1}^{k} J_{[i]}$. Let

$$L = \min\left\{ \frac{\sum_{i=t}^{k} p(J_{[i]})}{\sum_{i=t}^{k} m_i} \;\middle|\; t = 1, 2, \ldots, k \right\}$$

denote the minimum average load of the machines with GoS level at least t, where $p(J_{[i]}) = \sum_{J_j \in J_{[i]}} p_j$. Clearly, $L \geq OPT$, where OPT denotes the optimal value. Observation 1 and Observation 2 still hold for the instance I of the problem $P|GoS_k|C_{min}$.

For any instance $I = (\mathcal{J}, \mathcal{M}; p; g)$ of the problem $P|GoS_k|C_{min}$ and an integer constant λ, we construct a *transformed instance* $I^\sharp(\lambda)$ as follows:

• Partition the jobs into two subsets J^B and J^S, where $J^B = \{J_j \mid p_j > L/\lambda\}$ and $J^S = \{J_j \mid p_j \leq L/\lambda\}$.

• For every job J_j in J^B, the transformed instance $I^\sharp(\lambda)$ contains a *corresponding job* $J_j^\sharp$ with processing time $p_j^\sharp = \lceil \frac{p_j}{L/\lambda^2} \rceil \frac{L}{\lambda^2} \leq \frac{\lambda+1}{\lambda} p_j$.

The GoS level of job $J_j^\sharp$ is $g(J_j)$.

• For $i = 1, 2, \ldots, k$, let $J_{[i]}^S = J^S \cap J_{[i]}$. The instance $I^\sharp(\lambda)$ contains a number of *auxiliary jobs*, each of processing time L/λ. Specifically, for $i = 1, 2, \ldots, m$, let $J_{[i]}^A$ be a set of k_i auxiliary job with GoS level i, where

$$k_i = \left\lceil \frac{\sum_{J_j \in J_{[i]}^S} p_j}{\frac{L}{\lambda}} \right\rceil.$$

• The machines remain the same as in instance I.

Note that in the instance $I^\sharp(\lambda)$, the processing times of all jobs are integer multiples of L/λ^2. Let

$$L^\sharp = \min \left\{ \frac{\sum_{i=t}^{k} \left(p^\sharp(J_{[i]}^B) + p^\sharp(J_{[i]}^A) \right)}{\sum_{i=t}^{k} m_i} \;\middle|\; t = 1, 2, \ldots, k \right\}$$

denote the minimum average load of the machines with GoS level at least t in instance $I^\sharp(\lambda)$. As the similar arguments in the section 2, we have $L \leq L^\sharp \leq (1 + 2/\lambda)L$.

It is easy to see that Observation 3, Lemma 1 and Lemma 2 still hold. Note that all jobs in the instance $I^\sharp(\lambda)$ have processing times of the form tL/λ^2, where $t \in \{\lambda, \lambda + 1, \ldots, \lambda^2\}$. The jobs in the instance $I^\sharp(\lambda)$ can be represented as a set $N = \{\overrightarrow{n^i} \mid \overrightarrow{n^i} = (n_\lambda^i, n_{\lambda+1}^i, \ldots, n_{\lambda^2}^i),\ i = 1, 2, \ldots, k\}$, where n_t^i denotes the number of jobs with GoS level i whose processing times are equal to tL/λ^2. An assignment to a machine is a vector $\overrightarrow{v} = (v_\lambda, v_{\lambda+1}, \ldots, v_{\lambda^2})$, where v_t is the number of jobs of processing times tL/λ^2 assigned to that machine. The *processing time* of assignment $\overrightarrow{v}$, denoted $l(\overrightarrow{v})$, is $\sum_{t=\lambda}^{\lambda^2}(v_t \cdot tL/\lambda^2)$. Denote by F the set of all possible assignment vectors with processing time between $L^\sharp/2$ and $2L^\sharp$, *i.e.*, $F = \{\overrightarrow{v} \mid L^\sharp/2 \leq l(\overrightarrow{v}) \leq 2L^\sharp; \overrightarrow{v} \leq \overrightarrow{n} = \sum_{i=1}^{k} \overrightarrow{n^i}\}$. It is easy to verify that $|F| \leq (2\lambda^2 + 4\lambda)^{\lambda^2}$ holds as in Section 3. Denote by ψ_i the set of all feasible vectors with processing time less than $2L^\sharp$ that can be allocated to the machine with GoS level i, *i.e.*, $\psi_i = \{\overrightarrow{u} \mid \overrightarrow{u} \leq \sum_{t=i}^{k} \overrightarrow{n^t}; L^\sharp/2 \leq l(\overrightarrow{u}) \leq 2L^\sharp\}$. For every $\overrightarrow{v} \in F$, we define $F^{\overrightarrow{v}} = \{\overrightarrow{u} \in F \mid l(\overrightarrow{u}) \geq l(\overrightarrow{v})\}$ and $\psi_i^{\overrightarrow{v}} = \{\overrightarrow{u} \in \psi_i \mid l(\overrightarrow{u}) \geq l(\overrightarrow{v})\}$. For each vector $\overrightarrow{u} \in \psi_i^{\overrightarrow{v}}$, denote by $x_i^{\overrightarrow{u}}$ be the numbers of machines with GoS level i that are assigned $\overrightarrow{u}$ in $\psi_i^{\overrightarrow{v}}$. For every $\overrightarrow{v} \in F$, we construct an integer linear programming ILP($\overrightarrow{v}$) with arbitrary objective function, and that the corresponding constraints are:

$$
\begin{aligned}
& ILP\ (\overrightarrow{v}) \\
\sum_{\overrightarrow{u} \in \psi_i^{\overrightarrow{v}}} x_i^{\overrightarrow{u}} &= m_i; \quad i = 1, 2, \ldots, k \\
\sum_{i=t}^{k} \sum_{\overrightarrow{u} \in \psi_i^{\overrightarrow{v}}} x_i^{\overrightarrow{u}}\, \overrightarrow{u} &\leq \sum_{i=t}^{k} \overrightarrow{n^t}; \quad t = 1, 2, \ldots, k \\
x_i^{\overrightarrow{u}} &\in Z^+ \cup \{0\} \qquad \forall \overrightarrow{u} \in \psi_i^{\overrightarrow{v}}
\end{aligned}
$$

here, the first set of constraints guarantee that each machine is assigned at least one vector (a set of jobs), and the second set of constraints ensure that no jobs is used in more than once. For every $\overrightarrow{v} \in F$, the number of variables in the integer linear programming is

$$\sum_{i=1}^{k} |\psi_i^{\overrightarrow{v}}| \leq k|F| \leq k(2\lambda^2 + 4\lambda)^{\lambda^2},$$

and that the number of constrains is

$$k + k(\lambda^2 - \lambda + 1) + \sum_{i=1}^{k} |\psi_i^{\overrightarrow{v}}| \leq k\Big(\lambda^2 - \lambda + 2 + (2\lambda^2 + 4\lambda)^{\lambda^2}\Big).$$

Both values are constants, as the two numbers λ and k are fixed constants, which do not depend on the input.

To solve the integer linear programming ILP($\overrightarrow{v}$), by utilizing Lenstra's algorithm in [11] whose running time is exponential in the dimension of the program but polynomial in the logarithms of the coefficients, we can decide whether the integer linear programming ILP($\overrightarrow{v}$) has a feasible solution in time $O(log^{O(1)}n)$, where the hidden constant depends exponentially on λ and k. And since the integer linear programming ILP($\overrightarrow{v}$) can be constructed in $O(n)$ time, we can find the optimal value $OPT^\sharp = \max\{l(\overrightarrow{v})|$ ILP($\overrightarrow{v}$) has a feasible solution of the instance $I^\sharp(\lambda)\}$ in time $O(|F|(log^{O(1)}n + n)) = O(n)$. Hence, the following theorem holds.

Theorem 5. *For any fixed integer λ, an optimal solution for the transformed instance $I^\sharp(\lambda)$ of the problem $P|GoS_k|C_{min}$ can be computed in time $O(n)$, where the hidden constant depends exponentially on λ and k.*

By Theorem 5, for the problem $P|GoS_k|C_{min}$, Step 2 in the algorithm PTAS-1 can be executed in time $O(n)$. Similar to Theorem 2, we can prove the following result.

Theorem 6. *For any $\epsilon > 0$, the algorithm PTAS-1 can produce a schedule $\sum$ for the problem $P|GoS_k|C_{min}$, whose output value OUT satisfies $OUT \geq (1-\epsilon)OPT$, and the running time is $O(n)$, where the hidden constant depends exponentially on $1/\epsilon$ and k.*

6 Conclusion

In this paper, we respectively present four approximation schemes for the max-min allocation problem under a grade of service provision and its two special versions, and the running time of the algorithm PTAS-1 for the problem $P|GoS|C_{min}$ is $O(mn^{O(1/\epsilon^2)})$, where $\epsilon > 0$ is any real number.

In future study, it is desirable to design a new PTAS with running time $O(n)$ for the problem $P|GoS|C_{min}$ or to extend the approximation schemes in the current paper to the general goals as in [1,7]. The most challenging study is to design a PTAS for the general goals with running time $O(n)$ as in [1].

References

1. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation Schemes for Scheduling on Parallel Machines. Journal of Scheduling 1, 55–66 (1998)
2. Asadpour, A., Feige, U., Saberi, A.: Santa Claus Meets Hypergraph Matchings. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 10–20. Springer, Heidelberg (2008)
3. Asadpour, A., Saberi, A.: An Approximation Algorithm for Max-Min Fair Allocation of Indivisible Goods. In: Proc. of ACM-SIAM Symposium on the Theory of Computation (STOC), pp. 114–121 (2007)
4. Bansal, N., Sviridenko, M.: The Santa Claus Problem. In: Proc. of ACM-SIAM Symposium on the Theory of Computation (STOC), pp. 31–40 (2006)
5. Bezakova, I., Dani, V.: Allocating Indivisible Goods. SIGecom Exchanges 5(3), 11–18 (2005)
6. Chakrabarty, D., Chuzhoy, J., Khanna, S.: On Allocating Goods to Maximize Fairness (manuscript, 2009)
7. Epstein, L., Sgall, J.: Approximation Schemes for Scheduling on Uniformly Related and Identical Parallel Machines. Algorithmica 39(1), 43–57 (2004)
8. Feige, U.: On Allocations that Maximize Fairness. In: ACM-SIAM Annual Symposium on Discrete Algorithms (SODA), pp. 287–293 (2008)
9. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. Annals of Discrete Mathematics 5, 287–326 (1979)
10. Hwang, H.C., Chang, S.Y., Lee, K.: Parallel Machine Scheduling under a Grade of Service Provision. Computers and Operations Research 31, 2055–2061 (2004)
11. Lenstra, H.W.: Integer Programming with a Fixed Number of Variables. Mathematics of Operations Research 8, 538–548 (1983)
12. Lenstra, J.K., Shmoys, D.B., Tardos, E.: Approximation Algorithms for Scheduling Unrelated Parallel Machines. Mathematical Programming, Series A 46(2), 259–271 (1990)
13. Ou, J., Leung, J.Y.-T., Li, C.L.: Scheduling Parallel Machines with Inclusive Processing Set Restrictions. Naval Research Logistics 55(4), 328–338 (2008)
14. Schuurman, P., Woeginger, G.J.: Polynomial Time Approximation Algorithms for Machine Scheduling: Ten Open Problems. Journal of Scheduling 2, 203–213 (1999)
15. Woeginger, G.J.: When does a Dynamic Programming Formulation Guarantee the Existence of a Fully Polynomial Time Approximation Scheme (FPTAS)? INFORMS Journal on Computing 12, 57–75 (2000)

A Linear Time Algorithm for Computing the Most Reliable Source on a Tree with Faulty Vertices

Wei Ding[1] and Guoliang Xue[2]

[1] Zhejiang Water Conservancy and Hydropower College,
Hangzhou, Zhejiang 310018, China
dingweicumt@163.com
[2] Department of Computer Science and Engineering, Arizona State University,
Tempe, AZ 85287-8809, USA
xue@asu.edu

Abstract. Given an unreliable communication network, we seek for determining a vertex from the network, the expected number of vertices which connects to is maximum. Such vertex is named the most reliable source (MRS) on the network. The communication failures may occur to links or vertices of the network. The case was generally studied, where no failure happens to each vertex and each link has an independent operational probability. Practically, failures frequently happen to the vertices, including the transmitting fault and receiving fault. Recently, another case is proposed, where each link is steady and each vertex has an independent transmitting probability and receiving probability, and an $\mathcal{O}(n^2)$ time algorithm is presented for computing the MRS on such tree networks with n vertices. In this paper, we propose a faster algorithm for this case, whose time complexity is $\mathcal{O}(n)$.

Keywords: Most reliable source, Transmitting probability, Receiving probability.

1 Introduction

A computer network or communication network is often modeled as an undirected graph $G = (V, E)$, where n vertices represent processing vertices or switching elements and m edges represent communication links [3]. For any given vertex pair u and v of the network, the communication between u and v is composed of every communication link on the path connecting u with v. Failures may happen to vertices or communication links [4,5,6,8,11]. As networks grow in size, they become increasingly vulnerable to the failures of some links and/or vertices. In past decades, a large number of network reliable problems have arisen rapidly and been extensively studied, others referring to [1,2,7,9,10].

In unreliable communication networks, the vertex, the expected number of vertices which connects to is maximum, is named the *most reliable source* (MRS) on the network. The problem for determining the MRS on given network is one of

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 14–23, 2009.

the network reliable problems, which has caused many experts' interests. Several articles studied the case where no failure happens to vertices and each link has an independent operational probability. For tree networks, Melachrinoudis and Helander [8] gave an $\mathcal{O}(n^2)$ time algorithm and Xue [11] presented an $\mathcal{O}(n)$ time algorithm separately. Later, Colbourn and Xue [5] proposed a linear time algorithm for computing the MRS on series-parallel graphs. Recently, Ding [6] concentrated on another case where no failure happens to links and each vertex has an independent probability of making faults, and proposed an $\mathcal{O}(n^2)$ time algorithm. In [6], the probability of making faults of each vertex includes two sides, one is *transmitting fault* and the other is *receiving fault*. For any vertex u and v of the network, the communication link from u to v is correct if and only if no failure happens to the transmitting process of u and no failure happens to the receiving process of v. In this paper, we continue to study such case and design a faster algorithm, whose time complexity is $\mathcal{O}(n)$.

The rest of this paper is organized as follows. In Sect. 2, we present some definitions and fulfil some preliminary works. In Sect. 3, we take effect of the results in Sect. 2 to design a linear time algorithm for computing the MRS on a tree network where each link is steady and each vertex has an independent transmitting probability and receiving probability. In Sect. 4, we present a numerical example to illustrate our algorithm. In Sect. 5, some suggestions on future research are concluded.

2 Definitions and Preliminaries

In the rest of this paper, we let $T = (V, E, P)$ denote a weighted tree network with n vertices, where T is regarded as a rooted tree with root vertex $\mathfrak{R}$. We let L denote the set composed of all leaves of T. For each $v_i \in V$, it is associated with a pair of weights $\langle \underline{p}(v_i), \overline{p}(v_i) \rangle$ of P, where $\underline{p}(v_i)$ denotes the probability that v_i transmits messages correctly and $\overline{p}(v_i)$ denotes the probability that v_i receives messages correctly. For convenience, we call $\underline{p}(v_i)$ *transmitting probability* of v_i and call $\overline{p}(v_i)$ *receiving probability* of v_i.

For each edge $\{v_i, v_j\} \in E$, there are two arcs associated with $\{v_i, v_j\}$. We let (v_i, v_j) and (v_j, v_i) denote the directed edge from v_i to v_j and from v_j to v_i respectively. The operational probability of (v_i, v_j) is denoted by $p(v_i, v_j)$. Considering the vertex operational probabilities (including the transmitting probability and the receiving probability) are independent and the operational probability of (v_i, v_j) is determined by the transmitting probability of v_i and the receiving probability of v_j, the operational probability of (v_i, v_j) can be computed by

$$p(v_i, v_j) = \underline{p}(v_i) \cdot \overline{p}(v_j). \tag{1}$$

Similarly, $p(v_j, v_i) = \underline{p}(v_j) \cdot \overline{p}(v_i)$. In general, $p(v_i, v_j)$ is distinct with $p(v_j, v_i)$, so that $\{v_i, v_j\}$ is associated with two nonsymmetric arcs on weight (see Fig. 1).

The arc set composed of $2n-2$ arcs is denoted by A, i.e. $A = \{(v_i, v_j), (v_j, v_i) : \{v_i, v_j\} \in E\}$. For each $(v_i, v_j) \in A$, the arc operational probability of (v_i, v_j) is computed by (1). Moreover, we introduce P^* to denote the set of arc operational

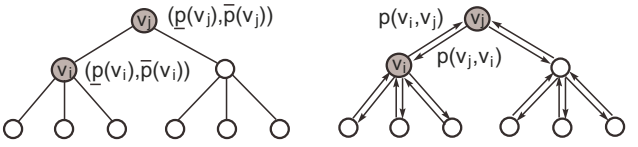


Fig. 1. Left-hand undirected rooted tree $T = (V, E, P)$ where P denotes the set of vertex operational probability pair is transformed into right-hand directed rooted tree $T^* = (V, A, P^*)$ where P^* denotes the set of arc operational probability

probability, consisting of all operational probabilities of all arcs of A, i.e. $P^* = \{p(v_i, v_j) : (v_i, v_j) \in A\}$. As a result, we construct a new directed rooted tree $T^* = (V, A, P^*)$ based on $T = (V, E, P)$ (see Fig. 1).

Lemma 1. *To compute the MRS on an undirected tree $T = (V, E, P)$ is equivalent to compute the MRS on its corresponding directed tree $T^* = (V, A, P^*)$.*

It is clear that $T = (V, E, P)$ is an undirected tree where failures only happen to vertices and $T^* = (V, A, P^*)$ is a directed tree where failures only happen to links. Lemma 1 states that to compute the MRS on T where all links are steady and failures only happen to vertices is equivalent to compute the MRS on its corresponding T^* where each vertex has no failure and failures only happen to links. Consequently, our main task is how to compute the MRS on a directed rooted tree network.

For each $v_i \in V$, we use $\mathcal{C}(v_i)$ to denote the set which consists of all children of v_i. In the rest of this paper, for convenience, we always suppose that f_i denotes the parent of v_i (except the root vertex $\mathfrak{R}$). Each f_i corresponds to one exact vertex of V, which is signed by subscript $i(f)$ (i.e. $v_{i(f)}$).

Let $T_\alpha(v_i)$ to denote the subtree of T rooted at v_i. The vertex set of $T_\alpha(v_i)$ is denoted by $V_\alpha(v_i)$ and the vertex set composed of all vertices outside of $T_\alpha(v_i)$ is denoted by $V_\beta(v_i)$. We introduce $U \uplus W$ to denote $U \cup W$ provided $U \cap W = \emptyset$. Hence $V = V_\alpha(v_i) \uplus V_\beta(v_i)$.

Given a vertex $v_i \in V$, it is evident that $V_\alpha(v_i) = \{v_i\}$ if v_i is a leaf of T and $V_\beta(v_i) = \emptyset$ if v_i is the root of T. Otherwise, we presents the decomposition scheme of $V_\alpha(v_i)$ and $V_\beta(v_i)$ in detail by Lemma 2 (see Fig. 2).

Lemma 2. *Given a vertex v_i of V, we obtain*
(i) *If $v_i \in V - L$, then $V_\alpha(v_i)$ can be decomposed as*

$$V_\alpha(v_i) = \left(\biguplus_{v_k \in \mathcal{C}(v_i)} V_\alpha(v_k) \right) \uplus \{v_i\}. \tag{2}$$

(ii) *If $v_i \in V - \{\mathfrak{R}\}$, then $V_\beta(v_i)$ can be decomposed as*

$$V_\beta(v_i) = \left(\biguplus_{v_t \in \mathcal{C}(f_i) - \{v_i\}} V_\alpha(v_t) \right) \uplus V_\beta(f_i) \uplus \{f_i\}. \tag{3}$$

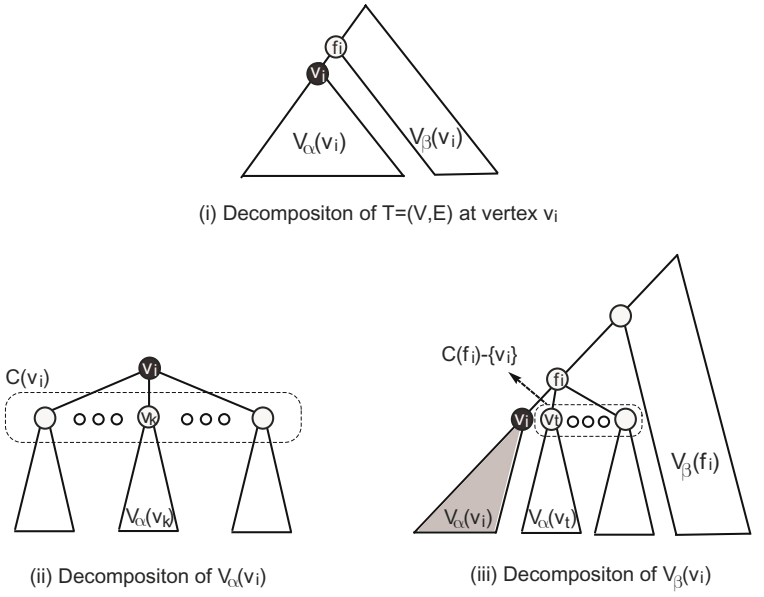


Fig. 2. Illustration of decomposing V at vertex v_i colored with black

By Lemma 2, V is recursively decomposed from top root to bottom leaves, of which our algorithm for computing the MRS on T^* is designed on basis.

Definition 1. *Given directed rooted tree $T^* = (V, A, P^*)$ based on $T = (V, E, P)$, let $\mathcal{X}(v_i)$ denote the expected number of vertices in $V_\alpha(v_i)$ which v_i can connect to, and let $\mathcal{Y}(v_i)$ denote the expected number of vertices in $V_\beta(v_i)$ to which v_i can connect.*

Theorem 1. *Given directed rooted tree $T^* = (V, A, P^*)$ based on $T = (V, E, P)$, the expected number of vertices to which v_i connects is $\mathcal{X}(v_i) + \mathcal{Y}(v_i)$.*

Proof. For each $v_i \in V$, the vertices to which v_i connects are the union of vertices in $V_\alpha(v_i)$ and those in $V_\beta(v_i)$ to which v_i connects due to $V = V_\alpha(v_i) \uplus V_\beta(v_i)$. By Definition 1, the expected number of vertices in $V_\alpha(v_i)$ which v_i connects to is $\mathcal{X}(v_i)$ and the expected number of vertices in $V_\beta(v_i)$ which v_i connects to is $\mathcal{Y}(v_i)$. □

Based on Theorem 1, our critical task is to compute $\mathcal{X}(v_i)$ and $\mathcal{Y}(v_i)$ for each $v_i \in V$ in order to compute the expected number of vertices which v_i connects to. According to Definition 1, we investigate $\mathcal{X}(v_i) = 1$ for each $v_i \in L$ and $\mathcal{Y}(\Re) = 0$. Otherwise, the following Theorem 2 shows the formulas to compute $\mathcal{X}(v_i)$ and $\mathcal{Y}(v_i)$ recursively (see Fig. 3).

Theorem 2. *For each vertex v_i in $V - L$, we have*

$$\mathcal{X}(v_i) = 1 + \sum_{v_k \in \mathcal{C}(v_i)} \mathcal{X}(v_k) \cdot p(v_i, v_k). \tag{4}$$

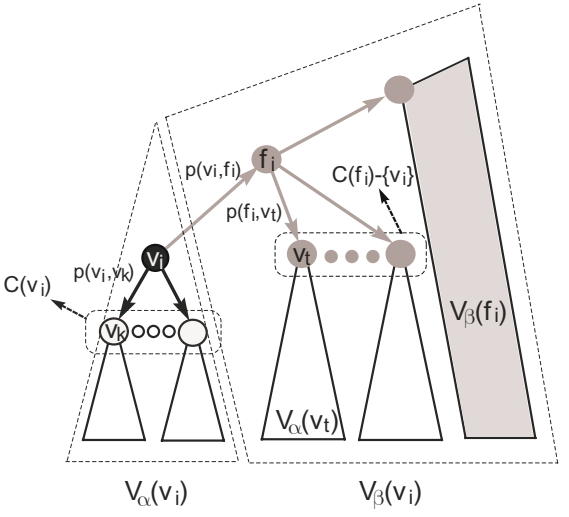


Fig. 3. An illustration of computing the expected number of vertices which v_i connects to for Theorem 1. It is also an illustration of computing $\mathcal{X}(v_i)$ and $\mathcal{Y}(v_i)$ for Theorem 2.

For each vertex v_i in $V - \{\mathfrak{R}\}$, we have

$$\mathcal{Y}(v_i) = \left(\mathcal{Y}(f_i) + \sum_{v_t \in \mathcal{C}(f_i) - \{v_i\}} \mathcal{X}(v_t) \cdot p(f_i, v_t) + 1 \right) \cdot p(v_i, f_i). \tag{5}$$

Proof. For each $v_i \in V - L$, (2) implies that the vertices in $V_\alpha(v_i)$ that v_i connects to are the union of vertices in each $V_\alpha(v_k)$ where $v_k \in \mathcal{C}(v_i)$ and v_i itself. It is certain that v_i connects to itself. The expected number of vertices in each $V_\alpha(v_k)$ to which v_i connects is $\mathcal{X}(v_k) \cdot p(v_i, v_k)$ by Definition 1.

For each $v_i \in V - \{\mathfrak{R}\}$, (3) makes it clear that v_i via f_i connects to all vertices in $V_\beta(v_i)$ and the vertices in $V_\beta(v_i)$ are from the union of vertices in each $V_\alpha(v_t)$ where $v_t \in \mathcal{C}(f_i) - \{v_i\}$ and vertices in $V_\beta(f_i)$ and f_i itself. It is certain that f_i connects to itself. The expected number of vertices in each $V_\alpha(v_t)$ to which f_i connects is $\mathcal{X}(v_t) \cdot p(f_i, v_t)$, and those in $V_\beta(f_i)$ is $\mathcal{Y}(f_i)$ by Definition 1. □

3 The Linear Time Algorithm

Based on Theorem 1, $\mathcal{X}(v_i) + \mathcal{Y}(v_i)$ is the expected number of vertices that v_i connects to. The maximum one amongst all such values corresponds to the MRS on T^*. Therefore, we can determine the MRS on T^* with n vertices by $\mathcal{O}(n)$ time provided that we can compute the values of $\mathcal{X}(v_i)$ and $\mathcal{Y}(v_i)$ for all $v_i \in V$ by $\mathcal{O}(n)$ time. In this section, we propose such an algorithm. At first, we introduce thr important Lemma 3 as follows.

Lemma 3. *For each vertex v_i in $V - \{\mathfrak{R}\}$, we have*

$$\mathcal{Y}(v_i) = (\mathcal{Y}(f_i) + \mathcal{X}(f_i) - \mathcal{X}(v_i)\, p(f_i, v_i)) \cdot p(v_i, f_i). \tag{6}$$

Proof. Based on definition of $\mathcal{X}(v_i)$ and $\mathcal{Y}(v_i)$ and referring to Fig. 3, we conclude from (5) that for each vertex v_i in $V - \{\mathfrak{R}\}$

$$\begin{aligned}
\mathcal{Y}(v_i) &= \left(\mathcal{Y}(f_i) + \sum_{v_t \in \mathcal{C}(f_i) - \{v_i\}} \mathcal{X}(v_t)\, p(f_i, v_t) + 1\right) \cdot p(v_i, f_i) \\
&= \left(\mathcal{Y}(f_i) + \sum_{v_t \in \mathcal{C}(f_i)} \mathcal{X}(v_t)\, p(f_i, v_t) - \mathcal{X}(v_i)\, p(f_i, v_i) + 1\right) \cdot p(v_i, f_i) \\
&= \left(\mathcal{Y}(f_i) + \left(1 + \sum_{v_t \in \mathcal{C}(f_i)} \mathcal{X}(v_t)\, p(f_i, v_t)\right) - \mathcal{X}(v_i)\, p(f_i, v_i)\right) \cdot p(v_i, f_i) \\
&= \big(\mathcal{Y}(f_i) + \mathcal{X}(f_i) - \mathcal{X}(v_i)\, p(f_i, v_i)\big) \cdot p(v_i, f_i).
\end{aligned}$$

□

Lemma 3 makes it clear that the value of $\mathcal{Y}(v_i)$ can be computed by $\big(\mathcal{Y}(f_i) + \mathcal{X}(f_i) - \mathcal{X}(v_i)p(f_i, v_i)\big) \cdot p(v_i, f_i)$ after the values of $\mathcal{X}(v_i), \mathcal{X}(f_i), \mathcal{Y}(f_i)$ have been obtained. In fact, we can apply (4) to compute recursively each value of $\mathcal{X}(v_i)$ from bottom leaves up to top root among T^* (see Step_2 of algorithm DRMRS), beginning with $\mathcal{X}(v_i) = 1$ for all $v_i \in L$. On the other hand, we can apply (6) to compute recursively each value of $\mathcal{Y}(v_i)$ from top root down to bottom leaves amongst T^* (see Step_3 of algorithm DRMRS), beginning with $\mathcal{Y}(\mathfrak{R}) = 0$.

For simplicity of presentation, we let $\mathcal{H}(T^*)$ denote the height of T^* and let h denote the variable of current height. Let V_h denote the set composed of all vertices on the h-th level of T^*. Specially, $V_{\mathcal{H}(T^*)} = \{\mathfrak{R}\}$. Let C_i denote the set composed of all children of v_i. Hence

$$V = \biguplus_{h=1}^{\mathcal{H}(T^*)} V_h \qquad \text{and} \qquad V_{h-1} = \biguplus_{v_i \in V_h} C_i,\ h = 2, \ldots, \mathcal{H}(T^*)$$

Furthermore, it yields that

$$L = \biguplus_{h=1}^{\mathcal{H}(T^*)} \{V_h \cap L\} \tag{7}$$

and

$$V - L = \biguplus_{h=1}^{\mathcal{H}(T^*)} \Big\{V_h \cap (V - L)\Big\}. \tag{8}$$

Then it is not hard to be verified that

$$\sum_{h=1}^{\mathcal{H}(T^*)} |V_h| = |V| \tag{9}$$

and

$$\sum_{v_i \in V - L} |C_i| = |E| \ . \tag{10}$$

For each $v_i \in V$, let $\mathbf{X}_i$ record the value of $\mathcal{X}(v_i)$ and $\mathbf{Y}_i$ record the value of $\mathcal{Y}(v_i)$, let $\mathbf{P}_{[i,j]}$ record the operational probability of (v_i, v_j). Accordingly, the value of $\mathcal{X}(f_i)$ is recorded by $\mathbf{X}_{i(f)}$ and $\mathcal{Y}(f_i)$ by $\mathbf{Y}_{i(f)}$. Initially, we set $\mathbf{X}_i$ to one and set $\mathbf{Y}_i$ to zero. Our algorithm DRMRS for computing $\mathbf{X}_i$ and $\mathbf{Y}_i$ is presented as follows.

Algorithm DRMRS

Input: The directed rooted tree $T^* = (V, A, P^*)$.
Output: The value of $\mathbf{X}_i$ and $\mathbf{Y}_i$ for each vertex v_i of V.

Step_1 {*Initializing the value of each* $\mathbf{X}_i$ *and* $\mathbf{Y}_i$. }
  for each vertex v_i of V **do**
    $\mathbf{X}_i := 1, \mathbf{Y}_i := 0$;
  end for
Step_2 { *Computing the value of each* $\mathbf{X}_i$. }
  for h from 1 up to $\mathcal{H}(T^*)$ **do**
    for each $v_i \in V_h$ **do**
      if $v_i \in L$ **then**
        break;
      else
        for each $v_k \in C_i$ **do**
          $\mathbf{X}_i := \mathbf{X}_i + \mathbf{X}_k \cdot \mathbf{P}_{[i,k]}$;
        end for
      end if
    end for
  end for
Step_3 { *Computing the value of each* $\mathbf{Y}_i$. }
  for h from $\mathcal{H}(T^*)$ down to 1 **do**
    if $h = \mathcal{H}(T^*)$ **then**
      break;
    else
      for each $v_i \in V_h$ **do**
        $\mathbf{Y}_i := \left(\mathbf{Y}_{i(f)} + \mathbf{X}_{i(f)} - \mathbf{X}_i \cdot \mathbf{P}_{[i(f),i]}\right) \cdot \mathbf{P}_{[i,i(f)]}$;
      end for
    end if
  end for

Theorem 3. *Given the directed rooted tree network* $T^* = (V, A, P^*)$ *with* n *vertices, algorithm DRMRS can compute the MRS on* T^* *correctly, whose total time complexity is* $\mathcal{O}(n)$.

Proof. At first, the Step_1 of algorithm DRMRS initializes the value of $\mathbf{X}_i$ and $\mathbf{Y}_i$ for each $v_i \in V$, which occupies $\mathcal{O}(n)$ time.

Subsequently, the Step_2 of algorithm DRMRS computes the value of each $\mathbf{X}_i$ from leaves up to root. The value of $\mathbf{X}_i$ where $v_i \in V_h \cap L$ is just the initial value $\mathbf{X}_i = 1$. The value of $\mathbf{X}_i$ where $v_i \in V_h \cap (V - L)$ can be computed by $\mathbf{X}_i = 1 + \sum_{v_k \in C_i} \mathbf{X}_k \cdot \mathbf{P}_{[i,k]}$ due to (4) since all values of $\mathbf{X}_l$ where $v_l \in V_{h-1} =$

$\biguplus_{v_i \in V_h} C_i$ have been obtained in the previous loop. Thus, the time spent in computing the value of $\mathbf{X}_i$ for all $v_i \in V$ is

$$\sum_{h=1}^{\mathcal{H}(T^*)} \sum_{v_i \in V_h \cap L} \mathcal{O}(1) + \sum_{h=1}^{\mathcal{H}(T^*)} \sum_{v_i \in V_h \cap (V-L)} \sum_{v_k \in C_i} \mathcal{O}(1)$$
$$\overset{(7)}{=} \sum_{v_i \in L} \mathcal{O}(1) + \sum_{h=1}^{\mathcal{H}(T^*)} \sum_{v_i \in V_h \cap (V-L)} \mathcal{O}(|C_i|)$$
$$\overset{(8)}{=} \mathcal{O}(|L|) + \sum_{v_i \in V-L} \mathcal{O}(|C_i|) \overset{(10)}{=} \mathcal{O}(|L|) + \mathcal{O}(|E|) = \mathcal{O}(n).$$

Finally, the Step_3 of algorithm DRMRS computes the value of each $\mathbf{Y}_i$ from root down to leaves. The value of $\mathbf{Y}_i$ where $v_i \in V_{\mathcal{H}(T^*)}$ is just the initial value $\mathbf{Y}_i = 0$. The value of $\mathbf{Y}_i$ where $v_i \in V_h, h = 1, 2, \ldots, \mathcal{H}(T^*) - 1$ can be computed by $\mathbf{Y}_i = \left(\mathbf{Y}_{i(f)} + \mathbf{X}_{i(f)} - \mathbf{X}_i \cdot \mathbf{P}_{[i(f),i]}\right) \cdot \mathbf{P}_{[i,i(f)]}$ due to (6) as all values of $\mathbf{X}_p$ where $v_p \in V$ have been computed previously and all values of $\mathbf{Y}_q$ where $v_q \in V_{h+1}$ have been obtained in the previous loop. Thus, the time spent in computing the value of $\mathbf{Y}_i$ for all $v_i \in V$ is

$$\mathcal{O}(1) + \sum_{h=1}^{\mathcal{H}(T^*)-1} \sum_{v_i \in V_h} \mathcal{O}(1) = \mathcal{O}(1) + \sum_{h=1}^{\mathcal{H}(T^*)} \mathcal{O}(|V_h|) - \mathcal{O}(|V_{\mathcal{H}(T^*)}|) \overset{(9)}{=} \mathcal{O}(n).$$

The total time occupied by algorithm DRMRS to compute the value of $\mathbf{X}_i$ and $\mathbf{Y}_i$ for all $v_i \in V$ is the sum of time spent separately in Step_1 and Step_2 and Step_3. Therefore, the time complexity of algorithm DRMRS is $\mathcal{O}(n)$. □

4 An Example

In this section, we set an undirected unrooted tree network (shown as the left-hand graph in Fig. 4 where each vertex is associated with one vertex operational probability pair) as an example to illustrate our algorithm. Our objective is to compute its MRS. According to Lemma 1, our critical task is to determine the MRS on its corresponding directed tree network rooted at vertex M (shown as the right-hand graph in Fig. 4 where each arc is associated with one arc operational probability).

In Table 1, the column of i shows all vertices of given graph and the column of $\mathbf{E}_i$ shows the expected number of vertices which i connects to. At first, Step_1 of algorithm DRMRS initializes $\mathbf{X}_i = 1$ for $i = \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{G}, \mathrm{H}, \mathrm{I}, \mathrm{L}$ and $\mathbf{Y}_{\mathrm{M}} = 0$. During the Step_2, it computes the values of $\mathbf{X}_{\mathrm{F}}, \mathbf{X}_{\mathrm{J}}, \mathbf{X}_{\mathrm{K}}, \mathbf{X}_{\mathrm{M}}$ in turn. During the Step_3, it computes the values of $\mathbf{Y}_i, i = \mathrm{M}, \mathrm{J}, \mathrm{K}, \mathrm{L}, \mathrm{D}, \mathrm{E}, \mathrm{F}, \mathrm{G}, \mathrm{H}, \mathrm{I}, \mathrm{A}, \mathrm{B}, \mathrm{C}$ in turn. Finally, the values in the column of $\mathbf{E}_i$ are computed by adding the values in the column $\mathbf{X}_i$ to the values in the column $\mathbf{Y}_i$ by the coordinates. It is evident that vertex F is the MRS on given graph.

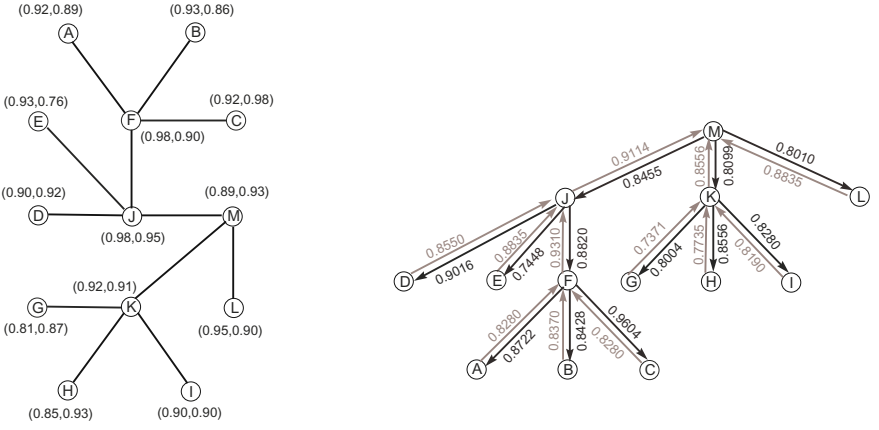


Fig. 4. Applying algorithm DRMRS to compute the MRS on given left-hand graph

Table 1. The data of $\mathbf{X}_i, \mathbf{Y}_i, \mathbf{E}_i$ for each vertex i on given graph

i	$\mathbf{X}_i$	$\mathbf{Y}_i$	$\mathbf{E}_i$
A	1.00000000000000	7.92455970350880	8.92455970350880
B	1.00000000000000	8.03530402202520	9.03530402202520
C	1.00000000000000	7.85153010350880	8.85153010350880
D	1.00000000000000	7.86567845522520	8.86567845522520
E	1.00000000000000	8.26640053706604	9.26640053706604
F	3.67540000000000	6.76752427960000	10.44292427960000
G	1.00000000000000	6.97517060521554	7.97517060521554
H	1.00000000000000	7.27692627460890	8.27692627460890
I	1.00000000000000	7.72758516135060	8.72758516135060
J	5.88810280000000	4.21312112424000	10.10122392424000
K	3.48400000000000	5.80044686892744	9.28444686892744
L	1.00000000000000	7.77487290412290	8.77487290412290
M	9.60108251740000	0.00000000000000	9.60108251740000

5 Conclusions

As we all know, the network failures may happen to vertices and/or network links. Some articles [5,8,11] seek for computing the MRS on given unreliable networks where no failures happen to vertices and each link has an independent operational probability. The paper [6] and this paper focus on how to compute the MRS on unreliable tree networks where each link is steady and each vertex has an independent transmitting probability and receiving probability. Actually, the network failures may happen to vertices and links simultaneously. We believe that the approach used in this paper can be applied to compute the MRS on such unreliable tree networks.

In past decades, a large number of network reliability problems [4] have arisen rapidly and been studied extensively. However, few papers were interested in the unreliable networks where failures happen to vertices. We also believe that our approach can be generalized to solve these reliability problems, particularly on unreliable tree networks with faulty vertices.

References

1. Ball, M.O., Lin, F.L.: A Reliability Model Applied to Emergency Service Vehicle Location. Oper. Res. 41(1), 18–36 (1993)
2. Ball, M.O., Provan, J.S., Shier, D.R.: Reliability Covering Problems. Networks 21(3), 345–357 (1991)
3. Bondy, J.A., Murty, U.S.R.: Graph Theory with Application. Macmillan, London (1976)
4. Colbourn, C.J.: The Combinatorics of Network Reliability. Oxford University Press, New York (1987)
5. Colbourn, C.J., Xue, G.L.: A Linear Time Algorithms for Computing the Most Reliable Source on a Series-Parallel Graph with Unreliable Edges. Theor. Comput. Sci. 209, 331–345 (1998)
6. Ding, W.: Most Reliable Source on an Unreliable Tree Network with Faulty Vertices (to appear)
7. Eiselt, H.A., Gendreau, M., Laporte, G.: Location of Facilities on a Network Subject to a Single-Edge Failure. Networks 22(3), 231–246 (1992)
8. Melachrinoudis, E., Helander, M.E.: A Single Facility Location Problem on a Tree with Unreliable Edges. Networks 27(3), 219–237 (1996)
9. Mirchandani, P.B., Odoni, A.R.: Locations of Medians on Stochastic Networks. Transport. Sci. 13, 85–97 (1979)
10. Nel, L.D., Colbourn, C.J.: Locating a Broadcast Facility in an Unreliable Network. INFOR. 28, 363–379 (1990)
11. Xue, G.L.: Linear Time Algorithms for Computing the Most Reliable Source on an Unreliable Tree Network. Networks 30(1), 37–45 (1997)

A 5/3-Approximation Algorithm for Joint Replenishment with Deadlines

Tim Nonner* and Alexander Souza

Department of Computer Science, Albert-Ludwigs-University of Freiburg,
Freiburg im Breisgau, Germany
tim.nonner@informatik.uni-freiburg.de

Abstract. The objective of the classical Joint Replenishment Problem (JRP) is to minimize ordering costs by combining orders in two stages, first at some retailers, and then at a warehouse. These orders are needed to satisfy demands that appear over time at the retailers. We investigate the natural special case that each demand has a deadline until when it needs to be satisfied. For this case, we present a randomized 5/3-approximation algorithm, which significantly improves the best known approximation ratio of 1.8 obtained by Levi and Sviridenko (APPROX'06). We moreover prove that JRP with deadlines is APX-hard, which is the first such inapproximability result for a variant of JRP. Finally, we extend the known hardness results by showing that JRP with linear delay cost functions is NP-hard, even if each retailer has to satisfy only three demands.

1 Introduction

The Joint Replenishment Problem (JRP) is one of the fundamental problems in inventory theory [1]. In this problem, we have a *warehouse* and some *retailers* $1, 2, \ldots, N$, which face demands D that appear over time. The time horizon is finite and partitioned into *periods* $1, 2, \ldots, T$. Therefore, each demand in D is defined by a tuple (i, t, h), where $i \in \{1, \ldots, N\}$ is its *release retailer*, $t \in \{1, \ldots, T\}$ is its *appearance period*, and $h : \mathbb{N}_0 \to \mathbb{R}^+$ is a monotonously increasing function that defines its *delay cost* (in the traditional setting, this function is linear). To satisfy arbitrary many demands in some period t, a retailer i needs to send an order to the warehouse in this period, which implies *retailer ordering cost* K_i. In this case, the warehouse orders in period t as well, which implies additional *warehouse ordering cost* K_0. However, this warehouse order can be used by arbitrary many retailer orders. More formally, a *schedule* for such an instance consists of a set of *warehouse orders* $S_0 \subseteq \{1, \ldots, T\}$ and a sequence of subsets of *retailer orders* $S_1, S_2, \ldots, S_N \subseteq S_0$ of the retailers $1, 2, \ldots, N$, respectively. This inclusion ensures that whenever a retailer orders, the warehouse orders as well. Consequently, the warehouse and retailer ordering costs of such a schedule are $K_0|S_0|$ and $\sum_{i=1}^{N} K_i|S_i|$, respectively. We assume that each demand $(i, t, h) \in D$

* Corresponding author. Supported by DFG research program No 1103 *Embedded Microsystems.*

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 24–35, 2009.

is satisfied by the first order $t' \in S_i$ with $t' \geq t$, whereas it then needs to be *delayed* during the periods $t, t+1, \ldots, t'-1$, which results in delay cost $h(t'-t)$. The objective of JRP is to find a schedule that satisfies all demands with minimum *cost*, that is, the sum of warehouse ordering costs, retailer ordering costs, and delay costs. Consequently, we need to balance ordering costs and delay costs. As defined above, we study JRP in the make-to-order variant [5], that is, each demand is satisfied after it appears. This variant is equivalent to the more common make-to-stock variant, where the demands are satisfied by goods which are ordered before the appearance of the demands, and then kept in inventory [14].

In the traditional setting treated in [3], the delay cost functions are linear, which corresponds to the well-known flow time objective from scheduling. We refer to this special case as JRP-L. However, in many scenarios, it is more reasonable to have fixed deadlines, which is for instance necessary if we have to deal with perishable goods in the make-to-stock variant, or with fixed contract deadlines in the make-to-order variant. Besides the flow time objective, this is also the most common timing constraint in basically all areas of scheduling. Only to mention one, speed-scaled scheduling has been investigated in the deadline [16] and in the flow time case [2]. We refer to JRP with deadlines as JRP-D. To implement this special case, for each demand $(i, r, h) \in D$, there is a *deadline* $d \geq r$ such that $h(t) = 0$, for $0 \leq t \leq d - r$, and $h(t) = \infty$, otherwise. We also refer to the periods $\{r, r+1, \ldots, d\}$ as the *due interval* of (i, r, h), and write this demand as (i, r, d). Hence, since this cost structure implies that a deadline is never exceeded, the cost of a schedule S simplifies to $\sum_{i=0}^{N} K_i |S_i|$.

A common generalization of JRP in the make-to-stock variant is the One-Warehouse Multi-Retailer Problem (OWMR) [12]. In this generalization, we are allowed to store the goods satisfying the demands at the warehouse, which contrasts to JRP, where the warehouse only has the role of a cross-docking point. Note that OWMR contains JRP as a special case if we set the delay costs at the warehouse to infinity, which insures that a good is never stored there.

Previous Work. Dozends of heuristics and exact algorithms with superpolynomial running time have been proposed for JRP and its variants during the last decades. Only to mention some older papers, we refer to [17,15,11]. Moreover, the NP-hardness of JRP-L and hence JRP was shown in [3], but little is known about its approximability. Only recently, there has been an increasing interest in finding constant factor approximation algorithms [13,14,12]. But as mentioned in [14], it is not even known whether this problem is APX-hard, i.e., if there is an $\epsilon > 0$ such that finding a $(1+\epsilon)$-approximation in polynomial time is NP-hard. Levi, Roundy, and Shmoys [13] presented a 2-approximation algorithm for JRP, even for arbitrary monotonously increasing delay cost functions, that can hence also be applied to JRP-D. Moreover, Levi and Sviridenko [12] improved the approximation ratio to 1.8, even for the more general OWMR setting, which is also the best known approximation guarantee for JRP-D. It is worth mentioning that this algorithm does not provide an improved approximation ratio in the special case that we only have deadlines. Independently, Becchetti et al. [4] presented a 2-approximation algorithm for the Latency Constrained

Data Aggregation Problem in tree networks, which contains JRP-D as a special case for trees of depth two. Finally, Buchbinder et al. [5] recently presented a 3-competitive online algorithm for JRP.

Both special cases, JRP-L and JRP-D, contain a prominent problem as a special case for $N = 1$, i.e., if there is a single retailer. In this case, JRP-L is a discrete time version of the well-known TCP Acknowledgment Problem [6], and JRP-D is the Interval Stabbing Problem [9], where, given a set of intervals, we have to find a minimum set of points such that each interval contains at least one of them (each due interval corresponds to an interval, and each order corresponds to a point). This problem is equivalent to Clique Cover in interval graphs [9], and can be solved via a simple greedy procedure in linear time. Therefore, we can also think of JRP-D as a hierarchical version of the Interval Stabbing Problem, which itself attracted a considerable amount of research. Even et al. [7] showed that even the capacitated version, where each point has a capacity that indicates how many intervals it may cover, is solvable in polynomial time. However, the Rectangle Stabbing Problem, which is a generalization of the Interval Stabbing Problem to two dimensions, is NP-hard [10], but there is a 2-approximation algorithm [8].

Contributions and Outline. We prove the APX-hardness of JRP-D in Section 7, which is the first APX-hardness proof for a variant of JRP. Moreover, we significantly improve the approximation ratio for JRP-D by presenting a randomized 5/3-approximation algorithm in Section 6. Note that none of the previous algorithms [13,12] gives an improved approximation ratio for the special case of deadlines. Besides that, this also improves the approximation ratio of the Latency Constrained Data Aggregation Problem for the special case of trees of depth two [4]. On the other hand, JRP can be solved in polynomial time by dynamic programming for a fixed number of retailers, or for a fixed number of time periods [11,15,17]. We show in Section 7 that the traditional setting JRP-L with linear delay cost functions is strongly NP-hard, even if each retailer needs to satisfy a constant number of demands. This case is not included in the NP-hardness proof in [3]. Specifically, we show that three demands suffice. It is worth noting that our NP-hardness proof is also much simpler than the NP-hardness proof in [3].

High-Level Algorithmic Ideas. We adapt some ideas from the algorithm for JRP of Levi and Sviridenko [12] with approximation ratio 1.8, which we name SHIFT. This algorithm has the following high-level structure: Combine two algorithms that both first select the warehouse orders S_0, and then, for each retailer i, select a subset of retailer orders $S_i \subseteq S_0$ separately. Each selection step is based on an LP-rounding approach, which exploits the 'line'-structure of the problem implied by the notion of time. Our algorithm has the same high-level structure and is based on a similar LP-rounding approach. We introduce this approach in Section 2, and show how to combine algorithms in the same section. But since we have deadlines, we are able to select retailer orders more greedily than in

algorithm SHIFT as explained in Section 3. This immediately allows us to adapt this algorithm to JRP-D in Section 4. It is worth noting that this adaption is far simpler than the original algorithm, but surprisingly does not yield an improved approximation ratio. Therefore, we introduce a different randomized warehouse order selection method in Section 6, which depends on an input density function of a random variable. We use two techniques to analyze this algorithm: First, we show that an optimal input density function corresponds to an optimal strategy for a game, which we call *generalized tally game*. We introduce this game separately in Section 5, since we think that it is of independent interest and raises some interesting questions. Second, we use an adaption of Wald's equation, whereas we are not aware of any previous use of such an adaption in the analysis of a randomized algorithm.

2 LP-Formulation and Continuous Schedules

We formulate JRP-D as an integer program by introducing an integral variable y_{it} for each retailer i and each period t that indicates whether retailer i orders in period t. Analogously, we have an integral variable x_t for each period t that indicates whether the warehouse orders in this period:

$$\text{minimize} \quad \sum_{t=1}^{T} K_0 x_t + \sum_{i=1}^{N} \sum_{t=1}^{T} K_i y_{it}$$

$$\text{subject to} \quad \sum_{t=r}^{d} y_{it} \geq 1 \qquad \text{for } (i,r,d) \in D \tag{1}$$

$$x_t \geq y_{it} \qquad \text{for } 1 \leq i \leq N, 1 \leq t \leq T \tag{2}$$

$$x_t \in \{0,1\} \quad \text{for } 1 \leq t \leq T$$

$$y_{it} \in \{0,1\} \quad \text{for } 1 \leq i \leq N, 1 \leq t \leq T$$

Constraints (1) enforce that during the due interval of each demand its retailer orders at least once, and constraints (2) ensure that whenever a retailer orders, the warehouse orders as well. Let (LP) be the corresponding linear program where we replace the integrality constraints by the constraints $0 \leq x_t \leq 1$, for $1 \leq t \leq T$, and $0 \leq y_{it} \leq 1$, for $1 \leq i \leq N, 1 \leq t \leq T$. In the following, let (x, y) be an optimum of (LP), which we can compute in polynomial time with the Ellipsoid method. Let $W := \sum_{t=1}^{T} x_t$ and $R_i := \sum_{t=1}^{T} y_{it}$, for $1 \leq i \leq N$.

Using the optimum (x, y) of (LP), we explain in the following how to replace the periods by 'continuous' time, which allows us to simplify the description of the following algorithms. To this end, we need the notion of a *continuous schedule* S', that is, a set of *warehouse orders* $S'_0 \subseteq [0, W)$ with a sequence of subsets of *retailer orders* $S'_1, S'_2, \ldots, S'_N \subseteq S'_0$. Hence, the difference to a schedule is that the warehouse orders are from the set $[0, W)$ instead of the set $\{1, \ldots, T\}$. We also refer to a value $t \in [0, W)$ as a *time*. Finally, define $x_0 := 0$ and $\overline{x}_t := \sum_{s=0}^{t} x_s$, for $0 \leq t \leq T$. We can *convert* a continuous schedule S' into a schedule S as follows:

1. For each period $t = 1, \dots, T$: If $S'_0 \cap [\overline{x}_{t-1}, \overline{x}_t) \neq \emptyset$, then add the order t to S_0.
2. For each retailer $i = 1, \dots, N$: Proceed analogously as for the warehouse in the first step, whereas replace S'_0 and S_0 by S'_i and S_i, respectively.

For each $1 \leq i \leq N$, since $S'_i \subseteq S'_0$, we obtain that $S_i \subseteq S_0$ as well, and hence S is truly a schedule. On the other hand, since we might pool orders during this conversion,

$$|S_i| \leq |S'_i|, \text{ for } 0 \leq i \leq N. \tag{3}$$

For each retailer i, define the multi-interval $r_i := \bigcup_{t=1}^{T} [\overline{x}_{t-1}, \overline{x}_{t-1} + y_{it})$, which is contained in $[0, W)$. Moreover, for a pair of times $a \leq b$, define $m_i(a, b) := \int_a^b \chi_i(x)\, dx$, where χ_i is the indicator function of the multi-interval r_i. We also refer to $m_i(a, b)$ of the *mass* of the interval $[a, b]$ in r_i. Clearly, $m_i(a, b) \leq b - a$. Moreover, we say that the interval $[a, b]$ is *covered* by a set $A \subseteq [0, W)$ if $A \cap [a, b] \neq \emptyset$. Finally, we say that $[a, b]$ has *unit mass* if it has mass one. Observe that $m_i(0, W) = R_i$, for each retailer i. Using these definitions, the following lemma provides a criteria for the feasibility of S with respect to S':

Lemma 1. *A continuous schedule S' can be converted into a feasible schedule S if for each retailer i and each pair of times $a \leq b$ with $m_i(a, b) \geq 1$, we have that $[a, b]$ is covered by S'_i.*

Proof. Consider a fixed retailer i and a demand $(i, r, d) \in D$. Then we have that $\sum_{t=r}^{d} y_{it} \geq 1$, and hence, $m_i(\overline{x}_{r-1}, \overline{x}_d) \geq 1$. Consequently, the interval $[\overline{x}_{r-1}, \overline{x}_d]$ is covered by S'_i, which implies by the construction of S that demand (i, r, d) is satisfied in S, since the corresponding constraint (1) is satisfied. □

Analogously, we say that a continuous schedule S' is *feasible* if the property from Lemma 1 is satisfied. We obtain the following lemma, which allows us to combine algorithms.

Lemma 2. *If there are two polynomial time (randomized) algorithms $\mathcal{A}^1$ and $\mathcal{A}^2$, where algorithm $\mathcal{A}^1$ returns a feasible continuous schedule S' with*

$$\mathbb{E}\left[\frac{|S'_0|}{W}\right] \leq \alpha_1 \text{ and } \mathbb{E}\left[\frac{|S'_i|}{R_i}\right] \leq \beta_1, \text{ for } 1 \leq i \leq N,$$

algorithm $\mathcal{A}^2$ returns a feasible continuous schedule S' with

$$\mathbb{E}\left[\frac{|S'_0|}{W}\right] \leq \alpha_2 \text{ and } \mathbb{E}\left[\frac{|S'_i|}{R_i}\right] \leq \beta_2, \text{ for } 1 \leq i \leq N,$$

$\alpha_1 \leq \alpha_2$, $\beta_1 \geq \alpha_1$, $\beta_2 \leq \alpha_2$, and there is an x with $\alpha_1 + \beta_1 x = \alpha_2 + \beta_2 x$, then there is a randomized $(\alpha_1 + \beta_1 x)/(1 + x)$-approximation algorithm for JRP-D.

Proof. Let $C_W := K_0 W$ and $C_R := \sum_{i=1}^{N} K_i R_i$ be the warehouse ordering costs and retailer ordering costs of the optimum (x, y) of (LP), respectively (recall that $W = \sum_{t=1}^{T} x_t$ and $R_i = \sum_{t=1}^{T} y_{it}$, for $1 \leq i \leq N$). Moreover, define $\lambda := C_R / C_W$. We compute a feasible continuous schedule S' with algorithm $\mathcal{A}^1$ or algorithm $\mathcal{A}^2$, whereas we choose $\mathcal{A}^1$ if $\lambda \leq x$ and $\mathcal{A}^2$ if $\lambda > x$, and then we use Lemma 1 to convert S' into a feasible schedule S. Because of inequalities (3), the cost $\sum_{i=0}^{N} K_i |S_i|$ of S is bounded from above by $\sum_{i=0}^{N} K_i |S'_i|$. Moreover, the cost of an optimal schedule is bounded from below by $C_W + C_R$. Therefore, the choice of the algorithm above, linearity of expectation, and simple arithmetic shows that the cost of S is at most $(\alpha_1 + \beta_1 x)/(1 + x)$ times the cost of an optimal schedule, which proves the claim. □

Lemma 2 allows us to restrict our attention to the computation of continuous schedules. Hence, in the remainder of this paper, a schedule is always a continuous schedule.

3 Greedy Selection of Retailer Orders

As explained in Section 2, recall that we only consider continuous schedules in the remainder of this paper, and we hence refer to a continuous schedule simply as a schedule. Now assume that we have already selected the warehouse orders S_0 of a schedule S such that each unit length interval $[a, b] \subseteq [0, W)$ is covered by S_0, i.e., $S_0 \cap [a, b] \neq \emptyset$. Then, in order to complete S, we only have to select the retailer orders with respect to S_0. We can do this as follows:

1. For each retailer $i = 1, \ldots, N$: Iteratively increase a time t, where initially $t \leftarrow 0$. In each iteration, set $t' > t$ to the time with $m_i(t, t') = 1$ if such a time exists. If yes, then set $I \leftarrow (t, t']$ and $t \leftarrow \max(S_0 \cap I)$, and add t to S_i. Otherwise, terminate the retailer order selection of retailer i and proceed with the next retailer.

Note that in each iteration, the interval I is the next interval we need to cover with some retailer order in order to ensure the feasibility of S, since no previously added retailer order covers this interval yet. Since we are only allowed to use warehouse orders, we *greedily* choose the warehouse order t that is as far away from the retailer order added in the last iteration as possible, but still covers I. Therefore, we also refer to I as the *cover interval* of the new retailer order t. The following lemma proves the correctness of this retailer order selection:

Lemma 3. *The computed schedule S is feasible.*

Proof. Consider a fixed retailer i. We assume in each iteration that there is a warehouse order in $S_0 \cap I$. But this holds since $t' - t \geq m_i(t, t') \geq 1$, and hence $S_0 \cap I \neq \emptyset$. We conclude that each interval with unit mass in r_i is covered by some retailer order in S_i, which yields the feasibility of the computed schedule S. □

By Lemma 3, we only have to explain how to select the warehouse orders S_0. We do this twice in the following, in Section 4 and Section 6. Combining both approaches as explained in Lemma 2 results in the randomized 5/3-approximation algorithm.

4 Selecting the Warehouse Orders as a Grid

In this section, we present a simple adaption of the algorithm of Levi and Sviridenko [12]. This adaption takes an additional parameter $0 \leq c \leq 1$ as input and works as follows:

1. Select the warehouse orders as $S_0 \leftarrow \{ci \mid i \in \mathbb{N} : ci < W\}$.
2. Use the greedy retailer order selection described in Section 3 to turn the warehouse orders S_0 into a feasible schedule S.

We can think of the computed warehouse orders S_0 as a *grid* with grid-length c. The following theorem analyzes the computed schedule S:

Theorem 1.

$$\frac{|S_0|}{W} \leq \frac{1}{c} \text{ and, for } 1 \leq i \leq N, \frac{|S_i|}{R_i} \leq \begin{cases} 2 & c > \frac{1}{2}, \\ \frac{1}{1-c} & c \leq \frac{1}{2}. \end{cases}$$

Proof. By the selection of the warehouse orders, we immediately conclude that $|S_0|/W \leq 1/c$. Therefore, we only have to consider the retailer orders. To this end, consider a fixed retailer i. Let then $t_1 < t_2 < \ldots < t_k$ be the retailer orders in S_i, and let $t_0 := 0$ and $t_{k+1} := W$. As in the claim of the theorem, we have to distinguish two cases:

Case $c > 1/2$: For each $1 \leq j \leq k$, we conclude with the definition of the greedy retailer order selection that $m_i(t_{j-1}, t_{j+1}) \geq 1$. Now observe that $m_i(t_{j-1}, t_{j+1}) = m_i(t_{j-1}, t_j) + m_i(t_j, t_{j+1})$. Using this, we find that

$$|S_i| \leq \sum_{j=1}^{k} m_i(t_{j-1}, t_j) + m_i(t_j, t_{j+1}) \leq 2 \sum_{j=1}^{k+1} m_i(t_{j-1}, t_j) = 2R_i,$$

which completes this case.

Case $c \leq 1/2$: Consider a fixed $1 \leq j \leq k$, and let $(t, t']$ be the cover interval of t_j as defined in Section 3. Observe that $t = t_{j-1}$, and recall that $m_i(t, t') = 1$. Now let $t \leq s \leq t'$ be the time such that $m_i(s, t') = c$. Consequently, $t' - s \geq m_i(s, t') = c$, and therefore, by the definition of the warehouse order selection, $S_0 \cap [s, t'] \neq \emptyset$. Hence, we find with the definition of the greedy retailer order selection that $t_j \in [s, t']$. Thus, we have $m_i(t_{j-1}, t_j) \geq m_i(t, t') - m_i(s, t') = 1 - c$. By applying these arguments to all retailer orders $t_1, t_2, \ldots, t_k$, we obtain that $(1 - c)k \leq R_i$, which completes this case and the proof of the theorem, since $k = |S_i|$. □

Theorem 1 yields that in order to minimize the warehouse ordering cost, we have to choose $c = 1$, and in order to minimize the retailer ordering cost, we have to choose a small c. Note that the former case immediately results in a 2-approximation algorithm with Lemma 2. In the algorithm of Levi and Sviridenko [12], a more involved warehouse and retailer selection method was used to deal with arbitrary delay cost functions. Specifically, they additionally *shifted* a similar grid randomly, but they used the same principle as in the proof of Theorem 1 to bound the ordering costs. Consequently, they derived a similar theorem as Theorem 1 for these costs. Finally, they used the arguments from Lemma 2 to combine the two algorithms for $c = 1$ and $c = 1/3$ to a 1.8 approximation algorithm. One might think that if we do not have to deal with arbitrary delay cost functions, it should be possible to improve the approximation ratio using the same principle. But surprisingly, this approach does not yield an improved approximation ratio for the special case that we only have deadlines (at least the used analysis does not yield an improvement). Hence, we need a significantly different method to select warehouse orders.

5 The Generalized Tally Game

In this section, we introduce two black-jack type games, the *tally game* and the *generalized tally game*. We need these games for the analysis of the algorithm described in Section 6. However, since we think that these games are of independent interest, we introduce them separately. In the *tally game*, we are initially given some value $x \geq 0$. In each round of the game, we draw some random value d according to the density function f of a random variable X with $0 \leq X \leq 1$ and $\mathbb{E}[X] = 1/2$. If $x - d \geq 0$, then we set $x \leftarrow x - d$, and proceed with the next round. Otherwise, the game finishes and we have to pay the remaining value of x (this is a pessimistic game). For a density function f and an initial value x, let $Z(f, x)$ be the random variable that describes the outcome of the game, i.e., the final value of x. Note that for any density function f and any $0 \leq x \leq 1$, $\mathbb{E}[Z(f, x)] \leq x$. The *generalized tally game* is then the extension where we are allowed to choose the density function f, but some adversary chooses the initial value of x. Since $X \leq 1$ we can wlog assume that $x \leq 1$. The question is then how to choose f such that the expected final pay is minimized with respect to such an adversarial choice? Hence, we need to minimize

$$\max_{0 \leq x \leq 1} \mathbb{E}[Z(f, x)]. \tag{4}$$

Observe that the function $x \mapsto \mathbb{E}[Z(f, x)]$ which maps $[0, 1]$ to itself satisfies the integral equation

$$\mathbb{E}[Z(f, x)] = \int_0^x f(x - y)\mathbb{E}[Z(f, y)]\ dy + x \int_x^1 f(y)\ dy. \tag{5}$$

Moreover, we have the two boundary conditions

$$\mathbb{E}[Z(f, 0)] = 0 \text{ and } \frac{d\mathbb{E}[Z(f, 0)]}{dx} = 1. \tag{6}$$

To see the second boundary condition, note that for any density function f, $\Pr[Z(f,x)=x] \to 1$ as $x \to 0$. Using this, we can formally interpret the generalized tally game as follows: Find a density function f such that the function which satisfies equation (5) and boundary conditions (6) with respect to f minimizes objective (4). However, consider the simple density function

$$f(x) := \begin{cases} 4x & 0 \le x \le 1/2, \\ 2-4x & 1/2 < x \le 1, \\ 0 & \text{otherwise}, \end{cases}$$

which is basically a triangle. The following lemma analyzes f (proof in the full version):

Lemma 4.

$$\max_{0 \le x \le 1} \mathbb{E}[Z(f,x)] < \frac{1}{3}$$

The full version of this paper contains some experiments where we compare f with some other natural density functions such as the density function of the uniform distribution. These experiments show that although the function f is relatively simply, and hence easy to formally analyze, it surprisingly meets or beats the performance of all other considered functions. However, finding the 'true' optimal function f remains an interesting open problem.

6 Random Selection of Warehouse Orders and a 5/3-Approximation Algorithm

In this section, we describe a randomized algorithm, which is our main building block in the 5/3-approximation algorithm. This randomized algorithm takes the density function f of a random variable X with $0 \le X \le 1$ and $\mathbb{E}[X] = 1/2$ as an additional input and works as follows:

1. Select the warehouse orders by iteratively increasing a time t, where initially $t \leftarrow 0$. In each iteration, draw a random d according to the density function f. If $t+d < W$, then set $t \leftarrow t+d$, and add t to S_0. Otherwise, stop the selection of warehouse order.
2. Apply the greedy retailer order selection method described in Section 3 to turn the warehouse orders S_0 into a feasible schedule S.

Our main tool for the analysis of this algorithm is the following adaption of Wald's equation (proof in full version):

Theorem 2. *Let $X_1, X_2, \dots$ be an infinite sequence of random variables such that for each $i \ge 1$, $\mathbb{E}[X_i \mid X_1, X_2, \dots, X_{i-1}] \ge \mu$ and $0 \le X_i \le c$ for two constants $\mu, c > 0$. Moreover, for some $C \ge 0$, let*

$$L := \max\left\{k \;\middle|\; \sum_{i=1}^{k} X_i \leq C\right\}.$$

Then $\mathbb{E}[L] \leq C/\mu$.

An interesting fact about Theorem 2 is that we do not require that the variables $X_1, X_2, \ldots$ are independent, but we only need that each expected value $\mathbb{E}[X_i \mid X_1, X_2, \ldots, X_{i-1}]$ is bounded from below. On the other hand, if this property does not hold, then we do not obtain such a theorem. To see this, let X be uniformely distributed in $[0, 1]$, and let $X_i = X$, for each $i \geq 1$, i.e., all random variables $X_1, X_2, \ldots$ have the same outcome. It can then be easily verified that surprisingly $\mathbb{E}[L] = \infty$. Now we are ready to analyze the computed schedule S:

Lemma 5.

$$\frac{\mathbb{E}[|S_0|]}{W} \leq 2$$

Proof. Let $t_1 < t_2 < \ldots < t_L$ be an ordering of the warehouse orders S_0, and let $t_0 := 0$. Hence, L is a random variable with $L = |S_0|$. Moreover, for each $1 \leq j \leq L$, define $X_j := t_j - t_{j-1}$. We can think of $X_1, X_2, \ldots, X_L$ as a sequence of random variables which describe the distances between the warehouse orders. By the definition of the warehouse order selection, for each $1 \leq j \leq L$, we have $0 \leq X_j \leq 1$ and $\mathbb{E}[X_j] = \mathbb{E}[X] = 1/2$, where X is the random variable with the input density function f. By adding an infinite sequence of variables $X_{L+1}, X_{L+2} \ldots$, where $X_j := 1$, for $j \geq L + 1$, we can extend the sequence $X_1, X_2, \ldots, X_L$ to an infinite sequence $X_1, X_2, \ldots$ of random variables with $0 \leq X_j \leq 1$ and $\mathbb{E}[X_j] \geq 1/2$, for $j \geq 1$. Moreover, we conclude with the definition of the warehouse order selection that $L = \max\{k \mid \sum_{j=1}^{k} X_j < W\}$. Consequently, Theorem 2 implies that $\mathbb{E}[L] \leq 2W$, which proves the claim of the lemma. □

The following lemma establishes a relation to the generalized tally game introduced in Section 5:

Lemma 6. *For each retailer* $1 \leq i \leq N$,

$$\frac{\mathbb{E}[|S_i|]}{R_i} \leq \frac{1}{1 - \max_{0 \leq x \leq 1} \mathbb{E}[Z(f, x)]}.$$

Proof. Consider a fixed retailer i. Let $t_1 < t_2 < \ldots < t_L$ be an ordering of the retailer orders S_i, and let $t_0 := 0$. For each $1 \leq j \leq L$, define $X_j := m_i(t_{j-1}, t_j)$. Thus, we can now think of $X_1, X_2, \ldots, X_L$ as a sequence of random variables which describe the masses between the retailer orders. Note that $0 \leq X_j \leq 1$, for $1 \leq j \leq L$. We will moreover show that for each $1 \leq j \leq L$,

$$\mathbb{E}[X_j \mid X_1, X_2, \ldots, X_{j-1}] \geq 1 - \max_{0 \leq x \leq 1} \mathbb{E}[Z(f, x)]. \tag{7}$$

In this case, we can proceed similar to the proof of Lemma 5. Specifically, by adding an infinite sequence of variables $X_{L+1}, X_{L+2}, \ldots$, where $X_j := 1$, for

$i \geq L+1$, we can extend the sequence $X_1, X_2, \ldots, X_L$ to an infinite sequence $X_1, X_2, \ldots$ of random variables such that inequality (7) even holds for each variable in this infinite sequence. By the definition of the greedy retailer order selection, we find again that $L = \max\{k \mid \sum_{i=1}^{k} X_i < R_i\}$. Combining this, Theorem 2 implies that

$$\mathbb{E}\left[|S_i|\right] = \mathbb{E}\left[L\right] \leq \frac{R_i}{1 - \max_{0 \leq x \leq 1} \mathbb{E}\left[Z(f,x)\right]},$$

which proves the claim of the lemma.

To prove inequality (7), consider a fixed $1 \leq j \leq L$, and let $(t_{j-1}, t']$ be the cover interval of the retailer order t_j as defined in Section 3 with $t' - t_{j-1} \geq m_i(t_{j-1}, t') = 1$, and hence $t' - 1 \geq t_{j-1}$. On the other hand, since the warehouse orders are selected such that they have at most distance one, there is at least one warehouse order in the interval $[t'-1, t']$. Let then $t'_0 < t'_1 < \ldots < t'_s$ be all warehouse orders in $[t'-1, t']$, and let $x := t' - t'_0$. By the definition of the greedy retailer order selection, we find that $t'_s = t_j$. Moreover, by the definition of the warehouse order selection, we have that the random variables $t'_l - t'_{l-1}$, $1 \leq l \leq s$, are drawn according to the density function f. Therefore, the random variable $t' - t_j$ is identically distributed as the random variable $Z(f,x)$. Consequently, $\mathbb{E}\left[t' - t_j\right] = \mathbb{E}\left[Z(f,x)\right]$, and hence, $\mathbb{E}\left[m_i(t_j, t')\right] \leq \mathbb{E}\left[Z(f,x)\right]$. But then, since $m_i(t_{j-1}, t') = 1$, we have $\mathbb{E}\left[X_j\right] \geq 1 - \mathbb{E}\left[Z(f,x)\right]$, which implies inequality (7). Note that x depends on the outcome of the variables $X_1, X_2, \ldots, X_{j-1}$. □

Theorem 3. *There is a randomized* $5/3$*-approximation algorithm for JRP-D.*

Proof. By combining Lemma 4, Lemma 5, and Lemma 6, we find that we can choose the input density function f such that the randomized algorithm described in this section returns a schedule S with $\mathbb{E}\left[|S_0|\right]/W \leq 2$ and $\mathbb{E}\left[|S_i|\right]/R_i \leq 3/2$, for $1 \leq i \leq N$. Now recall the algorithm described in Section 4, which by Theorem 1 returns for $c = 1$ a schedule S with $|S_0|/W \leq 1$ and $|S_i|/R_i \leq 2$, for $1 \leq i \leq N$. The claim follows then from Lemma 2. □

7 Hardness Results

The following two theorems are proven in the full version of this paper:

Theorem 4. *JRP with deadlines (JRP-D) is APX-hard, even if each retailer has to satisfy only three demands.*

Theorem 5. *JRP with linear delay cost functions (JRP-L) is strongly NP-hard, even if each retailer has to satisfy only three demands.*

References

1. Aksoy, Y., Erenguc, S.S.: Multi-Item Inventory Models with Co-ordinated Replenishments: A Survey. International Journal of Operations Production Management 8(1), 63–73 (1988)
2. Albers, S., Fujiwara, H.: Energy-Efficient Algorithms for Flow Time Minimization. ACM Trans. Algorithms 3(4), 49 (2007)

3. Arkin, E., Joneja, D., Roundy, R.: Computational Complexity of Uncapacitated Multi-Echelon Production Planning Problems. Operations Research Letters 8(2), 61–66 (1989)
4. Becchetti, L., Korteweg, P., Marchetti-Spaccamela, A., Skutella, M., Stougie, L., Vitaletti, A.: Latency Constrained Aggregation in Sensor Networks. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 88–99. Springer, Heidelberg (2006)
5. Buchbinder, N., Kimbrel, T., Levi, R., Makarychev, K., Sviridenko, M.: Online Make-to-Order Joint Replenishment Model: Primal Dual Competitive Algorithms. In: Proc. of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 952–961 (2008)
6. Dooly, D.R., Goldman, S.A., Scott, S.D.: TCP Dynamic Acknowledgment Delay: Theory and Practice. In: Proc. of the 30th Annual ACM Symposium on the Theory of Computing (STOC), pp. 389–398 (1998)
7. Even, G., Levi, R., Rawitz, D., Schieber, B., Shahar, S., Sviridenko, M.: Algorithms for Capacitated Rectangle Stabbing and Lot Sizing with Joint Set-Up Costs. ACM Trans. Algorithms 4(3), Article No. 34 (2008)
8. Gaur, D.R., Ibaraki, T., Krishnamurti, R.: Constant Ratio Approximation Algorithms for the Rectangle Stabbing Problem and the Rectilinear Partitioning Problem. J. Algorithms 43(1), 138–152 (2002)
9. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Mathematics, vol. 57. North-Holland Publishing Co., The Netherlands (2004)
10. Hassin, R., Megiddo, N.: Approximation Algorithms for Hitting Objects with Straight Lines. Discrete Applied Mathematics 30(1), 29–42 (1991)
11. Kao, E.: A Multi Product Dynamic Lot Size Model with Individual and Joint Setup Costs. Operations Research 27, 279–289 (1979)
12. Levi, R., Sviridenko, M.: Improved Approximation Algorithm for the One-Warehouse Multi-Retailer Problem. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 188–199. Springer, Heidelberg (2006)
13. Levi, R., Roundy, R., Shmoys, D.B.: Primal-Dual Algorithms for Deterministic Inventory Problems. In: Proc. of the 36th ACM Symposium on Theory of Computing (STOC), pp. 353–362 (2004)
14. Levi, R., Roundy, R., Shmoys, D.B.: A Constant Approximation Algorithm for the One-Warehouse Multi-Retailer Problem. In: Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 365–374 (2005)
15. Veinott, A.: Minimum Concave Cost Solutions of Leontief Substitution Models of Multi-Facility Inventory Systems. Operations Research 17, 262–291 (1969)
16. Yao, F.F., Demers, A.J., Shenker, S.: A Scheduling Model for Reduced CPU Energy. In: Proc. of the 36th Annual Symposium on Foundations of Computer Science (FOCS), pp. 374–382 (1995)
17. Zangwill, W.I.: A Deterministic Multi-product Multi-Facility Production and Inventory Model. Operations Research 14, 486–507 (1966)

A PTAS for Node-Weighted Steiner Tree in Unit Disk Graphs

Xianyue Li[1], Xiao-Hua Xu[3], Feng Zou[2], Hongwei Du[3], Pengjun Wan[3,⋆], Yuexuan Wang[4], and Weili Wu[2]

[1] School of Mathematics and Statistics, Lanzhou University
Lanzhou, Gansu 730000, China
lixianyue@lzu.edu.cn
[2] Department of Computer Science, University of Texas at Dallas
Richardson, TX 75080, USA
phenix.zou@student.utdallas.edu, weiliwu@utdallas.edu
[3] Department of Computer Science, Illinois Institute of Technology
Chicago, IL 60616, USA
xxu23@iit.edu, hongwei@cs.cityu.edu.hk, wan@cs.iit.edu
[4] Institute of Theoretical Computer Science, Tsinghua University
Beijing 100084, China
wangyuexuan@tsinghua.edu.cn

Abstract. The node-weighted Steiner tree problem is a variation of classical Steiner minimum tree problem. Given a graph $G = (V, E)$ with node weight function $C : V \to R^+$ and a subset X of V, the node-weighted Steiner tree problem is to find a Steiner tree for the set X such that its total weight is minimum. In this paper, we study this problem in unit disk graphs and present a $(1+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$, when the given set of vertices is c-local. As an application, we use node-weighted Steiner tree to solve the node-weighted connected dominating set problem in unit disk graphs and obtain a $(5+\varepsilon)$-approximation algorithm.

Keywords: Node-weighted Steiner tree, minimum weighted connected dominating set, polynomial-time approximation scheme, approximation algorithm.

1 Introduction

Given a graph $G = (V, E)$ and a subset $X \subseteq V$, the classical Steiner tree problem is to find a tree of shortest length in G interconnecting X, where the length is the sum of the lengths of all edges in the tree. This problem is known to be NP-hard in graphs, and it is also proved to be NP-hard in most other metrics rather than Euclidean [7]. Lots of effort have been devoted to study the approximation algorithms for this problem [4,10,14,17,20] and some of them have successfully achieved constant ratios [10,17,20]. The best known result among all of them is $\rho = 1 + \frac{\ln 3}{2} \approx 1.55$ by Robins et.al [17] up till now.

⋆ This work was partially supported by NSF under grant CNS-0831831.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 36–48, 2009.

The *Node-weighted Steiner Tree* problem (NWST) is a variation of the classical Steiner tree problem. Given a graph $G = (V, E)$ with node weight function $C : V \to R^+$ and a subset X of V, which is denoted as the *terminal set*, the node-weighted Steiner tree problem is to find a Steiner tree for the set X such that the total weight of this Steiner tree is minimum. In 1991, Berman [3] proved that NWST problem can not be approximated within a factor of $o(\log k)$. Later, Klein and Ravi [13] presented the first asymptotically optimal solution of approximation ratio $2 \ln k$, by constructing the Steiner tree using "spiders", which is a special kind of tree with at most one node of degree greater than two. Later, this ratio is improved to be $1.35 \ln k$ by Guha and Khuller [8]. They introduce a new concept called "branch-spider" based on "spider".

Recently, researchers are interested in this problem on a special type of graphs called unit disk graphs, which has a wide application in networks. A *unit disk graph* is associated with a set of unit disks in the Euclidean plane. Each vertex in the graph is the center of a unit disk and an edge exists between two vertices u and v if and only if the Euclidean distance between u and v is at most 1. Zou et al. [21] is the first to give a 3.875-approximation algorithm by converting the node-weighted Steiner tree problem to the classical Steiner tree problem.

In this paper, we concern about the same problem from a different perspective. We present a polynomial-time approximation scheme (PTAS) when the given set of vertices is *c-local*. A *polynomial-time approximation scheme* (PTAS) is a family of approximation algorithm with ration $1+\varepsilon$ for any $\varepsilon > 0$ and such a scheme would be the best for a NP-hard problem we can expect.

As an application, we use our algorithm to solve minimum weighted connected dominating set problem in unit disk graph. The Minimum Weighted Connected Dominating Set problem (MWCDS) is a generalization of the minimum connected dominating set problem. Given a graph $G = (V, E)$ with node weight function $C : V \to R^+$, the *minimum weighted connected dominating set* problem is to find a connected dominating set of G such that its total weight is minimized. Up till now, the best known approximation ratio for MWCDS in general graphs is $O(\log n)$ [8].

In unit disk graphs, researchers usually construct an approximation algorithm for MWCDS with the following two steps. The first step is to find a DS, and the second step is to interconnect DS. Ambühl et al. [1] gave the first constant-factor algorithm for MWCDS with an approximation ratio of 89 with a 72-approximation algorithm for MWDS. Huang et al. [11] improved the approximation ratio from 89 to $10+\varepsilon$ with a $(6+\varepsilon)$-approximation algorithm for the first step. Recently, Dai and Yu [6] gave a $(5+\varepsilon)$-approximation algorithm for MWDS. Therefore, the best known approximation ratio for MWCDS in UDG is $8.875+\varepsilon$. In this paper, we first give a $(4+\varepsilon)$-approximation algorithm for MWDS and then obtain a $(5+\varepsilon)$-approximation algorithm by using Steiner tree to interconnect such DS.

The rest of this paper is organized as follows. In Section 2, we give some useful notations and lemmas. In Section 3, we introduce our main strategy, which is the partition and shifting strategy. Based on the partition, we present our

algorithm in Section 4. In Section 5, we first show that our algorithm is a PTAS for NWST in unit disk graphs when the given set of vertices is *c-local*. Then, as an application, we obtain a $(5+\varepsilon)$-approximation algorithm for MWCDS in unit disk graphs.

2 Preliminaries and Fundamental Lemmas

Given a node-weighted unit disk graph $G = (V, E)$ with weighted function C, and the terminal set X. For convenience, we normalize the weight function C such that for any vertex v in G, $C(v) \geq 1$. And we denote a vertex u in the Steiner tree T for X as *terminal vertex* if $u \in X$; otherwise, we call it *Steiner vertex*.

We introduce two kinds of distance between any two vertices u and v in the graph, which are called e-distance and w-distance, respectively. In detail, $dist_e(u, v)$ is calculated as the Euclidean distance between the two nodes and $dist_w(u, v)$ is calculated as the minimum weight of all the possible paths connecting u and v in G. The weight of each path here is calculated as the total weight of all intermediate vertices on that path.

Since the graph is node-weighted instead of edge-weighted, the construction of the minimum spanning tree, or saying the minimum node-weighted spanning tree on terminal set X (denoted as $T_s(X)$), is a little bit different here. Firstly, we create an edge-weighted complete graph G' on terminal set X such that for any edge (u, v) in G' $(u, v \in X)$, its weight is equal to the w-distance between u and v. Then let T_s be a minimum spanning tree of G'. Easy to see, for any edge (u, v) in T_s, it corresponds to the minimum weighted-path between u and v in G. In the following, we use $C(T_s)$ to denote the total weight of edges in T_s. Meanwhile, for simplicity, we keep T_s as it is without replacing the weighted-edge with the corresponding minimum weighted-path between any two nodes in the node-weighted graph.

A set of vertices X is called *c-local* in a node-weighted graph if in the minimum node-weighted spanning tree for X, the weight of longest edge is at most c. This definition could be considered as the node-weighted version of the *c-local* definition given by [18]. In the following of paper, we assume that the terminal set X is c-local for some constant c.

In [16], Robin et.al showed that the Minimum Spanning Tree Number for Euclidean metric is 5 and [19] shows that for any unit disk graph G, there exits a spanning tree T of G such that the maximum degree of G is at most 5. Thus, we can get the following lemma easily. (Proof is omitted due to lack of space.)

Lemma 1. *The minimum spanning tree of a given terminal set in a node-weighted graph has an approximation ratio of at most 5 for the optimal Steiner tree of the same given terminal set.*

Following are some of the preliminaries for MWCDS problem, which we will talk about as an application after introducing our own algorithm. Given a graph $G = (V, E)$, a *Dominating Set* (DS) is a subset $D \subseteq V$ such that for every vertex $v \in V$, either $v \in D$, or v has a neighbor in D. If the graph induced from D

is connected, then D is called a *Connected Dominating Set* (CDS). *Minimum Connected Dominating Set* (MCDS) problem is to find a connected dominating set in G with minimum size, which is a well-known NP-complete problem[7] and has been further shown to be NP-complete even though the given graph is a unit disk graph (UDG)[15]. The MWCDS is a generalization of MCDS, which is obviously NP-hard problem in unit disk graphs.

3 Partition and Shifting

One of the key strategies adopted in our algorithm is partition and shifting. Considered as a special way to make restriction and derandomize the probabilistic result to get a deterministic one, researchers has started to use partition and shifting strategy [2,9] in approximation algorithms from early 1980s.

Specifically, in our algorithm, we partition the graph according to the following strategy. Let A be the smallest square containing all vertices of G with size $q \times q$. For a given integer k, let $l = (\lfloor q/k \rfloor + 2)k$ and make the lower left corner of the

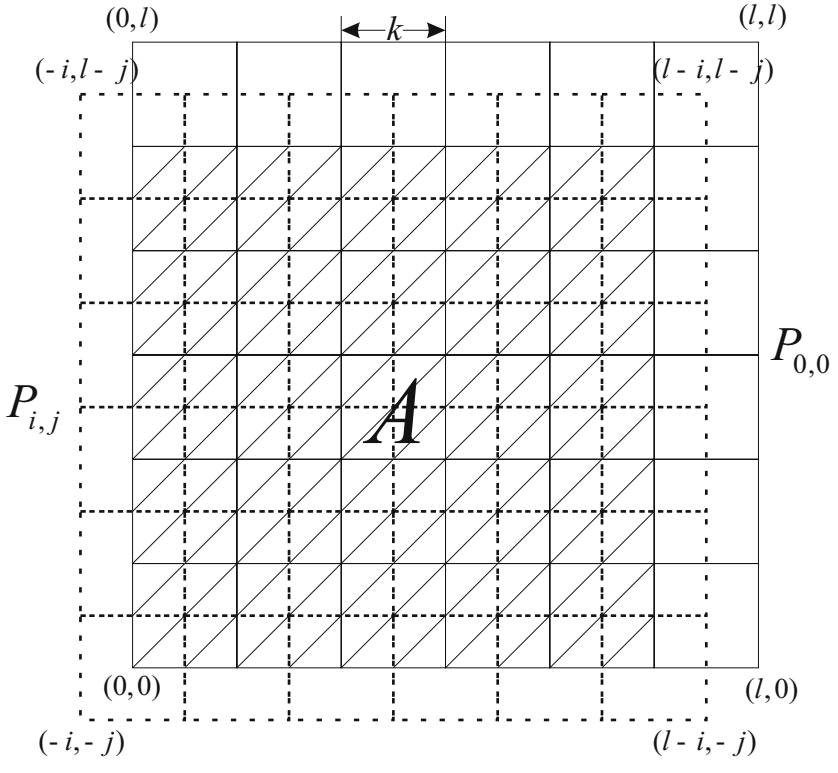


Fig. 1. Two partitions $P_{0,0}$ and $P_{i,j}$ with shadow area is A. The black is $P_{0,0}$ and the dashed is $P_{i,j}$.

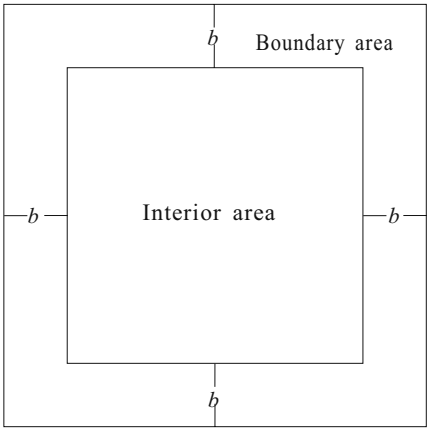


Fig. 2. The interior area and boundary area with boundary width $b = (1 + 1.5 \log k)c$

square A as the center of the coordinate system. We extend the area A to A' of size $l \times l$ and divide it into small cells such that the size of each cell is $k \times k$ (see Fig 1.) Furthermore, for each cell, we divide it into two parts: interior area and boundary area with boundary width $b = (1 + 1.5 \log k)c$ (as in Fig 2.). We call this partition as $P_{0,0}$. Then we shift the extended area A' to make its lower left corner positioned at point $(-i, -j)$ $(0 \leq i, j \leq k - 1)$ in the coordinate system, to get another partition $P_{i,j}$. Clearly, there are all k^2 possible partitions and any partition contains the area A.

Our intention of making use of partition and shifting strategy is that for any fixed partition, we first construct the local optimal solution for each cell. Then, we further modify the union of the local optimal solution of all cells to make it a feasible solution. In order to achieve the best solution, we use shifting to obtain a set of solutions on different partitions and choose the best solution among all these feasible solutions. With this strategy, we could better bound the approximation ratio of our algorithm.

4 NWST Approximation Algorithm

In this section, we present our approximation algorithm for this problem based on the partition and shifting strategy introduced in last section. Before introducing this algorithm, we first give some useful notations.

Recall that T_s is a minimum spanning tree of terminal set X in G. For a fixed partition P, we call an edge uv a crossing edge if at least one of the end nodes u or v is contained in boundary area of P. We use X_p to denote the set of terminal vertices contained in the interior area of P. Note that we study this problem under a fixed partition P in this section.

The algorithm has two steps as follows. Firstly, for each cell, we construct a local optimal Steiner forest on terminal vertices in the interior area of this cell. Then, union all these forests to obtain a local optimal Steiner forest $\hat{F}_p$ on X_p.

In the second step, we add all the crossing edges in T_S into $\hat{F}_p$ to get a Steiner tree on terminal set X. We call the resulted graph G_p for a specific partition P. In order to approximate the minimum node-weighted Steiner tree, we calculate all $G_{p_{i,j}}$ for $0 \leq i, j \leq k-1$ and choose the minimum one among all of them as the output of our algorithm.

Following, we describe in detail the construction of the local optimal Steiner forest and the final Steiner tree in our scheme.

4.1 Local Optimal Steiner Forest $\hat{F}_p$

Our target in this part is to construct a local optimal Steiner forest $\hat{F}_p$ on X_p. In order to achieve this goal, we first group the terminal vertices in the interior area of every cell satisfying that the w-distance between any two groups is greater than c. The grouping is achieved by first constructing the minimum spanning tree on the terminal vertices in the interior area of each cell and then deleting all edges with weight greater than c. Obviously, by doing so, terminal vertices will be divided into different connected components. We consider all terminals vertices in the same connected component to be in the same group. Clearly, the w-distance between any two groups is greater than c. Otherwise, there will be another spanning tree with weight less than our minimum spanning tree, which derives a contradiction.

For a fixed cell, let $Y_1, \ldots, Y_m$ be the different groups of all terminal vertices after grouping. In order to get desired solution, we merge $Y_1, \ldots, Y_m$ into new groups, construct Steiner minimum tree for each new group in this cell and then combine them to form a Steiner forest. If we calculate the total weight of vertices in the resulted Steiner forest to be the merging-cost of this specific merging, with different possible merging choices, we choose the merging with the minimum merging-cost among all of them. The corresponding Steiner forest is the local optimal Steiner forest that we are after for this cell in partition P.

For a fixed partition P, we denote $\hat{F}_p$ the local optimal Steiner forest on the terminal vertices X_p in graph G. It is calculated as the union of local optimal Steiner forest in each cell. From the method for $\hat{F}_p$ construction described above, we can obtain the following lemma easily.

Lemma 2. *$\hat{F}_p$ is a Steiner forest on X_P with the following properties:*
(1) Each tree in the forest $\hat{F}_p$ is completely included in some cell.
(2) The w-distance between any two terminal vertices in different trees of $\hat{F}_p$ is greater than c.

In the following, we will discuss the running time for computing a local optimal Steiner forest. Let n be the number of vertices of G. Since G is a unit disk graph, G can be cover by a square with size $n \times n$. Recall that the size of every cell is $k \times k$, there are at most $O(n/k)^2$ cells. Then, we will discuss the time for computing a local optimal Steiner forest in a cell. Let Y be the set of terminal vertices in the interior area in this cell and m the number of groups in the same cell. Since there is a minimum Steiner tree containing all of edges of induced subgraph

$G[Y]$, we shrink every component of $G[Y]$ to a new vertex and set Y' as the set of these new vertices. Easy to see that $m < |Y'|$. If we find a minimum Steiner tree on Y', and replace every vertex in Y' by the corresponding component, we obtain a minimum Steiner tree on Y. Hence, the time-complexity to compute local optimal Steiner forest is $O(2^m M(|Y'|))$ [18], where $M(|Y'|)$ is the time to compute an optimal Steiner tree on terminal set Y' and $M(|Y'|)$ is exponential in $|Y'|$ [5,12]. In order to show $|Y'|$ is bounded by a constant value, we divide the cell into some squares such that the size of each square is $\frac{\sqrt{2}}{2} \times \frac{\sqrt{2}}{2}$. Then the terminal vertices in each square must belong to the same component of $G[Y]$. Hence, there are at most $2k^2$ components in $G[Y]$, i.e., $|Y'| \leq 2k^2$. In Section 5, we will show that k is only related with c and ε. Hence, we can compute a local optimal Steiner forest in polynomial times.

4.2 Constructing NWST T_{out} from $\hat{F}_p$

Recall that in the above subsection, we get a local optimal Steiner forest $\hat{F}_p$ on X_p and the w-distance between any two terminal vertices in different trees of $\hat{F}_p$ is greater than c.

Let E_p^s be the set of all crossing edges in T_s under a partition P. In order to interconnect the disconnected components in the Steiner forest $\hat{F}_p$, we add all edges in E_p^s into $\hat{F}_p$ and then replace every crossing edge by corresponding path in G. Denote G_p as the resulting graph, we have

Lemma 3. *G_p contains a Steiner tree interconnecting X.*

Proof. In order to prove this statement, it is sufficient to show (1) $X \subseteq V(G_p)$; (2) G_p is connected.

Obviously, vertices in the interior area of a partition $X_p \subseteq V(G_p)$. If a vertex is in the boundary area of a partition, it must be on one of the crossing edges included in the set E_p, which has already been added into $V(G_p)$. So $X \subseteq V(G_p)$. It is sufficient to show G_p is connected. For convenience, we keep G_p as it is without replacing every crossing edge by corresponding path, i.e., G_p is obtained from $\hat{F}_p$ by adding all edges in E_p^s. Clearly, if this G_p is connected, after replacing every crossing edge, the resulting graph G_p is also connected.

Now, suppose to the contrary that G_p is disconnected. Then, G_p can be divided into two disjoint subgraphs G_p^1 and G_p^2 such that there are no edges connecting G_p^1 and G_p^2 in G_p. Since G_p is obtained from $\hat{F}_p$ by adding all edges in E_p^s, there are some terminal vertices contained in G_p^1 and G_p^2. Since T_s is a spanning tree of terminal set X in G, there is an edge L in T_s connecting G_p^1 and G_p^2. Because all crossing edges are added in G_p, the edge L must be a non-crossing edge. Therefore, L is contained in some cell. Denote u and v as the endpoints of this edge. Also let T_u and T_v be the two trees containing u and v in the cell, respectively. Since c is the maximum edge weight among all edges in T_s, we have $dist_w(u, v) \leq c$. On the other hand, from Lemma 2, we have $dist_w(u, v) > c$. This derives a contradiction. Hence, G_p is connected. □

Recall that there are all together k^2 different partitions and for every partition $P_{i,j}$, we could obtain a graph $G_{P_{i,j}}$. Among all k^2 graphs, we choose the minimum-weight graph and prune it into a Steiner tree on X. This final tree, denoted as T_{out}, is the output of our algorithm.

5 Theoretical Analysis

In this section, we study the approximation ratio of our algorithm and show that for any $\varepsilon > 0$, choosing appropriate integer k, the approximation ratio is $1 + \varepsilon$.

There are two steps in our proof. In the first step, we show that for any partition P, $C(\hat{F}_p) \leq C(T_{OPT})$, where T_{OPT} is the optimal solution for node-weighted Steiner tree on terminal set X. In the second step, we show that our algorithm has a performance ratio of $1 + \varepsilon$.

5.1 $C(\hat{F}_P) \leq C(T_{OPT})$

Let T_p be the minimum Steiner tree in G on X_p. Since T_{OPT} is also a Steiner tree on X_p, clearly $C(T_p) \leq C(T_{OPT})$. In order to prove $C(\hat{F}_P) \leq C(T_{OPT})$, we construct a new Steiner forest F_p on X_p, which is modified from T_p satisfying that each tree in F_p is completely included in a cell and also $C(\hat{F}_p) \leq C(F_p)$. Following gives some useful notations for further proof.

For any Steiner tree, we call a Steiner vertex a *real Steiner vertex* if its degree is at least 3. Besides, a path between two vertices in the Steiner tree is a *Stem* if its endpoints are either a terminal vertex or a real Steiner vertex and also all the other vertices are 2-degree Steiner vertices. We modify T_p to be the desired forest F_p with the following 3 steps.

In the first step, we delete all stems with weight greater than c in T_p, and denote the resulting forest by F'_p. After this, the w-distance between any two trees in F'_p is greater than c because of the optimality of T_p. Also we have $C(F'_p) \leq C(T_p)$.

In the second step, we further modify F'_p to guarantee that each tree in it is interconnecting terminal vertices in the same cell. If there is a tree T^* in F'_p connecting terminal vertices in different cells, T^* must have some Steiner vertices between the boundary areas of two adjacent cells since the e-distance between any two vertices is at most 1. By Steiner vertices, we mean those vertices not belong to the X_p. If we draw two vertical lines, the distance between which is $2c$ as illustrated in the figure 3, there must exist a vertex u in the tree T^* within these two lines since the e-distance between any two vertices is at most 1. Therefore, the e-distance between u and any boundaries of the interior area is more than $1.5c \log k$. Since the weight of any stem in F'_p is at most c and the weight of any vertex is at least 1, the e-distance between any two adjacent real Steiner vertices is at most c, where two real Steiner vertices are *adjacent* if they can be connected without any other real Steiner vertices. Now, we count the number of real Steiner vertices to connect the vertex u and any boundary of the interior area. Clearly, it must use at least

$$1 + 2 + \cdots + 2^{(1.5 \log k)-1} = 2^{1.5 \log k} - 1 = k^{1.5} - 1$$

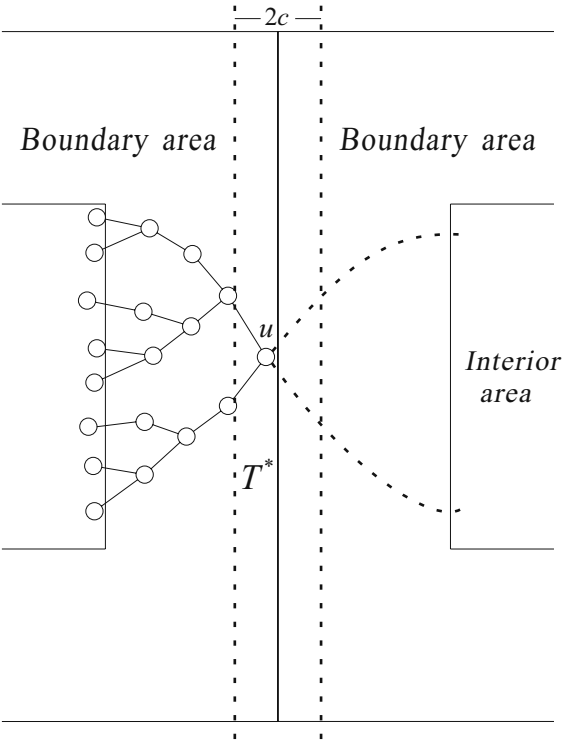


Fig. 3. The tree T^* and the vertex u

real Steiner vertices in the tree T^*(see figure 3). Hence, there are at least

$$k^{1.5} - 1 + (k^{1.5} - 1)(c - 1) = c(k^{1.5} - 1)$$

Steiner vertices in the tree between u and the boundary of interior area. By deleting all of these vertices, at most k more trees will be created along this boundary. If the w-distance of any two trees is at most c, connect them to be a new tree. As there are at most k trees, the whole weight will increase at most $c(k-1)$. Meanwhile, as we delete at least $c(k^{1.5} - 1)$ vertices, which means the whole weight decreases at least $c(k^{1.5} - 1)$. Hence, the weight of the new F'_p is decreased by doing so. Repeating this step until there are no trees connecting different cells, denoted the resulting forest by F''_p. We can see that the w-distance between any two trees in F''_p is also greater than c and $C(F''_p) \leq C(F'_p)$.

In the last step, if any tree of F''_p is completely included in a cell, we do nothing. Otherwise, there must exist a tree such that all its terminal vertices are in a same cell, but at least one Steiner vertex is in a different cell. In this case, we modify F''_p using the same method described in Step 2. Clearly, any tree in the new F''_p is completely in one cell. Finally, for any tree, we reconstruct Steiner minimum tree on its terminal vertices in the cell. Let F_p be the resulting graph afterwards, we can obtain the following lemmas.

Lemma 4. *F_p is a Steriner forest on T_p with the following properties:*
(1) Each tree of F_p is completely included in some cell.
(2) The w-distance between any two trees in F_p is greater than c. Furthermore, the w-distance between any two terminal vertices in different trees of F_p is greater than c.
(3) $C(F_p) \leq C(T_p) \leq C(T_{OPT})$.

Lemma 5. $C(\hat{F}_p) \leq C(F_p) \leq C(T_{OPT})$.

Proof. It is only necessary to show $C(\hat{F}_p) \leq C(F_p)$. Recalling the constructions of $\hat{F}_p$ and F_p, for a fixed cell, every Y_i is completely contained in one tree of F_p. Hence, F_p will be one of possible merging solutions as well. Since $\hat{F}_p$ is the minimum solution of all possible merging choices, we have $C(\hat{F}_p) \leq C(F_p)$. □

5.2 Performance Analysis

Based on Lemma 5 and the construction of T_{out}, we obtain the main theorem in this paper.

Theorem 1. *The approximation ratio for the NWST problem used in our algorithm to interconnect the terminal set is* $1 + 40c\lceil 1 + 1.5 \log k \rceil / k$.

Proof. Recall that T_{out} is consisted of two parts, local optimal $\hat{F}_p$ and $E_{p_{i,j}}$. To bound total weight of the $E_{p_{i,j}}$, we consider the number of times each vertex in the terminal set appears in the boundary area in all k^2 partitions.

If we divide every cell into 1×1 squares, for different partitions, the same terminal vertex must lie in different square according to the shifting strategy we used. Since there are at most $4ck\lceil 1 + 1.5 \log k \rceil$ squares in boundary area, a terminal vertex will appear in the boundary area at most $4ck\lceil 1 + 1.5 \log k \rceil$ times. For an edge in T_s, since both of its endpoints are terminal vertices, it will be considered as a crossing edge at most $2 \times 4ck\lceil 1 + 1.5 \log k \rceil$ times in all k^2 partitions. Hence, we have

$$\begin{aligned}
\sum_{0 \leq i,j \leq k-1} C(G_{p_{i,j}}) &\leq k^2 C(\hat{F}_p) + \sum_{0 \leq i,j \leq k-1} C(E_{p_{i,j}}) \\
&\leq k^2 C(T_{OPT}) + \sum_{0 \leq i,j \leq k-1} C(E_{p_{i,j}}) \\
&\leq k^2 C(T_{OPT}) + 8ck\lceil 1 + 1.5 \log k \rceil C(T_s) \\
&\leq k^2 C(T_{OPT}) + 8ck\lceil 1 + 1.5 \log k \rceil \times 5C(T_{OPT}) \\
&\leq k^2 C(T_{OPT}) + 40ck\lceil 1 + 1.5 \log k \rceil C(T_{OPT}).
\end{aligned}$$

Therefore, we have

$$C(T_{out}) \leq \left(\sum_{0 \leq i,j \leq k-1} C(G_{p_{i,j}}) \right) / k^2 \leq \left(1 + 40c\lceil 1 + 1.5 \log k \rceil / k \right) C(T_{OPT}).$$

The proof is then finished. □

Corollary 1. *For any given $\varepsilon > 0$, let $k > \lceil (41\,c/\varepsilon)^2 \rceil$. Then $C(T_{out}) \leq (1+\varepsilon)\, C(T_{OPT})$.*

5.3 Application on MWCDS

As an application, we apply NWST algorithm into MWCDS problem. Recall that the problem of MWCDS is to construct the connected dominating set in a node-weighted graph with the minimum total weight. Normally, researchers start with calculating dominating set for the graph first and then interconnecting them. Obviously, the node-weighted Steriner tree can be used in the MWCDS problem to interconnect all nodes of the DS to get better approximation algorithm. Therefore, we can obtain the following results.

Theorem 2. *There is a (4+ε)-approximation algorithm for minimum weighted dominating set problem. (Proof is omitted due to lack of space.)*

Corollary 2. *There is a (5+ε)-approximation algorithm for MWCDS by using node-weighted Steriner tree to interconnect all nodes of the DS.*

Proof. For any node-weighted graph G and a given Dominating Set DS of G, denote OPT_{CDS} and T_{OPT} be the optimal CDS of the G and the optimal Steiner tree of G on the given DS, respectively. Since the induced graph $G[DS \bigcup OPT_{CDS}]$ is connected, this graph contains a Steiner tree of G on DS. Hence, we have $C(T_{OPT}) \leq C(DS) + C(OPT_{CDS})$.

By Theorem 2, for any $\varepsilon > 0$, we can obtain a dominating set D of G with $C(D) \leq (4+\varepsilon/7)\, C(OPT_{CDS})$. Then, using our algorithm for D, we can get a Steiner tree T interconnecting D with $C(T) \leq (1+\varepsilon/7)\, C(T_{OPT})$. Since D is a dominating set, clearly, $V(T)$ is a connected dominating set of G and

$$\begin{aligned} C(T) &\leq \big(1+\frac{\varepsilon}{7}\big)\, C(T_{OPT}) \\ &\leq \big(1+\frac{\varepsilon}{7}\big)\, (\, C(D) + C(OPT_{CDS})\,) \\ &\leq \big(1+\frac{\varepsilon}{7}\big)\big(4+\frac{\varepsilon}{7}\big)\, C(OPT_{CDS}) + \big(1+\frac{\varepsilon}{7}\big) C(OPT_{CDS}) \\ &\leq \big(4+6\frac{\varepsilon}{7}\big)\, C(OPT_{CDS}) + \big(1+\frac{\varepsilon}{7}\big) C(OPT_{CDS}) \\ &\leq (5+\varepsilon)\, C(OPT_{CDS}). \end{aligned}$$

The proof is then finished. □

6 Conclusion and Discussion

In this paper, adopting the strategy of partition and shifting, we propose a $(1+\varepsilon)$-approximation algorithm for NWST problem in unit disk graphs, which is the best solution for this problem we could ever have without proving P=NP. As an application, we give a (5+ε)-approximation solution for MWCDS problem

in unit disk graphs afterwards, by interconnecting the DS constructed by the $(4+\varepsilon)$-approximation algorithm using our PTAS solution for NWST problem, which better bounds the performance of the MWCDS compared with existing algorithms.

References

1. Ambühl, C., Erlebach, T., Mihalák, M., Nunkesser, M.: Constant-factor Approximation for Minimum-weight (Connect) Dominating Sets in Unit Disk Graphs. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 3–14. Springer, Heidelberg (2006)
2. Baker, B.S.: Approximation Algorithm for NP-Complete Problems on Planar Graphs. Journal of the ACM 41, 153–180 (1994)
3. Berman, P.: Personal Communciation (1991)
4. Berman, P., Ramaiyer, V.: Improved Approximations for the Steiner Tree Problem. Journal of Algorithms 17, 381–408 (1994)
5. Chen, D., Du, D.Z., Hu, X.D., Lin, G.H., Wang, L., Xue, G.: Approximation for Steienr Trees with Minimum Number of Steiner Points. Theoretical Computer Science 262, 83–99 (2001)
6. Dai, D., Yu, C.: A 5-Approximation Algorithm for Minimum Weighted Dominating Set in Unit Disk Graph. Theoretical Computer Science (to appear)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-completeness. Freeman, New York (1979)
8. Guha, S., Khuller, S.: Improved Methods of Approximating Node Weighted Steiner Tree and Connected Dominating Sets. Information and Computation 150, 57–74 (1999)
9. Hochbaum, D.S., Maass, W.: Approximation Schemes for Covering and Packing Problems in Image Processing and VLSI. Journal of the ACM 32, 130–136 (1985)
10. Hougardy, S., Prömel, H.J.: A 1.598-Approximation Algorithm for the Steiner Problem in Graphs. In: Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 448–453 (1999)
11. Huang, Y., Gao, X., Zhang, Z., Wu, W.: A Better Constant-Factor Approximation for Weighted Dominating Set in Unit Disk Graph. Jounral of Combibatorial Optimization (to appear)
12. Hwang, F.K., Richards, D.S.: Steiner tree problems. Networks 22, 55–89 (1992)
13. Klein, P., Ravi, R.: A Nearly Best-Possible Approximation Algorithm for Node-Weighted Steiner Trees. Journal of Algorithms 19, 104–115 (1995)
14. Kou, L.T., Markowsky, G., Berman, L.: A Fast Algorithm for Steiner Trees. Acta Informatica 15(2), 141–145 (1981)
15. Lichtenstein, D.: Planar Formulae and their Uses. SIAM Journal on Computing 11, 329–343 (1982)
16. Robins, G., Salowe, J.S.: On the Maximum Degree of Minimum Spanning Trees. In: Proceedings of the ACM Symposium on Computational Geometry, pp. 250–258 (1994)
17. Robins, G., Zelikovski, A.: Improved Steiner Tree Approximation in Graphs. In: Proc. of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 770–779 (2000)

18. Wang, L., Jiang, T.: An Approximation Scheme for Some Steiner Tree Problem in the Plane. Networks 28, 187–193 (1996)
19. Wu, W., Du, H., Jia, X., Li, Y., Huang, S.C.H.: Minimum Connected Dominating Sets and Maximal Independent Sets in Unit Disk Graphs. Theor. Comput. Sci. 352, 1–7 (2006)
20. Zelikovsky, A.: An 11/6 Approximation Algorithm for the Network Steiner Problem. Algorithmica 9, 463–470 (1993)
21. Zou, F., Li, X., Kim, D., Wu, W.: Two Constant Approximation Algorithms for Node-Weighted Steiner Tree in Unit Disk Graphs. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 278–285. Springer, Heidelberg (2008)

DNA Library Screening, Pooling Design and Unitary Spaces*

Suogang Gao[1], Zengti Li[2], Jiangchen Yu[3], Xiaofeng Gao[3], and Weili Wu[3]

[1] Mathematics and Information College, Hebei Normal University
Shijiazhuang 050016, China
sggao@hebtu.edu.cn
[2] Mathematics and Information College, Langfang Normal College
Langfang 065000, China
[3] Department of Computer Science, University of Texas at Dallas
Richardson, Texas 75080, USA
weiliwu@utdallas.edu

Abstract. Pooling design is an important research topic in bioinformatics due to its wide applications in molecular biology, especially DNA library screening. In this paper, with unitary spaces over finite fields, we present two new constructions whose efficiency ratio, i.e., the ratio between the number of tests and the number of items, is smaller than some of existing construction.

Keywords: Pooling design, s^e-disjunct matrix, unitary space.

1 Introduction

A recent important research area in molecular biology is the study of gene functions. Such a study is often supported by a high quality DNA library which usually obtained through a large amount of testing and screening. Consequently, the efficiency of testing and screening becomes critical to the the success and ultimately the impact of the study. Pooling design is a data mining method which can deal with data in such an efficient way so that the number of tests can be reduced significantly. For example, the Life Science Division of Los Alamos National Laboratories reported [7] that they was facing 220,000 clones in their study and did only 376 tests with help from pooling design while individual testing requires 220,000 tests.

In pooling design, a clone is also referred as an *item*. Suppose we face a set of n items with some positive ones. The problem is to identify all positive items with a smaller number of tests. There are several models of pooling design with different testing mechanisms for different applications [1]. The classical model

* Support in part by Natural Science Foundation of Hebei Province, China (No. A2008000128), Natural Science Foundation of Educational Committee of Hebei Province, China (No. 2007137), and National Science Foundation under grants CCF 0621829 and 0627233.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 49–60, 2009.

has a simplest testing mechanism. Each test is on a subset of items, called a *pool*. When the pool contains a positive items, the test-outcome is *positive*; otherwise, the test-outcome is *negative*. Usually, the pooling design requires that all pools are constructed at the beginning of testing so that all tests can be performed simultaneously. Therefore, a pooling design can be represented by a binary matrix. Such a binary matrix has rows indexed with pools and columns indexed with items and the entry at intersection of row p_i and column j is 1 if and only $j \in p_i$.

A pooling design or corresponding binary matrix is said to be d^e*-disjunct* if for any $d+1$ columns of $j_0, j_1, ..., j_d$, there exist $e+1$ rows p_{i_1}, ..., $p_{i_{e+1}}$ in each of which the entry at column j_0 is 1 and entries at $j_1, ..., j_d$ are 0s. Every d^e-disjunct pooling design can identify up to d positive items in the situation that there may exist at most e errors in test-outcomes. Furthermore, a d^e-disjunct matrix is said to be fully d^e-disjunct if in addition, it is not $(d')^{e'}$-disjunct for $d' > d$ or $e' > e$.

There are several constructions for the d^e-disjunct pooling design and for the fully d^e-disjunct pooling design [2,3,4,5,6,8,9,10]. In this paper, we present two new constructions. Those constructions are based on study of unitary spaces over finite fields. We will show that those constructions can provide some pooling designs with smaller ratio between the number of pools and the number of items, compared with some important existing constructions.

2 Unitary Space

Let $\mathbb{F}_{q^2}$ be a finite field with q^2 elements, where q is a power of a prime. $\mathbb{F}_{q^2}$ has an *involutive automorphism* $a \mapsto \bar{a} = a^q$, and the fixed field of this automorphism is $\mathbb{F}_q$. Let $n = 2\nu + \delta$, where $\delta = 0$ or 1, and let

$$H_\delta = \begin{cases} \begin{pmatrix} 0 & I^{(\nu)} \\ I^{(\nu)} & 0 \end{pmatrix}, & \text{if } \delta = 0, \\ \begin{pmatrix} 0 & I^{(\nu)} & \\ I^{(\nu)} & 0 & \\ & & 1 \end{pmatrix}, & \text{if } \delta = 1. \end{cases}$$

The *unitary group* of degree n over $\mathbb{F}_{q^2}$, denoted by $U_n(\mathbb{F}_{q^2})$, consists of all $n \times n$ matrix T over $\mathbb{F}_{q^2}$ satisfying $TH_\delta(\bar{T})^t = H_\delta$. There is an action of $U_n(\mathbb{F}_{q^2})$ on $\mathbb{F}_{q^2}^{(n)}$ defined by

$$\mathbb{F}_{q^2}^{(n)} \times U_n(\mathbb{F}_{q^2}) \longrightarrow \mathbb{F}_{q^2}^{(n)}$$
$$((x_1, x_2, \cdots, x_n), T) \longmapsto (x_1, x_2, \cdots, x_n)T.$$

The vector space $\mathbb{F}_{q^2}^{(n)}$ together with the above group action of the unitary group $U_n(\mathbb{F}_{q^2})$, is called the *n-dimensional unitary space over* $\mathbb{F}_{q^2}$. An m-dimensional subspace P is said to be of type (m, r), if $PK(\bar{P})^t$ is of rank r. In particular, subspaces of type $(m, 0)$ are called m-dimensional *totally isotropic subspaces.*The subspaces of type (m, r) exist if and only if $2r \leq 2m \leq n + r$. The subspace

of type (m,r), which contains subspaces of type (m_1,r_1), exists if and only if $2r \le 2m \le n+r, 2r_1 \le 2m_1 \le n+r_1$ and $0 \le r-r_1 \le 2(m-m_1)$. From [11], the number of subspaces of type (m,r), denoted by $N(m,r;n)$, is given by

$$N(m,r;n) = q^{r(n+r-2m)} \frac{\prod_{i=n+r-2m+1}^{n} (q^i - (-1)^i)}{\prod_{i=1}^{r} (q^i - (-1)^i) \prod_{i=1}^{m-r} (q^{2i}-1)}. \tag{1}$$

Let $N(m_1,r;m,r;n)$ denote the number of subspaces of type (m_1,r) contained in a given subspace of type (m,r). From [11]

$$N(m_1,r;m,r;n) = q^{2r(m-m_1)} \frac{\prod_{i=m-m_1+1}^{m-r} (q^{2i}-1)}{\prod_{i=1}^{m_1-r} (q^{2i}-1)}. \tag{2}$$

Let $N'(m_1,r;m,r;n)$ denote the number of subspaces of type (m,r) containing a given subspace of type (m_1,r). From [11]

$$N'(m_1,r;m,r;n) = \frac{\prod_{i=1}^{n+r-2m_1} (q^i - (-1)^i)}{\prod_{i=1}^{n+r-2m} (q^i - (-1)^i) \prod_{i=1}^{m-m_1} (q^{2i}-1)}. \tag{3}$$

Lemma 1. *Let $\mathbb{F}_{q^2}^{(n)}$ denote the n-dimensional unitary space over a finite field $\mathbb{F}_{q^2}$, with $2r \le 2m_0 \le 2i \le 2m \le n+r, r = 2s+\delta_1$ and $\delta_1 \le \delta$, where $\delta_1 = 0$, or $\delta_1 = 1$. Fix a subspace W_0 of type (m_0,r) in $\mathbb{F}_{q^2}^{(n)}$, and a subspace W of type (m,r) in $\mathbb{F}_{q^2}^{(n)}$ such that $W_0 \subset W$. Then the number of subspaces A of type (i,r) in $\mathbb{F}_{q^2}^{(n)}$, where $W_0 \subset A \subset W$, is $N(i-m_0,0;m-m_0,0;2(\nu+s-m_0))$.*

Proof. Let $\sigma = \nu + s - m_0$. Since the unitary group $U_n(\mathbb{F}_{q^2})$ acts transitively on each set of subspaces of the same type, we may assume that W has the matrix representation of the form

$$W = \begin{array}{c} \begin{array}{ccccccccc} s & m_0-2s & \sigma & s & m_0-2s & \sigma & \delta_1 & \delta-\delta_1 & \end{array} \\ \begin{pmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & W_1 & 0 & 0 & W_2 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} s \\ s \\ \delta_1 \\ m_0-2s-\delta_1 \\ m-m_0 \end{array}$$

where (W_1, W_2) is a subspace of type $(m-m_0, 0)$ in $\mathbb{F}_{q^2}^{(2(\nu+s-m_0))}$. By (2), the number of subspaces A of type (i,r), where $W_0 \subset A \subset W$, is $N(i-m_0,0;m-m_0,0;2(\nu+s-m_0))$. □

3 Construction I

Definition 1. *For $2r \le 2d_0 < 2d < 2k \le n + r$, assume that P_0 is a fixed subspace of type (d_0, r) in $\mathbb{F}_{q^2}^{(n)}$. Let M be a binary matrix whose columns (rows) are indexed by all spaces of type (k, r) containing P_0 (spaces of type (d, r) containing P_0) in $\mathbb{F}_{q^2}^{(n)}$ such that $M(A, B) = 1$ if $A \subseteq B$ and 0 otherwise. This matrix is denoted by $M_1(n, d, k)$.*

Theorem 1. *Suppose $2r \le 2d_0 < 2d < 2k \le n + r, r = 2s + \delta_1$, where $\delta_1 = 0, 1$ and set $b = \frac{q^2(q^{2(k-d_0-1)}-1)}{q^{2(k-d)}-1}$. Then $M_1(n, d, k)$ is $l^e - disjunct$ for $1 \le l \le b$ and*

$$\begin{aligned} e = {} & q^{2(k-d)} N(d - d_0 - 1, 0; k - d_0 - 1, 0; 2(\nu + s - d_0)) \\ & -(l-1) q^{2(k-d-1)} N(d - d_0 - 1, 0; k - d_0 - 2, 0; 2(\nu + s - d_0)). \end{aligned}$$

Proof. Let $C, C_1, \cdots, C_l$ be $l + 1$ distinct columns of $M_1(n, d, k)$. To obtain the maximum numbers of subspaces of type (d, r) containing P_0 in

$$C \cap \bigcup_{i=1}^{s} C_i = \bigcup_{i=1}^{s} (C \cap C_i),$$

we may assume that each $C \cap C_i$ is a subspace of type $(k - 1, r)$.

Then each $C \cap C_i$ covers $N(d - d_0, 0; k - d_0 - 1, 0; 2(\nu + s - d_0))$ subspaces of type (d, r) containing P_0 from Lemma 1. However, the coverage of each pair of C_i and C_j overlaps at a subspace of type $(k-2, r)$ containing P_0, where $1 \le i, j \le s$. Therefore, from Lemma 1 only C_1 covers the full $N(d - d_0, 0; k - d_0 - 1, 0; 2(\nu + s - d_0))$ subspaces of type (d, r) containing P_0, while each of $C_2, \cdots, C_l$ can cover a maximum of $N(d - d_0, 0; k - d_0 - 1, 0; 2(\nu + s - d_0)) - N(d - d_0, 0; k - d_0 - 2, 0; 2(\nu + s - d_0))$ subspaces of type (d, r) not covered by C_1. By (2), the number of the subspaces of type (d, r) of C not covered by $C_1, C_2, \cdots, C_l$ is at least

$$\begin{aligned} e = {} & N(d - d_0, 0; k - d_0, 0; 2(\nu + s - d_0)) \\ & -N(d - d_0, 0; k - d_0 - 1, 0; 2(\nu + s - d_0)) \\ & -(l-1)(N(d - d_0, 0; k - d_0 - 1, 0; 2(\nu + s - d_0)) \\ & -N(d - d_0, 0; k - d_0 - 2, 0; 2(\nu + s - d_0))) \\ = {} & q^{2(k-d)} N(d - d_0 - 1, 0; k - d_0 - 1, 0; 2(\nu + s - d_0)) \\ & -(l-1) q^{2(k-d-1)} N(d - d_0 - 1, 0; k - d_0 - 2, 0; 2(\nu + s - d_0)). \end{aligned}$$

Note that $\dfrac{N(d - d_0 - 1, 0; k - d_0 - 1, 0; 2(\nu + s - d_0))}{N(d - d_0 - 1, 0; k - d_0 - 2, 0; 2(\nu + s - d_0))} = \dfrac{q^{2(k-d_0-1)} - 1}{q^{2(k-d)} - 1}$. Since $e > 0$,

$$l < \frac{q^2(q^{2(k-d_0-1)} - 1)}{q^{2(k-d)} - 1} + 1.$$

Set

$$b = \frac{q^2(q^{2(k-d_0-1)} - 1)}{q^{2(k-d)} - 1}.$$

Then $1 \le l \le b$. □

Corollary 1. *Suppose that* $2r \le 2d_0 < 2d < 2k \le n + r, r = 2s + \delta_1$ *and* $1 \le l \le \min\{b, q^2 + 1\}$. *Then* $M_1(n, d, k)$ *is not* l^{e+1}*-disjunct, where* b *and* l *are as in Theorem 1.*

Proof. Let C be a subspace of type (k, r) containing P_0, and E be a fixed subspace of type $(k-2, r)$ containing P_0 and contained in C. By Lemma 1, we obtain the number of subspaces of type $(k-1, r)$ containing E and contained in C is

$$N(1, 0; 2, 0; 2(\nu + s - k + 2)) = q^2 + 1.$$

For $1 \le l \le \min\{b, q^2 + 1\}$, we choose l distinct subspaces of type $(k-1, r)$ containing E and contained in C, denote these subspaces by $Q_i (1 \le i \le l)$. For each Q_i, we choose a subspace C_i of type (k, r) such that $C \cap C_i = Q_i (1 \le i \le l)$. Hence, each pair of C_i and C_j overlaps at the same subspace E of type $(k-2, r)$, where $1 \le i, j \le l$. By Theorem 1, we have desired result. □

Corollary 2. *Suppose that* $d = d_0 + 1$ *and* $1 \le l \le q^2$. *Then* $M_1(n, d, k)$ *is* l^e*-disjunct, but is not* l^{e+1}*-disjunct, where* $e = q^{2(k-d_0-2)}(q^2 - l + 1)$.

Proof. Setting $d = d_0 + 1$ in the e formula of Theorem 1, we obtain

$$e = q^{2(k-d_0-2)}(q^2 - l + 1).$$

The second statement follows directly from Corollary 1. □

The following theorem tells us how to choose k so that the test to item ratio is minimized.

Theorem 2. *For* $2r \le 2m_0 < 2m \le n + r$, *the sequence* $N'(m_0, r; m, r; n)$ *is unimodal and gets its peak at* $m = \lfloor \frac{n+r+m_0}{3} \rfloor$ *or* $m = \lfloor \frac{n+r+m_0}{3} \rfloor + 1$.

Proof. For $2r \le 2m_0 < 2m \le n + r$, by (3), we have

$$\begin{aligned}
\frac{N'(m_0, r; m_1, r; n)}{N'(m_0, r; m_2, r; n)} &= \frac{\prod_{i=m_1-m_0+1}^{m_2-m_0} (q^{2i} - 1)}{\prod_{i=n+r-2m_2+1}^{n+r-2m_1} (q^i - (-1)^i)} \\
&= \frac{\prod_{i=0}^{m_2-m_1-1} (q^{2(m_1-m_0+1+i)} - 1)}{\prod_{i=0}^{2(m_2-m_1)-1} (q^{n+r-2m_2+1+i} - (-1)^{n+r-2m_2+1+i})} \qquad (4) \\
&= \prod_{i=0}^{m_2-m_1-1} \frac{q^{m_1-m_0+1+i} - 1}{q^{n+r-2m_2+1+2i} - (-1)^{n+r-2m_2+1+2i}} \\
&\times \prod_{i=0}^{m_2-m_1-1} \frac{q^{m_1-m_0+1+i} + 1}{q^{n+r-2m_2+1+2i+1} - (-1)^{n+r-2m_2+1+2i+1}}.
\end{aligned}$$

If $\lfloor \frac{n+r+m_0}{3} \rfloor + 1 \le m_1 < m_2$, then $\frac{n+r+m_0}{3} < m_1$. It implies that

$$2m_1 + m_2 > 3m_1 > n + r + m_0. \qquad (5)$$

Since $i \leq m_2 - m_1 - 1$, by (5) we have

$$m_1 + 2m_2 > n + r + m_0 + 1 + (m_2 - m_1 - 1) \geq n + r + m_0 + 1 + i.$$

Thus $m_1 - m_0 + 1 + i > n + r - 2m_2 + 2i + 2$. It follows that

$$q^{m_1 - m_0 + 1 + i} > q^{n+r-2m_2+2i+2} > q^{n+r-2m_2+2i+1}.$$

Therefore,

$$\frac{q^{m_1 - m_0 + 1 + i} - 1}{q^{n+r-2m_2+1+2i} - (-1)^{n+r-2m_2+1+2i}} > 1$$

and

$$\frac{q^{m_1 - m_0 + 1 + i} + 1}{q^{n+r-2m_2+1+2i} - (-1)^{n+r-2m_2+1+2i+1}} > 1.$$

From (4) we have $N'(m_0, r; m_2, r; n) < N'(m_0, r; m_1, r; n)$.

If $m_0 \leq m_1 < m_2 \leq \lfloor \frac{n + r + m_0}{3} \rfloor$, then $m_2 \leq \frac{n+r+m_0}{3}$. Thus

$$m_1 + 2m_2 < 3m_2 \leq n + r + m_0 < n + r + m_0 + i.$$

It follows that $m_1 - m_0 + 1 + i < n + r - 2m_2 + 1 + 2i$. So

$$q^{m_1 - m_0 + 1 + i} < q^{n+r-2m_2+1+2i} < q^{n+r-2m_2+1+2i+1}.$$

It follows that

$$\frac{q^{m_1 - m_0 + 1 + i} - 1}{q^{n+r-2m_2+1+2i} - (-1)^{n+r-2m_2+1+2i}} < 1$$

and

$$\frac{q^{m_1 - m_0 + 1 + i} + 1}{q^{n+r-2m_2+1+2i} - (-1)^{n+r-2m_2+1+2i+1}} < 1.$$

From (4) we have $N'(m_0, r; m_2, r; n) > N'(m_0, r; m_1, r; n)$. □

4 The Discussions of Test Efficiency

Identifying most positive items with the least tests is one of our goals. Therefore, discussing how to make the ratio t/n smaller is significative. In our matrix,

$$t_1/n_1 = \frac{N'(d_0, r; d, r; n)}{N'(d_0, r; k, r; n)} = \frac{\prod_{i=d-d_0+1}^{k-d_0} (q^{2i} - 1)}{\prod_{i=n+r-2k+1}^{n+r-2d} (q^i - (-1)^i)}.$$

We first will explain several facts on the ratio:

(1) Parameter $d_0(n, r)$ only appears in the numerator (denominator). It is easy to show that the larger the d_0, n and r are, the smaller the ratio is.

(2) Noting that the increasing speed of $(q^i-(-1)^i)(q^{i+1}-(-1)^{i+1})$ is larger than $(q^{2i}-1)$, so the smaller the d and k are, the smaller the ratio is.

D'yachkov et al. [2] constructed with subspaces of $GF(q)$, where q is a prime power, each of the columns (rows) is labeled by an $k(d)$-dimensional space, $m_{ij}=1$ if and only if the label of row i is contained in the label of column j. In order to compare with t/n, we should take the dimension of the space of $GF(q)$ to be $2(\nu+s-d_0)$, and replace q by q^2, where $n=2\nu+\delta, r=2s+\delta_1$. Assume that the test efficiency is t_2/n_2, then

$$\frac{t_2}{n_2}=\frac{\begin{bmatrix}2(\nu+s-d_0)\\ d\end{bmatrix}_{q^2}}{\begin{bmatrix}2(\nu+s-d_0)\\ k\end{bmatrix}_{q^2}}=\frac{\prod\limits_{i=d+1}^{k}(q^{2i}-1)}{\prod\limits_{i=2(\nu+s-d_0)-k+1}^{2(\nu+s-d_0)-d}(q^{2i}-1)}.$$

Theorem 3. *If $2d_0>k-\delta-\delta_1$, then $t_1/n_1<q^{2d_0(d-k)}t_2/n_2$, where $\frac{k-\delta-\delta_1}{2}<d_0<d<k$.*

Proof. We have

$$\begin{aligned}
\frac{\frac{t_1}{n_1}}{\frac{t_2}{n_2}}&=\frac{\prod\limits_{i=d-d_0+1}^{k-d_0}(q^{2i}-1)}{\prod\limits_{i=n+r-2k+1}^{n+r-2d}(q^i-(-1)^i)}\Big/\frac{\prod\limits_{i=d+1}^{k}(q^{2i}-1)}{\prod\limits_{i=2(\nu+s-d_0)-k+1}^{2(\nu+s-d_0)-d}(q^{2i}-1)}\\
&=\frac{\prod\limits_{i=0}^{k-d-1}(q^{2(d-d_0+1+i)}-1)}{\prod\limits_{i=0}^{2k-2d-1}(q^{n+r-2k+1+i}-(-1)^{n+r-2k+1+i})}\Big/\frac{\prod\limits_{i=0}^{k-d-1}(q^{2(d+1+i)}-1)}{\prod\limits_{i=0}^{k-d-1}(q^{2(2\nu+2s-2d_0-k+1+i)}-1)}\\
&=\prod_{i=0}^{k-d-1}\frac{q^{2(d-d_0+1+i)}-1}{q^{2(d+1+i)}-1}\frac{\prod\limits_{i=0}^{k-d-1}q^{2(2\nu+2s-2d_0-k+1+i)}-1}{\prod\limits_{i=0}^{2k-2d-1}(q^{n+r-2k+1+i}-(-1)^{n+r-2k+1+i})}\\
&<\prod_{i=0}^{k-d-1}\frac{q^{2(d-d_0+1+i)}}{q^{2(d+1+i)}}\\
&\quad\times\prod_{i=0}^{k-d-1}\left(\frac{q^{2\nu+2s-2d_0-k+1+i}-1}{q^{n+r-2k+1+2i}-(-1)^{n+r-2k+1+2i}}\frac{q^{2\nu+2s-2d_0-k+1+i}+1}{q^{n+r-2k+1+2i+1}-(-1)^{n+r-2k+1+2i+1}}\right)\\
&=\frac{1}{q^{2d_0(k-d)}}\prod_{i=0}^{k-d-1}\left(\frac{q^{2\nu+2s-2d_0-k+1+i}-1}{q^{n+r-2k+1+2i}-(-1)^{n+r-2k+1+2i}}\frac{q^{2\nu+2s-2d_0-k+1+i}+1}{q^{n+r-2k+1+2i+1}-(-1)^{n+r-2k+1+2i+1}}\right).
\end{aligned}$$

Since $2d_0>k-\delta-\delta_1$, and $n=2\nu+\delta, r=2s+\delta_1$, we have

$$\begin{aligned}
&\frac{q^{2\nu+2s-2d_0-k+1+i}-1}{q^{n+r-2k+1+2i}-(-1)^{n+r-2k+1+2i}}\\
&=\frac{q^{2(\nu+s)-k+1+i-2d_0}-1}{q^{2(\nu+s)-k+1+2i-(k-\delta-\delta_1)}-(-1)^{n+r-2k+1+2i}}<1,
\end{aligned}$$

and

$$\frac{q^{2\nu+2s-2d_0-k+1+i}+1}{q^{n+r-2k+1+2i+1}-(-1)^{n+r-2k+1+2i+1}}$$
$$=\frac{q^{2(\nu+s)-k+1+i-2d_0}+1}{q^{2(\nu+s)-k+1+2i-(k-\delta-\delta_1-1)}-(-1)^{n+r-2k+1+2i+1}}<1.$$

Therefore, $t_1/n_1 < q^{2d_0(d-k)}t_2/n_2$, where $\frac{k-\delta-\delta_1}{2} < d_0 < d < k$. □

5 Construction II

Definition 2. *For $2 \le 2r \le 2d < 2k \le n+r$, let M be a binary matrix whose columns (rows) are indexed by all subspaces of type (k,r) ((d,r)) in $\mathbb{F}_{q^2}^{(n)}$ such that $M(A,B)=1$ if $A \subseteq B$ and 0 otherwise. This matrix is denoted by $M_2(n,d,k)$.*

Theorem 4. *Suppose $0 \le 2r-4 \le 2d < 2k-2 \le n+r-2$. If $1 \le s \le q^{2r}$, then $M_2(n,d,k)$ is $s^e-disjunct$, where*

$$e = q^{2(k-d-1)d+2r}.$$

Proof. Let $C, C_1, \cdots, C_s$ be $s+1$ distinct columns of $M_2(n,d,k)$. To obtain the maximum number of subspaces of type (d,r) in

$$C \cap \bigcup_{i=1}^{s} C_i = \bigcup_{i=1}^{s} (C \cap C_i),$$

we may assume that each $C \cap C_i$ is a subspace of type $(k-1,r)$, where $1 \le i \le s$. By (2), the number of the subspaces of type (d,r) of C not covered by $C_1, C_2, \cdots, C_s$ is at least

$$N(d,r;k,r;n) - sN(d,r;k-1,r;n)$$
$$= q^{2r(k-d-1)} \frac{\prod_{i=k-d+1}^{k-r-1}(q^{2i}-1)}{\prod_{i=1}^{d-r}(q^{2i}-1)} \big(q^{2k}-q^{2r}-s(q^{2(k-d)}-1)\big).$$

Since $0 \le 2r-4 \le 2d < 2k-2 \le n+r-2$, we obtain

$$\frac{\prod_{i=k-d+1}^{k-r-1}(q^{2i}-1)}{\prod_{i=1}^{d-r}(q^{2i}-1)}$$
$$= \frac{\prod_{i=0}^{d-r-2}(q^{2(i+k-d+1)}-1)}{\prod_{i=0}^{d-r-2}(q^{2(i+1)}-1)} \frac{1}{q^{2(d-r)}-1}$$

$$= \prod_{i=0}^{d-r-2} \frac{q^{2(i+k-d+1)}-1}{q^{2(i+1)}-1} \frac{1}{q^{2(d-r)}-1}$$
$$= \prod_{i=0}^{d-r-2} q^{2(k-d)} \frac{q^{2(i+1)}-\frac{1}{q^{2(k-d)}}}{q^{2(i+1)}-1} \frac{1}{q^{2(d-r)}-1} > q^{2(k-d)(d-r-1)-2(d-r)}.$$

Since $1 \le s \le q^{2r}$, we obtain

$$q^{2k}-q^{2r}-s(q^{2(k-d)}-1) \ge q^{2k}-q^{2r}-q^{2r}(q^{2(k-d)}-1)$$
$$= q^{2k-2d+2r}(q^{2(d-r)}-1) > q^{2k-2d+2r}.$$

Hence $e = q^{2(k-d-1)d+2r}$. □

Theorem 5. *Suppose* $0 \le 2r-4 \le 2d < 2k-2 \le n+r-2$. *Let* $p = \frac{q^{2(d+1)}-q^{2r}}{q^2-1} - 1$. *If* $1 \le s \le p$, *then* $M_2(n,d,d+1)$ *is fully* s^e*-disjunct, where* $e = p-s$.

Proof. By (2), we have $N(d,r;d+1,r;n) = p+1$. It follows that we can pick $s+1$ distinct subspaces $C, C_1, \ldots, C_s$ of type $(d+1,r)$ such that $C \cap C_i$ and $C \cap C_j$ are two distinct subspaces of type (d,r), where $1 \le i,j \le s$. By the principle of inclusion and exclusion, the number of subspaces of type (d,r) in C but not in each C_i is $p-s+1$, where $1 \le i \le s$. It follows that $e \le p-s$.

On the other hand, similar to the proof of Theorem 4 we obtain

$$e \ge N(d,r;d+1,r;n) - s - 1 = p - s.$$

Hence $e = p-s$. □

The following theorem tells us how to choose k so that the test to item ratio is minimized.

Theorem 6. *Fox fixed integers* $r < n$, $N(m,r;n)$ *is a monotonic decreasing in* $m \in [\lfloor \frac{n+r}{3} \rfloor + 1, \lfloor \frac{n+r}{2} \rfloor]$ *and a monotonic increasing in* $m \in [r, \lfloor \frac{n+r}{3} \rfloor]$.

Proof. For any $\lfloor \frac{n+r}{3} \rfloor + 1 \le m_1 < m_2 \le \lfloor \frac{n+r}{2} \rfloor$, by (1), we have

$$\frac{N(m_2,r;n)}{N(m_1,r;n)} = \frac{q^{n+r-2m_1}-(-1)^{n+r-2m_1}}{q^r(q^{m_1-r+1}+1)} \cdot \frac{q^{n+r-2m_1-1}-(-1)^{n+r-2m_1-1}}{q^r(q^{m_1-r+1}-1)}$$
$$\cdot \frac{q^{n+r-2m_1-2}-(-1)^{n+r-2m_1-2}}{q^r(q^{m_1-r+2}+1)} \cdot \frac{q^{n+r-2m_1-3}-(-1)^{n+r-2m_1-3}}{q^r(q^{m_1-r+2}-1)}$$
$$\cdot \ldots$$
$$\cdot \frac{q^{n+r-2m_2+2}-(-1)^{n+r-2m_2+2}}{q^r(q^{m_2-r}+1)} \cdot \frac{q^{n+r-2m_2+1}-(-1)^{n+r-2m_2+1}}{q^r(q^{m_2-r}-1)}.$$

Since $\lfloor \frac{n+r}{3} \rfloor + 1 \le m_1 < m_2 \le \lfloor \frac{n+r}{2} \rfloor$, then $\frac{n+r}{3} \le m_1$. Thus

$$m_1 + 1 > n + r - 2m_1.$$

It follows that

$$q^{n+r-2m_1} < q^{m_1+1}.$$

So

$$q^{n+r-2m_1} - (-1)^{n+r-2m_1} < q^{m_1+1} + q^r = q^r(q^{m_1-r+1}+1).$$

That is

$$1 > \frac{q^{n+r-2m_1} - (-1)^{n+r-2m_1}}{q^r(q^{m_1-r+1}+1)}.$$

Note that

$$\begin{aligned}
&\frac{q^{n+r-2m_1} - (-1)^{n+r-2m_1}}{q^r(q^{m_1-r+1}+1)} > \frac{q^{n+r-2m_1-1} - (-1)^{n+r-2m_1-1}}{q^r(q^{m_1-r+1}-1)} \\
&> \frac{q^{n+r-2m_1-2} - (-1)^{n+r-2m_1-2}}{q^r(q^{m_1-r+2}+1)} > \frac{q^{n+r-2m_1-3} - (-1)^{n+r-2m_1-3}}{q^r(q^{m_1-r+2}-1)} \\
&> \cdots \\
&> \frac{q^{n+r-2m_2+2} - (-1)^{n+r-2m_2+2}}{q^r(q^{m_2-r}+1)} > \frac{q^{n+r-2m_2+1} - (-1)^{n+r-2m_2+1}}{q^r(q^{m_2-r}-1)}.
\end{aligned}$$

Hence $\frac{N(m_2,r;n)}{N(m_1,r;n)} < 1$. That is $N(m_2,r;n) < N(m_1,r;n)$.

For any $r \le m_1 < m_2 \le \lfloor \frac{n+r}{3} \rfloor$, by (1), we have

$$\begin{aligned}
\frac{N(m_2,r;n)}{N(m_1,r;n)} = &\frac{q^{n+r-2m_1} - (-1)^{n+r-2m_1}}{q^{m_1+1}+q^r} \cdot \frac{q^{n+r-2m_1-1} - (-1)^{n+r-2m_1-1}}{q^{m_1+1}-q^r} \\
&\cdot \frac{q^{n+r-2m_1-2} - (-1)^{n+r-2m_1-2}}{q^{m_1+2}+q^r} \cdot \frac{q^{n+r-2m_1-3} - (-1)^{n+r-2m_1-3}}{q^{m_1+2}-q^r} \\
&\cdot \cdots \\
&\cdot \frac{q^{n+r-2m_2+2} - (-1)^{n+r-2m_2+2}}{q^{m_2}+q^r} \cdot \frac{q^{n+r-2m_2+1} - (-1)^{n+r-2m_2+1}}{q^{m_2}-q^r}.
\end{aligned}$$

Since $r \le m_1 < m_2 \le \lfloor \frac{n+r}{3} \rfloor$, then $m_2 \le \frac{n+r}{3} < \frac{n+r+1}{3}$. It implies that

$$m_2 - r < n - 2m_2 + 1.$$

So

$$q^{n-2m_2+1} - \frac{(-1)^{n+r-2m_2+1}}{q^r} > q^{n-2m_2+1} - 1 > q^{m_2-r} - 1.$$

It follows that

$$\frac{q^{n-2m_2+1} - \frac{(-1)^{n+r-2m_2+1}}{q^r}}{q^{m_2-r}-1} > 1.$$

Thus

$$\frac{q^{n+r-2m_2+1} - (-1)^{n+r-2m_2+1}}{q^{m_2}-q^r} > 1.$$

Note that

$$\begin{aligned}
&\frac{q^{n+r-2m_1}-(-1)^{n+r-2m_1}}{q^{m_1+1}+q^r} > \frac{q^{n+r-2m_1-1}-(-1)^{n+r-2m_1-1}}{q^{m_1+1}-q^r}\\
&> \frac{q^{n+r-2m_1-2}-(-1)^{n+r-2m_1-2}}{q^{m_1+2}+q^r} > \frac{q^{n+r-2m_1-3}-(-1)^{n+r-2m_1-3}}{q^{m_1+2}-q^r}\\
&> \cdots\\
&> \frac{q^{n+r-2m_2+2}-(-1)^{n+r-2m_2+2}}{q^{m_2}+q^r} > \frac{q^{n+r-2m_2+1}-(-1)^{n+r-2m_2+1}}{q^{m_2}-q^r}.
\end{aligned}$$

Hence $\frac{N(m_2,r;n)}{N(m_1,r;n)} > 1$. That is $N(m_2,r;n) > N(m_1,r;n)$. □

Theorem 7. *If $d = r, k = r+1 < \frac{n}{2}$, then the test efficiency of construction II is smaller than that of [2]*

Proof. If $d = r$ and $k = r+1$, then the disjunct matrix of construction II is $M_2(n,r,r+1)$ and the disjunct matrix of [2] is $M(n,r+1,r)$. Let $\frac{t}{n}$ be the test efficiency of $M_2(n,r,r+1)$ and let $\frac{t_1}{n_1}$ be the test efficiency of $M(n,r+1,r)$, respectively.
Then

$$\begin{aligned}
\frac{t}{n} &= \frac{N(d,r;n)}{N(k,r;n)}\\
&= \frac{N(r,r;n)}{N(r+1,r;n)}\\
&= q^{r(n+r-2r)}\frac{\prod_{i=n+r-2r+1}^{n}(q^i-(-1)^i)}{\prod_{i=1}^{r}(q^i-(-1)^i)\prod_{i=1}^{r-r}(q^{2i}-1)}\cdot\frac{\prod_{i=1}^{r}(q^i-(-1)^i)\prod_{i=1}^{r+1-r}(q^{2i}-1)}{q^{r(n+r-2(r+1))}\prod_{i=n+r-2(r+1)+1}^{n}(q^i-(-1)^i)}\\
&= \frac{q^{r+1}-q^r}{q^{n-r}-(-1)^{n-r}}\cdot\frac{q^{r+1}+q^r}{q^{n-r-1}-(-1)^{n-r-1}},
\end{aligned}$$

and

$$\frac{t_1}{n_1} = \frac{\begin{bmatrix} n \\ d \end{bmatrix}_q}{\begin{bmatrix} n \\ k \end{bmatrix}_q} = \frac{\prod_{i=d+1}^{k}(q^i-1)}{\prod_{i=n-k+1}^{n-d}(q^i-1)} = \frac{q^{r+1}-1}{q^{n-r}-1}.$$

Since $r+1 < \frac{n}{2}$, we have

$$\frac{q^{r+1}+q^r}{q^{n-r-1}-(-1)^{n-r-1}} < 1.$$

Therefore, $\frac{t}{n} < \frac{t_1}{n_1}$. □

References

1. Du, D., Hwang, F.K.: Pooling Designs and Nonadaptive Group Testing. World Scientific, Singapore (2006)
2. D'yachkov, A.G., Hwang, F.K., Macula, A.J., Vilenkin, P.A., Weng, C.: A Construction of Pooling Designs with Some Happy Surprises. J. Computational Biology 12, 1129–1136 (2005)
3. D'yachkov, A.G., Macula, A.J., Vilenkin, P.A.: Nonadaptive Group and Trivial Two Stage Group Testing with Error-Correction d^e-Disjunct Inclusion Matrices. In: Csiszár, I., Katona, G.O.H., Tardos, G. (eds.) Entropy, Search, Complexity, 1st edn., pp. 71–84. Springer, Berlin (2007)
4. Huang, T., Weng, C.: Pooling Spaces and Non-adaptive Pooling Designs. Discrete Math. 282, 163–169 (2004)
5. Li, Z., Gao, S., Du, H., Zou, F., Wu, W.: Two Constructions of New Error-correcting Pooling Designs from Orthogonal Spaces over a Finite Field of Characteristic 2. J. Comb. Optim. (to appear)
6. Macula, A.J.: A Simple Construction of d-Disjunct Matrices with Certain Constant Weights. Discrete Math. 162, 311–312 (1996)
7. Marathe, M.V., Percus, A.G., Torney, D.C.: Combinatorial Optimization in Biology (manuscript, 2000)
8. Ngo, H., Du, D.: New Constructions of Non-adaptive and Error-tolerance Pooling Designs. Discrete Math. 243, 161–170 (2002)
9. Ngo, H., Du, D.: A Survey on Combinatorial Group Testing Algorithms with Applications to DNA Library Screening. DIMACS Ser. Discrete Math. Theoretical Comp. Sci. 55, 171–182 (2000)
10. Park, H., Wu, W., Liu, Z., Wu, X., Zhao, H.G.: DNA Screening, Pooling Design and Simplicial Complex. J. Comb. Optim. 7, 289–394 (2003)
11. Wan, Z.: Geometry of Classical Groups over Finite Fields, 2nd edn. Science Press, Beijing (2002)

Improved Algorithms for the Gene Team Problem*

Biing-Feng Wang, Shang-Ju Liu, and Chien-Hsin Lin

Department of Computer Science, National Tsing Hua University Hsinchu
Taiwan 30043, R. China
bfwang@cs.nthu.edu.tw,
{phoenix,jsing}@venus.cs.nthu.edu.tw

Abstract. A *gene team* is a set of genes that appear in two or more species, possibly in a different order yet with the distance between adjacent genes in the team for each chromosome always no more than a certain threshold. The focus of this paper is the problem of finding gene teams of two chromosomes. Béal *et al.* [1] had an $O(n \log^2 n)$-time algorithm for this problem. In this paper, two $O(n \log d)$-time algorithms are proposed, where $d \leq n$ is the number of gene teams. The proposed algorithms are obtained by modifying Béal *et al.*'s algorithm, using two different approaches. Béal *et al.*'s algorithm can be extended to find the gene teams of k chromosomes in $O(kn \log^2 n)$ time. Our improved algorithms can be extended to find the gene teams of k chromosomes in $O(kn \log d)$ time.

Keywords: bioinformatics, comparative genomics, conserved gene clusters, gene teams, algorithms.

1 Introduction

Comparing multiple genome sequences is an important method to discover new biological insights. If a group of genes remain physically close to each other in multiple genomes, often called a *conserved gene cluster*, then the genes may be either historically or functionally related [15]. For example, Overbeek *et al.* [13] showed that the functional dependency of proteins can be inferred by considering conserved gene clusters in multiple genomes. This conservation of spatial clustering of genes has also been used to predict features of interest, such as operons and physical interactions of proteins [6,4,10,11]. Therefore, identifying conserved gene clusters is an important step towards understanding the evolution of genomes and predicting the functions of genes.

In recent years, several models had been developed to capture the essential biological features of a conserved gene cluster. The first such model was introduced by Uno and Yagiura [16], in which a genome sequence is considered as a

* This research is supported by the National Science Council of the Republic of China under grant NSC-97-2221-E-007-053-MY3.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 61–72, 2009.

permutation of distinct genes and a *common interval* is defined to be a set of genes that appear consecutively, possibly in different orders, in two given permutations. Uno and Yagiura had an algorithm that finds all common intervals of two permutations of n genes in $O(n+K)$ time, using $O(n)$ space, where K is the number of common intervals. Heber and Stoye [9] extended this work to find all common intervals of k permutations in $O(kn+K)$ time, using $O(kn)$ space. In addition, Didier [5] extended this model to include paralogs by considering a sequence definition more general than a strict permutation, and gave an algorithm that finds all common intervals of two sequences in $O(n^2 \log n)$ time, using $O(n)$ space, on the extended model. Later, Schmidt and Stoye [14] improved this result to $O(n^2)$ time. Schmidt and Stoye's algorithm can be extend to find all common intervals of k sequences in $O(kn^2)$ time, using $O(n^2)$ additional space.

In the definition of common intervals, a single misplaced gene disqualifies the commonality between two otherwise similar intervals. To remedy this drawback, Béal *et al.* [1,12] introduced the concept of *gene teams*. A *gene team* is a set of genes that appear in two or more species, possibly in a different order yet with the distance between adjacent genes in the team for each chromosome always no more than a certain threshold. Similar to the model in [16], Béal *et al.* considered a chromosome as a permutation of distinct genes. They gave an efficient algorithm that finds the gene teams of two chromosomes in $O(n \log^2 n)$ time, using $O(n)$ space. He and Goldwasser [8] generalized the gene team model to handle general sequences, in which multiple copies of the same gene are allowed. They had an algorithm that finds gene teams of two general sequences in $O(mn)$ time, using $O(m+n)$ space, where m and n are, respectively, the numbers of genes in the two given sequence.

The focus of this paper is the problem of finding gene teams of two chromosomes. As mentioned above, Béal *et al.* had an $O(n \log^2 n)$-time algorithm for this problem. In this paper, two improved algorithms are proposed. Both of the proposed algorithms require $O(n \log d)$ time, using $O(n)$ space, where $d \leq n$ is the number of gene teams. Our algorithms are obtained by modifying Béal *et al.*'s algorithm, using two different approaches. Béal *et al.*'s algorithm can be extended to find the gene teams of k chromosomes in $O(kn \log^2 n)$ time [1]. Our improved algorithms are modified versions of theirs, and thus can also be easily extended to find the gene teams of k chromosomes in $O(kn \log d)$ time.

The remainder of this paper is organized as follows. Section 2 gives notation and definitions that are used throughout this paper. Section 3 reviews Béal *et al.*'s algorithm. Then, our two improved algorithms are proposed in Sections 4 and 5, respectively. Finally, concluding remarks are given in Section 6.

2 Notation and Definitions

Let Σ be a set of n genes. A *gene order* G of Σ is a permutation of the genes in Σ, in which each gene α is associated with a real number $P_G(\alpha)$, called its *position*, such that for any two genes α and β in G, $P_G(\alpha) \leq P_G(\beta)$ if α appears

before β. Let G be a gene order of Σ. The *distance* of two genes α and β in G is given by $|P_G(\alpha) - P_G(\beta)|$. For any subset $S \subseteq \Sigma$, let $G(S)$ be the gene order of S that is obtained from G by deleting the genes not in S. Let $\delta \geq 0$ be a real number. A subset $C \subseteq \Sigma$ is a *δ-chain* of G if the distance between any two consecutive genes in $G(C)$ is at most δ. A δ-chain C of G is *maximal* if there dose not exist any δ-chain C' of G such that $C' \supset C$.

Example 1. Consider $\Sigma = \{a, b, c, d, e, f\}, \delta = 3$, and a gene order $G = (d_1, e_2, c_3, a_4, f_5, b_8)$, where the letters represent genes and the numbers in the subscript denote positions. Then, $G(\{c, d, f\}) = (d_1, c_3, f_5)$. The set $\{c, d, f\}$ is a δ-chain of G since each pair of consecutive genes in $G(\{c, d, f\})$ is distant by less than δ; but it is not maximal, since $\{c, d, e, f\}$ is also a δ-chain of G. The set $\{d, f\}$ is not a δ-chain of G, since the distance between d and f is 4.

Let G_1 and G_2 be two gene orders of Σ. A subset $S \subseteq \Sigma$ is a *δ-set* of G_1 and G_2 if it is a δ-chain of both G_1 and G_2. A *δ-team* of G_1 and G_2 is a δ-set that is maximal with respect to set inclusion. Given two gene orders G_1, G_2 of Σ and a real number $\delta \geq 0$, the *gene team problem* is to identify all the δ-teams of G_1 and G_2. It is easy to see that the set of δ-teams of G_1 and G_2 form a partition of Σ.

Example 2. Consider $\Sigma = \{a, b, c, d, e, f\}, \delta = 3$, and two gene orders $G_1 = (d_1, e_2, c_3, a_4, f_5, b_8)$ and $G_2 = (a_1, b_2, c_3, d_7, e_8, f_9)$. Then, $\{d, e\}, \{e, f\}, \{d, e, f\}$ are δ-sets. Since $\{d, f\}$ is not a δ-chain of $G_1, \{d, f\}$ is not a δ-set. The δ-teams of G_1 and G_2 are $\{a, c\}, \{b\}, \{d, e, f\}$.

3 Béal *et al.*'s Algorithm

In this section, we review Béal *et al.*'s algorithm in [1] for finding the δ-teams of two gene orders G_1 and G_2 of Σ. A *δ-league* of G_1 and G_2 is a union of δ-teams of G_1 and G_2. Trivially, Σ is a δ-league of G_1 and G_2. Let $Team(G_1, G_2)$ be the set of δ-teams of G_1 and G_2. Béal *et al.* observed that any maximal δ-chain of G_1 or of G_2 is a δ-league, and gave the following important property for finding the δ-teams of two gene orders.

Lemma 1. [1] *Let $C \subseteq \Sigma$ be a maximal δ-chain of G_1 or of G_2. Then, $Team(G_1, G_2) = Team(G_1(C), G_2(C)) \cup Team(G_1(\Sigma \backslash C), G_2(\Sigma \backslash C))$.*

Consider a gene order G of Σ. For convenience, we say that a maximal δ-chain C of G is *small* if the size of C is at most $|\Sigma|/2$. If G is not a δ-chain, a small maximal δ-chain of G can be found efficiently as follows. We scan the genes in G from the two ends simultaneously by using two pointers p_1 and p_2. The pointer p_1 examines the distance between each pair of consecutive genes in G from left to right, and the pointer p_2 examines the distance between each pair of consecutive genes in G from right to left. Initially, p_1 and p_2 point, respectively, to the first and last genes in G. Then, repeatedly, we do the following. If the current distance examined by p_1 is larger than δ, a small maximal δ-chain is

found, which contains the genes that have encountered by p_1; and if the current distance examined by p_2 is larger than δ, a small maximal δ-chain is found, which contains the genes that have encountered by p_2. Otherwise, move p_1 one step to the right and move p_2 one step to the left, and proceed to the next iteration until p_1 and p_2 cross in the middle. In case p_1 and p_2 cross in the middle, G is a δ-chain.

If G is not a δ-chain, the above procedure finds a small maximal δ-chain C in $O(|C|)$ time. Otherwise, the above procedure terminates in $O(|\Sigma|)$ time. If both G_1 and G_2 are δ-chains, Σ is a gene team. Otherwise, a small maximal δ-chain of G_1 or of G_2 can be found efficiently by a simple extension of the above procedure as follows: By using four pointers, we scan the genes in each of G_1 and G_2 from the two ends to the center simultaneously.

For any subset $S \subseteq \Sigma$, let $L_1(S)$ denote a doubled linked list storing $G_1(S)$ and $L_2(S)$ denote a doubled linked list storing $G_2(S)$. For convenience, define an operation as follows:

SmallChain$(L_1(S), L_2(S))$: if both $G_1(S)$ and $G_2(S)$ are δ-chains, return S; otherwise, return a small maximal δ-chain of $G_1(S)$ or of $G_2(S)$.

Based upon the above discussion, we have the following.

Lemma 2. [1] *SmallChain can be implemented in $O(|C|)$ time, where C is the output.*

Based on Lemmas 1 and 2, Béal *et al.*'s gave the following efficient algorithm for the gene team problem.

Algorithm 1. Gene-Teams
Input: two gene orders G_1, G_2 of Σ and a real number $\delta \geq 0$
Output: the set of all δ-teams of G_1 and G_2
begin
1 $GT \leftarrow \emptyset$; construct $L_1(\Sigma)$ and $L_2(\Sigma)$
2 FindTeam$(L_1(\Sigma), L_2(\Sigma))$
3 **output**(GT)
end

Procedure FindTeam$(L_1(S), L_2(S))$
begin
1 $C \leftarrow$ SmallChain$(L_1(S), L_2(S))$
2 **if** $C = S$ **then** $GT \leftarrow GT \cup \{S\}$ /* a δ-team S is found
3 **else**
4 **begin**
5 construct $L_1(C), L_1(S \backslash C), L_2(C)$, and $L_2(S \backslash C)$
6 FindTeam$(L_1(C), L_2(C))$ /* solve two sub-problems recursively
7 FindTeam$(L_1(S \backslash C), L_2(S \backslash C))$
8 **end**
end

Consider the running time of FindTeam. By Lemma 2, Line 1 requires $O(|C|)$ time. If both $G_1(S)$ and $G_2(S)$ are δ-chains, FindTeam terminates at Line 2 in $O(|S|)$ time. Assume that not both $G_1(S)$ and $G_2(S)$ are δ-chains. Line 5 is implemented in $O(|C| \log |C|)$ time as follows. By symmetry, we may assume that C is a small maximal δ-chain of $G_1(S)$. Then, since $L_1(S)$ is either the concatenation of $L_1(C)$ and $L_1(S \backslash C)$, or the concatenation of $L_1(S \backslash C)$ and $L_1(C)$, we can obtain $L_1(C)$ and $L_1(S \backslash C)$ in $O(1)$ time by simply splitting $L_1(S)$ into two parts. The list $L_2(S \backslash C)$ is obtained in $O(|C|)$ time from $L_2(S)$ by removing the nodes storing the genes in C. The construction of $L_2(C)$ is the bottleneck. It is done in $O(|C| \log |C|)$ time by sorting genes in C according to their positions in G_2. Let $T(n)$ be the time required for running FindTeam on two gene orders of size n. Then, the recursive calls in Lines 6 and 7 take $T(|C|) + T(|S \backslash C|)$ time, respectively. Therefore, we have $T(n) = \max_{1 \leq i \leq n/2}\{T(n-i) + T(i) + O(i \log i)\}$. By induction, it can be shown that $T(n) = O(n \log^2 n)$ [1]. Consequently, we have the following.

Theorem 1. [1] *Algorithm 1 solves the gene team problem in $O(n \log^2 n)$ time.*

Remark 1. Béal *et al.* showed that it is easy to modify FindTeam to find the δ-team containing a designated gene $\alpha \in \Sigma$ in $O(n \log n)$ time. In fact, FindTeam can be modified to do the finding in $O(n)$ time as described below. First, compute C as SmallChain$(L_1(S), L_2(S))$. If $C = S$, S is the desired δ-team. Otherwise, do the following: obtain $L_1(X)$ and $L_2(X)$ from $L_1(S)$ and $L_2(S)$ in $O(|S \backslash X|)$ time by removing genes not in X, and then call FindTeam$(L_1(X), L_2(X))$ recursively, where X denotes C if $\alpha \in C$ and denotes $S \backslash C$ otherwise.

4 The First Improved Algorithm

This section presents our first $O(n \log d)$-time algorithm for the gene team problem, where $d \leq n$ is the number of δ-teams. The presented algorithm is a modified version of Béal *et al.*'s algorithm. Given $L_1(S)$ and $L_2(S)$, their algorithm computes $Team(G_1(S), G_2(S))$ as follows. First, find a small maximal δ-chain of $G_1(S)$ or of $G_2(S)$. If $S = C$, then S is a δ-team; otherwise, construct $L_1(C), L_1(S \backslash C), L_2(C)$, and $L_2(S \backslash C)$, and compute $Team(G_1(C), G_2(C))$ and $Team(G_1(S \backslash C), G_2(S \backslash C))$ recursively. By symmetry, assume that C is a small maximal δ-chain of $G_1(S)$. Then, the bottleneck of their algorithm is the construction of $L_2(C)$. They implemented the construction in $O(|C| \log |C|)$ time by sorting genes in C according to their positions in G_2. It is well-known that sorting n integers in a range 1 to n^c, for any constant c, can be done in linear time [2]. Our improvement is obtained by reducing the construction time of $L_2(C)$ to amortized $O(|C|)$, based on a non-trivial application of integer sort.

During the course of our first algorithm, each gene α in G_1 is associated with an integer label such that integer sort may be applied to construct $L_1(C)$ according to the labels. Similarly, each gene α in G_2 is associated with an integer label. Our first improved algorithm is as follows.

Algorithm 2. MODIFIED-GENE-TEAMS
Input: two gene orders G_1, G_2 of Σ and a real number $\delta \geq 0$
Output: the set of all δ-teams of G_1 and G_2
begin
1 $GT \leftarrow \emptyset$; construct $L_1(\Sigma)$ and $L_2(\Sigma)$
2 MODIFIEDFINDTEAM($L_1(\Sigma), L_2(\Sigma)$)
3 **output**(GT)
end

Procedure MODIFIEDFINDTEAM($L_1(S), L_2(S)$)
begin
1 $k \leftarrow |S|$
2 label the genes in $L_1(S)$ and $L_2(S)$ from left to right,
using the integers in $[1, k]$
3 **repeat**
4 $C \leftarrow$ SMALLCHAIN($L_1(S), L_2(S)$)
5 **if** $C \neq S$ **then**
6 **begin**
7 construct $L_1(C), L_1(S \backslash C), L_2(C)$, and $L_2(S \backslash C)$
8 /* solve the subproblem induced by C recursively
9 MODIFIEDFINDTEAM($L_1(C), L_2(C)$)
10 /* solve the subproblem induced by $S \backslash C$ in next iteration
11 $S \leftarrow S \backslash C$
12 **end**
13 **until** $C = S$
14 $GT \leftarrow GT \cup \{S\}$ /* a δ-team S is found
end

The correctness of Algorithm 2 is ensured by Lemma 1. Consider a fixed iteration of the repeat-loop in MODIFIEDFINDTEAM. For ease of discussion, assume that C is a small maximal δ-chain of $G_1(S)$. Then, the bottleneck is the construction of $L_2(C)$ at Line 7. We do the construction of $L_2(C)$ as follows.

Case 1: $|C| \geq k^{1/2}$.
In this case, we construct $L_2(C)$ in $O(|C|)$ time by integer sort, according to the labels of the genes in C. Note that all labels are integers in $[1, k]$. In this case, $O(1)$ time is spent on each gene in C.

Case 2: $|C| < k^{1/2}$.
We construct $L_2(C)$ in $O(|C| \log |C|)$ time by using a comparison sort. In this case, $O(\log |C|)$ amortized time is spent for each gene in C.

Lemma 3. *Line 7 of MODIFIEDFINDTEAM takes $O(|C|)$ time if $|C| \geq k^{1/2}$, and takes $O(|C| \log |C|)$ time otherwise.*

We proceed to discuss the overall time complexity of Algorithm 2. For ease of discussion, we describe the execution of Algorithm 2 on a given gene orders G_1

and G_2 by a recursion tree RT, which is defined as follows. The nodes in RT are classified into two subsets X and Y. Each node $x \in X$ represents a δ-team, denoted by Q_x. Each node $y \in Y$ represents a subset $S_y \subseteq \Sigma$, indicating that there is a recursive call to ModifiedFindTeam with input $(L_1(S_y), L_2(S_y))$. All nodes $x \in X$ are leaves. For each node $y \in Y$, y is the parent of a node $x \in X$, if $Q_x \subseteq S_y$, indicating that Q_x is found at Line 14 of the recursive call corresponding to y; and y is the parent of another node $y' \in Y$, if $S_{y'} \subset S_y$, indicating that the recursive call corresponding to y' is issued by the call corresponding to y.

Clearly, the running time of Algorithm 2 is proportional to the total construction time of $L_1(S_y)$ and $L_2(S_y)$ for every node $y \in Y$. Consider the total amortized time spent on a fixed gene $\alpha \in \Sigma$ for the construction. Let $x \in X$ be the leaf with $\alpha \in Q_x$, and let $(y(1), y(2), ..., y(q), x)$ be the path from the root of RT to the leaf x. (See Fig 1.) Then, $\{S_{y(i)} \mid 1 \leq i \leq q\}$ is the collection of all subsets S_y in RT such that the construction of $L_1(S_y)$ and $L_2(S_y)$ involves α. Note that $y(1)$ is the root of RT and $S_{y(1)} = \Sigma$. According to Lines 4 and 9 of ModifiedFindTeam, a recursive call having input of size k may issue recursive calls having input of size at most $k/2$. Thus, $|S_{y(i)}| \leq |S_{y(i-1)}|/2$ for $1 < i \leq q$. Since $|S_{y(1)}| = n$ and $|S_{y(q)}| \geq |Q_x|$, we have $q \leq \log(n/|Q_x|)$. Let t_i be the amortized time spent on α for the construction of $L_1(S_{y(i)})$ and $L_2(S_{y(i)})$. If $|S_{y(i)}| \geq |S_{y(i-1)}|^{1/2}$, we call $y(i)$ a *cheap node* of α, since by Lemma 3, $t_i = O(1)$; otherwise, we call $y(i)$ an *expensive node* of α, and $t_i = O(\log |S_{y(i)}|)$ by Lemma 3.

Lemma 4. $t_1 + t_2 + ... + t_q = O(\log(n/|Q_x|))$.

Proof. Let $a_1 = \sum_{y(i) \text{ is cheap}} \{t_i\}$ and $a_2 = \sum_{y(i) \text{ is expensive}} \{t_i\}$. Since $q \leq \log(n/|Q_x|)$, we have $a_1 = O(\log(n/|Q_x|))$. If $|Q_x| \geq n^{1/2}$, α does not have any expensive node and thus $a_1 + a_2 = O(\log(n/|Q_x|))$. Therefore, the lemma

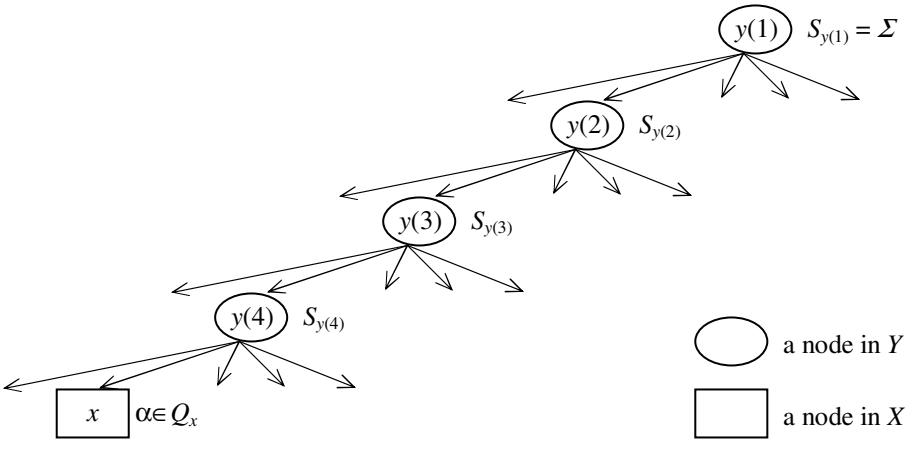


Fig. 1. An illustration of $(y(1), y(2), ..., y(q), x)$, in which $q = 4$

holds for $|Q_x| \geq n^{1/2}$. Assume that $|Q_x| < n^{1/2}$. Let $(e(1), e(2), ..., e(s))$ be the sequence of expensive nodes of α on the path from the root of RT to the leaf x. Clearly, $|S_{e(i)}| < |S_{e(i-1)}|^{1/2}$ for $1 < i \leq s$. Thus, it is easy to see that $|S_{e(i)}| < n^{(1/2)^i}$. And thus, we have $a_2 = O(\sum_{1 \leq i \leq s}(\log |S_{e(i)}|)) = O(\sum_{1 \leq i \leq s}((1/2)^i \log n)) = O(\log n)$. We had assumed that $|Q_x| < n^{1/2}$. Therefore, $a_2 = O(\log n) = O(\log(n/|Q_x|))$. Consequently, the lemma holds. □

Let GT be the set of δ-teams of G_1 and G_2. Then, by Lemma 4, the overall time complexity of Algorithm 2 is proportional to

$$\begin{array}{ll} & \sum_{Q \in GT} \sum_{\alpha \in Q} \log(n/|Q|) \\ \leq & \sum_{Q \in GT} |Q| \times \log(n/|Q|) \\ = & \sum_{Q \in GT} |Q| \times \log n - \sum_{Q \in GT} |Q| \times \log |Q| \\ = & n \log n - \sum_{Q \in GT} |Q| \times \log |Q|. \end{array}$$

Let d be the number of δ-teams in GT. It is easy to check that the minimum of $\sum_{Q \in GT} |Q| \times \log |Q|$ occurs when all δ-teams in GT are of the same size. Therefore, $\sum_{Q \in GT} \sum_{\alpha \in Q} \log(n/|Q|) \leq n \log n - d(n/d) \times \log(n/d) = n \log d$. We have the following.

Theorem 2. *Algorithm 2 solves the gene team problem in $O(n \log d)$ time, where d is the number of δ-teams.*

Remark 2. With a more careful analysis, it can be shown that the running time of Algorithm 1 is $O(n \log n \log d)$.

5 The Second Improved Algorithm

This section presents our second improved algorithm for the gene team problem, which also requires $O(n \log d)$-time. Similar to Algorithm 1, our second algorithm is designed based upon the following strategy: repeatedly partition two given gene orders into smaller gene orders according to Lemma 1, until all gene orders are δ-chains. Béal *et al.* implemented the above strategy by using recursive calls. To improve the running time, our second algorithm dose not use recursive calls. Instead, we maintain two *job queues* R and W, which are described as follows. Each job queue is a collection of subsets $S \subseteq \Sigma$. Each subset S in the queues indicates that we need to solve the sub-problem defined by $G_1(S)$ and $G_2(S)$. We call R the *ready queue*. Each subset S in R is represented by $(L_1(S), L_2(S))$. We call W the *waiting queue*. Each subset S in W is simply represented by S. Our algorithm works as follows. Initially, we compute an array A_1 storing the sequence of genes in G_1 and an array A_2 storing the sequence of genes in G_2. And, we set $R = \emptyset, W = \{\Sigma\}$, and $GT = \emptyset$. Then, we proceed to iterate as follows, until both job queues are empty. If R is empty, we do the following: construct $(L_1(S), L_2(S))$ and include it into R for all $S \in W$, and then empty the waiting queue W. Otherwise, we extract an element $(L_1(S), L_2(S))$ from R and do the following. First, we compute $C = \textsc{SmallChain}(L_1(S), L_2(S))$. If

$C = S$, we include the δ-team S into GT; otherwise, we construct $L_1(S\backslash C)$ and $L_2(S\backslash C)$, insert $(L_1(S\backslash C), L_2(S\backslash C))$ into R, and insert C into W. Then, we proceed to the next iteration. The above algorithm is formally described as follows.

Algorithm 3. NONRECURSIVE-GENE-TEAMS
Input: two gene orders G_1, G_2 of Σ and a real number $\delta \geq 0$
Output: the set of all δ-teams of G_1 and G_2
begin
1 $A_1 \leftarrow$ the sequence of genes in G_1; $A_2 \leftarrow$ the sequence of genes in G_2
2 $R \leftarrow \emptyset$; $W \leftarrow \{\Sigma\}$; $GT \leftarrow \emptyset$
3 **while** $(W \neq \emptyset)$ **do**
4 **begin**
5 MAKELIST /* move sub-problems in W into R
6 PARTITION /* partition each sub-problem in R by Lemma 1
7 **end**
8 **output**(GT)
end

Procedure MAKELIST
begin
1 **for** each $S \in W$ **do**
2 construct $L_1(S)$ and $L_2(S)$ and then insert $(L_1(S), L_2(S))$ into R
3 $W \leftarrow \emptyset$
end

Procedure PARTITION
begin
1 **while** $(R \neq \emptyset)$ **do**
2 **begin**
3 $(L_1(S), L_2(S)) \leftarrow$ an element extracted from R
4 $C \leftarrow$ SMALLCHAIN$(L_1(S), L_2(S))$
5 **if** $C = S$ **then** $GT \leftarrow GT \cup \{S\}$ /* a δ-team S is found
6 **else begin**
7 obtain $L_1(S\backslash C)$ and $L_2(S\backslash C)$ from $L_1(S)$ and $L_2(S)$
8 $R \leftarrow R \cup \{(L_1(S\backslash C), L_2(S\backslash C))\}$ /* insert $S\backslash C$ into R
9 $W \leftarrow W \cup \{C\}$ /* insert C into W
10 **end**
11 **end**
end

The correctness of Algorithm 3 is ensured by Lemma 1. The time complexity is analyzed as follows. First, we show that the while-loop in Lines 3–7 of Algorithm 3 performs $O(\log n)$ iterations. Consider a fixed iteration of the while-loop. At the beginning, R is empty. In Line 5, the call to MAKELIST moves all subsets in W into R. In Line 6, the call to PARTITION inserts a subset C into W only if

$|C| \leq |S|/2$ for some $S \in R$. Thus, at each iteration, the size of the largest subset in W is reduced by a factor of at least $1/2$. Before entering the while-loop, the largest subset in W is Σ. Therefore, the while-loop performs $O(\log n)$ iterations.

Lines 1 and 2 of Algorithm 3 require $O(n)$ time. In the following, we first show that the time complexity of Algorithm 3 is $O(n \log n)$ by showing that both MakeList and Partition require $O(n)$ time. MakeList needs to construct $L_1(S)$ and $L_2(S)$ for all $S \in W$. By symmetry, only the construction of $L_1(S)$ for all $S \in W$ is described. Let $S_1, S_2, ..., S_k$ be the subsets in W. Note that these subsets are mutually disjoint. Moreover, $S_1, S_2, ..., S_k$ and the subsets in GT form a partition of Σ. For any subset $S_i \in W$, the order of the genes in $L_1(S_i)$ is the same as their order in A_1. Therefore, all $L_1(S_i)$ can be constructed as follows. Initially, set $L_1(S_i) = \emptyset$ for each $i, 1 \leq i \leq k$. Then, we examine the genes in A_1 from left to right. If $A_1[j], 1 \leq j \leq n$, is not a gene of a set in GT, we append it to the tail of $L_1(S_i)$, where S_i is the subset containing $A_1[j]$. Clearly, the above construction takes $O(n)$ time. Next, consider the running time of Partition. At the beginning, there are at most n genes in the subsets of R. Each iteration of the while-loop takes $O(|C|)$ time and moves $|C|$ genes from R into GT or W (at Lines 5 and 9), until R is empty. Thus, the overall running time of Partition is $O(n)$. Consequently, the time complexity of Algorithm 3 is $O(n \log n)$.

In the following, we further show that with slight modifications, Algorithm 3 can be implemented in $O(n \log d)$ time. As mentioned, the while-loop of Algorithm 3 performs $O(\log n)$ iterations. At each iteration, the most critical step is Line 2 of MakeList, which needs to construct $L_1(S)$ and $L_2(S)$ for all $S \in W$. For $i \geq 1$, let Z_i be the union of the subsets in W at the beginning of the i-th iteration of the while-loop of Algorithm 3. Consider the i-th iteration for a fixed i. Clearly, the genes in A_1 and A_2 that are not in Z_i are useless to the construction of $L_1(S)$ and $L_2(S)$ for any subset $S \in W$. Therefore, to reduce the running time, we modify Algorithm 3 as follows: at the end of the i-th iteration of the while-loop, we remove from A_1 and A_2 those genes that are not in Z_{i+1}, so that at the beginning of the $(i+1)$-th iteration, A_1 and A_2 contain only the genes in Z_{i+1}.

With the above modification, the time complexity of Algorithm 3 is analyzed as follows. Consider the i-th iteration of the while-loop of Algorithm 3. Since A_1 and A_2 contain only the genes in Z_i, the call to MakeList at Line 5 takes $O(|Z_i|)$ time. After Line 5, the union of the subsets in R is Z_i. Therefore, the call to Partition at Line 6 also takes $O(|Z_i|)$ time. Before proceeding to the next iteration, we need to remove from A_1 and A_2 those genes not in Z_{i+1}. By a simple scan, such a removal is done in $O(|Z_i|)$ time. Therefore, the running time of the i-th iteration is $O(|Z_i|)$. Consequently, the overall time complexity of Algorithm 3 is $O(\sum |Z_i|)$.

Let GT be the set of δ-teams of G_1 and G_2 and let d be the number of δ-teams in GT. In the following, we show that $\sum |Z_i| = O(n \log d)$. Consider a δ-team $Q \in GT$. As mentioned, each iteration of the while-loop of Algorithm 3 reduces the size of the largest subset in W by a factor of at least $1/2$. Thus,

at the beginning of the i-th iteration, the size of the largest subset in W is at most $n/(2^{i-1})$. Therefore, $Q \subseteq Z_i$ only if $n/(2^{i-1}) \geq |Q|$. That is, each Q is found not later than the end of the $(\log(n/|Q|)+1)$-th iteration. Thus, the size of Q contributes to $\sum |Z_i|$ at most $\log(n/|Q|)+1$ times. And therefore, $\sum |Z_i| \leq \sum_{Q \in GT} |Q| \times (\log(n/|Q|)+1) = n+n\log n - \sum_{Q \in GT} |Q| \times \log |Q|$. As mentioned in Section 4, the minimum of $\sum_{Q \in GT} |Q| \times \log |Q|$ occurs when all δ-teams in GT are of the same size. Therefore, $\sum |Z_i| \leq n+n\log n - d(n/d) \times \log(n/d) = n+n\log d$. We have the following.

Theorem 3. *Algorithm 3 solves the gene team problem in $O(n \log d)$ time, where d is the number of δ-teams.*

It is nature to extend the definition of δ-teams to a set of gene orders $\{G_1, G_2, ..., G_k\}$. A gene order describes the structure of a linear chromosome. Also, it is nature to extend the definition of δ-teams to a set of circular chromosomes. As indicated in [1], Algorithm 1 can be easily extended to solve the problem of finding the δ-teams of a set of k gene orders or a set of k circular chromosomes in $O(kn \log^2 n)$ time. Similarly, Algorithms 2 and 3 can be easily extended to obtain the following.

Theorem 4. *The δ-teams of a set of k gene orders or a set of k circular chromosomes can be found in $O(kn \log d)$ time, where d is the number of δ-teams.*

6 Concluding Remarks

A graph problem, called the *common connected component problem* (*CCP*), was introduced by Gai *et al.* [7]. The gene team problem is a special case of the *CCP*, in which the input is two unit interval graphs. Coulon and Raffinot [3] extended Algorithm 1 to solve the *CCP* on k interval graphs in $O(kn \log^2 n)$ time. Similarly, Algorithms 2 and 3 can be easily extended to solve the *CCP* on a set of k interval graphs in $O(kn \log d)$ time, where $d \leq n$ is the number of common connected components.

He and Goldwasser [8] considered a generalization of the gene team problem, in which each of G_1 and G_2 may contain multiple copies of the same gene. They had an $O(mn)$-time, $O(m+n)$-space algorithm for the generalized problem, where m and n are, respectively, the lengths of G_1 and G_2. Let $D = \sum_{\alpha \in \Sigma} o_1(\alpha) \times o_2(\alpha)$, where $o_1(\alpha)$ and $o_2(\alpha)$ are, respectively, the numbers of copies of α in G_1 and G_2. We remark that with some efforts, both Algorithms 2 and 3 can be extended to solve the generalized problem in $O(\min\{D \log n, nm\})$ time. According to our experimental results, Algorithm 3 is more efficient than Algorithm 2. However, while being extended to solve the generalized gene team problem, Algorithm 2 uses less space than Algorithm 3, since Algorithm 2 solves sub-problems in a depth-first way, while Algorithm 3 solves sub-problems in a breadth-first way. The space requirements of the two extended algorithms are, respectively, $O(n+m)$ and $O(D)$.

References

1. Béal, M.-P., Bergeron, A., Corteel, S., Raffinot, M.: An Algorithmic View of Gene Teams. Theor. Comput. Sci. 320(2-3), 395–418 (2004)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
3. Coulon, F., Raffinot, M.: Fast Algorithms for identifying maximal common connected sets of interval graphs. Discrete Applied Mathematics 154(12), 1709–1721 (2006)
4. Dandekar, T., Snel, B., Huynen, M., Bork, P.: Conservation of Gene Order: a Fingerprint for Proteins that Physically Interact. Trends Biochem. Sci. 23, 324–328 (1998)
5. Didier, G.: Common Intervals of Two Sequences. In: Benson, G., Page, R.D.M. (eds.) WABI 2003. LNCS (LNBI), vol. 2812, pp. 17–24. Springer, Heidelberg (2003)
6. Ermolaeva, M.D., White, O., Salzberg, S.L.: Prediction of Operons in Microbial Genomes. Nucleic Acids Res. 29(5), 1216–1221 (2001)
7. Gai, A.-T., Habib, M., Paul, C., Raffinot, M.: Identifying Common Connected Components of Graphs. Technical Report, LIRMM-03016 (2003)
8. He, X., Goldwasser, M.H.: Identifying Conserved Gene Clusters in the Presence of Homology Families. Journal of Computational Biology 12(6), 638–656 (2005)
9. Heber, S., Stoye, J.: Finding all Common Intervals of k Permutations. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 207–218. Springer, Heidelberg (2001)
10. Lathe III, W.C., Snel, B., Bork, P.: Gene Context Conservation of a Higher Order than Operons. Trends Biochem. Sci. 25, 474–479 (2000)
11. Lawrence, J.: Selfish Operons: the Evolutionary Impact of Gene Clustering in Prokaryotes and Eukaryotes. Curr. Opin. Genet. Dev. 9(6), 642–648 (1999)
12. Luc, N., Risler, J.-L., Bergeron, A., Raffinot, M.: Gene Teams: a New Formalization of Gene Clusters for Comparative Genomics. Computational Biology and Chemistry 27(1), 59–67 (2003)
13. Overbeek, R., Fonstein, M., D'Souza, M., Pusch, G.D., Maltsev, N.: The Use of Gene Clusters to Infer Functional Coupling. Proc. Natl. Acad. Sci. USA 96(6), 2896–2901 (1999)
14. Schmidt, T., Stoye, J.: Quadratic Time Algorithms for Finding Common Intervals in Two and More Sequences. In: Sahinalp, S.C., Muthukrishnan, S.M., Dogrusoz, U. (eds.) CPM 2004. LNCS, vol. 3109, pp. 347–358. Springer, Heidelberg (2004)
15. Snel, B., Bork, P., Huynen, M.A.: The Identification of Functional Modules from the Enomic Association of Genes. Proc. Natl. Acad. Sci. USA 99(9), 5890–5895 (2002)
16. Uno, T., Yagiura, M.: Fast Algorithms to Enumerate all Common Intervals of Two Permutations. Algorithmica 26(2), 290–309 (2000)

Linear Coherent Bi-cluster Discovery via Line Detection and Sample Majority Voting

Yi Shi, Zhipeng Cai, Guohui Lin, and Dale Schuurmans

Department of Computing Science
University of Alberta, Edmonton, Alberta T6G 2E8
{ys3,zhipeng,ghlin,dale}@cs.ualberta.ca

Abstract. Discovering groups of genes that share common expression profiles is an important problem in DNA microarray analysis. Unfortunately, standard bi-clustering algorithms often fail to retrieve common expression groups because (1) genes only exhibit similar behaviors over a subset of conditions, and (2) genes may participate in more than one functional process and therefore belong to multiple groups. Many algorithms have been proposed to address these problems in the past decade; however, in addition to the above challenges most such algorithms are unable to discover linear coherent bi-clusters—a strict generalization of additive and multiplicative bi-clustering models. In this paper, we propose a novel bi-clustering algorithm that discovers linear coherent bi-clusters, based on first detecting linear correlations between pairs of gene expression profiles, then identifying groups by sample majority voting. Our experimental results on both synthetic and two real datasets, *Saccharomyces cerevisiae* and *Arabidopsis thaliana*, show significant performance improvements over previous methods. One intriguing aspect of our approach is that it can easily be extended to identify bi-clusters of more complex gene-gene correlations.

1 Introduction

Microarray analysis involves monitoring the expression levels of thousands of genes simultaneously over different conditions. Although such an emerging technology enables the language of biology to be spoken in mathematical terms, extracting useful information from the large volume of experimental microarray data remains a difficult challenge. One important problem in microarray analysis is to identify a subset of genes that have similar expression patterns under a common subset of conditions. Standard clustering methods, such as K-means clustering [8,5], hierarchical clustering [17,20] and self-organizing map [18], are usually not suitable for microarray data analysis for two main reasons: (1) genes exhibit similar behaviors not over all conditions, but over a subset of conditions, and (2) genes may participate in more than one functional processes and hence belong to multiple groups. Thus, traditional clustering algorithms typically do not produce a satisfactory solution. To overcome the limitations of the traditional clustering methods, the concept of bi-clustering was developed where one

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 73–84, 2009.

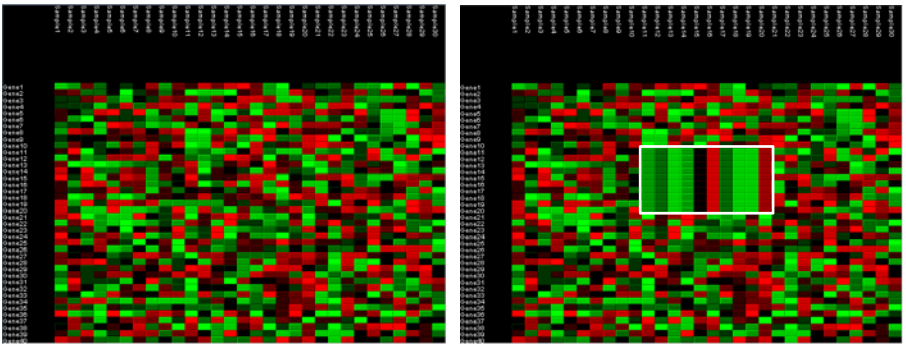


Fig. 1. Example of a constant row bi-cluster in gene expression matrix. The left image shows a gene expression matrix without any obvious bi-clusters; the right image shows an expression matrix with a constant row bi-cluster.

seeks groups of genes that exhibit similar expression patterns, but only over a subset of the sample conditions. Figure 1 illustrates a gene expression matrix without any obvious bi-clusters (left) and an expression matrix with a salient bi-cluster (right).

The term bi-clustering, also called co-clustering, or two-mode clustering was first mentioned by Hartigan in [7] and latter formalized by Mirkin in [14]. Cheng and Church [4] were the first to apply bi-clustering to gene expression analysis. Since then, dozens of bi-clustering algorithms have been proposed for the gene expression analysis. The general bi-clustering problem and many of its variants were proved to be NP-hard in [4], and therefore most bi-clustering algorithms comprise heuristic approaches unless special restrictions are made on the bi-cluster type and(or) bi-cluster structure. Among such bi-clustering algorithms, the majority assume that a expression matrix contains multiple bi-clusters rather than a single bi-cluster. Under the multiple bi-cluster circumstance, different bi-cluster structures can be considered, such as exclusive row and(or) column bi-clusters, checkerboard structure bi-clusters, non-overlapping tree-structured bi-clusters, non-overlapping non-exclusive bi-clusters, overlapping bi-clusters with hierarchical structure, arbitrarily positioned overlapping bi-clusters, and arbitrarily positioned overlapping bi-clusters [13]. The specific variant of the problem we address with the algorithm proposed in this paper, the Linear Coherent Bi-cluster Discovering (LCBD) algorithm, is the last form of bi-cluster structure; i.e., arbitrarily positioned overlapping bi-clusters. This last form is a a more general structure that covers most of the other bi-cluster structures.

Before designing a bi-clustering algorithm, one needs to determine what type (model) of individual bi-clusters to be looking for. There are six primary types considered in the literature, illustrated in Figure 2: (a) the constant value model, (b) the constant row model, (c) the constant column model, (d) the additive coherent model, where each row or column is obtained by adding a constant to

x	y	z	w
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0

(a)

x	y	z	w
1.2	1.2	1.2	1.2
0.8	0.8	0.8	0.8
1.5	1.5	1.5	1.5
0.6	0.6	0.6	0.6

(b)

x	y	z	w
1.2	0.8	1.5	0.6
1.2	0.8	1.5	0.6
1.2	0.8	1.5	0.6
1.2	0.8	1.5	0.6

(c)

x	y	z	w
1.2	0.8	1.5	0.6
1.0	0.6	1.3	0.4
2.0	1.6	2.3	1.4
0.7	0.3	1.2	0.3

(d)

x	y	z	w
2.0	4.0	8.0	1.0
1.0	2.0	4.0	0.5
4.0	8.0	16.0	2.0
1.0	2.0	4.0	0.5

(e)

x	y	z	w
2.0	4.0	3.0	5.0
1.5	2.5	2.0	3.0
2.3	4.3	3.3	5.3
4.5	8.5	6.5	10.5

(f)

Fig. 2. Examples of different bi-cluster types: (a) constant value model; (b) constant row model; (c) constant column model; (d) additive coherent model; (e) multiplicative coherent model; (f) linear coherent model

another row or column, (e) the multiplicative coherent model, where each row or column is obtained by multiplying another row or column by a constant value, and (f) the linear coherent model, where each column is obtained by multiplying another column by a constant value and then adding a constant [6]. To understand which type of bi-cluster structure makes the sense for gene expression analysis, one should note that the ultimate purpose is to identify pairs of biologically related genes such that, under certain conditions, one activates or deactivates the other, either directly or indirectly, during a genetic regulatory process. Because a gene may regulate a group of other genes, this problem becomes identifying groups of such genes, i.e., bi-clusters. Housekeeping genes, which are constitutively expressed over most conditions, are not biologically or clinically interesting. The genes that the first two bi-cluster models, i.e., (a) and (b) find tend to be this kind. Therefore, most existing algorithms are based on either the additive model (d) or the multiplicative model (e) [6]. Since type (f) is a more general type that unifies types (c), (d), and (e), we focus on seeking type (f) bi-clusters in this paper.

Our algorithm, the Linear Coherent Bi-cluster Discovering (LCBD) algorithm, is based on first detecting linear correlations between pairs of gene expression profiles, then identifying groups by sample majority voting. To evaluate our algorithm, we will compare its performance to six existing, well known bi-clustering algorithms: Cheng and Church's algorithm, CC [4]; Samba [19]; Order Preserving Sub-matrix Algorithm, OPSM [1]; Iterative Signature Algorithm, ISA [10,9]; Bimax [15]; and Maximum Similarity Bi-clusters of Gene Expression Data, MSBE [12]. The first five algorithms were selected and implemented in the survey [15].

The last algorithm, MSBE, is the first polynomial time bi-clustering algorithm that finds optimal solutions, but under certain constraints. To briefly explain each of the first five bi-clustering algorithms: in [4] Cheng and Church defined a merit score, called mean squared residue, to evaluate the quality of a bi-clustering, and then develop a greedy algorithm for finding δ-bi-clusters. Yang *et al.* improved Cheng and Church's method by allowing missing values in gene expression matrices. Tanay *et al.* [19] and Prelić *et al.* [15] search for bi-clusters of up-regulated or down-regulated expression values, while the original expression matrices are discretized to binary matrices during a pre-processing phase. Ihmels *et al.* [10,9] used gene and condition signatures to evaluate bi-clusters, and propose a random iterative signature algorithm (ISA) when no prior information of the matrix is available. Ben-Dor *et al.*[1] attempt to find the order-preserving sub-matrix (OPSM) bi-clusters in which all genes have same linear ordering, based on a heuristic algorithm.

The remainder of the paper is organized as follows: First, we present the details of our LCBD method in Section 2. Then Section 3 describes the experimental evaluation of our proposed method, comparing its performance on both synthetic and real data to other algorithms. Section 4 then assesses the advantages and disadvantages of the LCBD algorithm and proposes some possible approaches that may overcome the drawbacks of the LCBD algorithm.

2 Methods and Algorithms

Let $A(I, J)$ be an $n \times m$ real valued matrix, where $I = \{1, 2, 3, ..., n\}$ is the set of genes and $J = \{1, 2, 3, ..., m\}$ is the set of samples. The element a_{ij} of $A(I, J)$ represents the expression level of gene i under sample j. A row vector $A(i, J)$ and a column vector $A(I, j)$ represents the ith gene over all the samples and the jth sample over all the genes, respectively. Our algorithm is composed of three major steps.

In the first step, for each pair of genes $A(p, J)$ and $A(q, J)$, where $p, q \in \{1, 2, 3, ...n\}$ and $p \neq q$, we construct a two-dimension binary matrix that represents the 2D image of the two vectors with x-coordinates $A(p, J)$ and y-coordinates $A(q, J)$, respectively. A pixel in the 2D image is denoted by a 1 in the binary matrix. Using the binary matrix as input, we then identify lines in the 2D image based on the Hough transform. The Hough transform technique works on the following principle: First note that each point (pixel) in a 2D image can be passed through by an infinite number of lines, and each of which can be parameterized by r and θ, where r is the perpendicular distance between the origin and the line and θ is the angle between the perpendicular line and the x-coordinate. Then note that the set of lines that pass through a point forms a sinusoidal curve in the $r - \theta$ coordinate space. Now, if there is a common line that passes through a set of points in the original 2D image, their corresponding sinusoidal curves must have a point of intersection in the $r-\theta$ space. So by finding a point of intersection in $r-\theta$ space, one can identify a line that passes

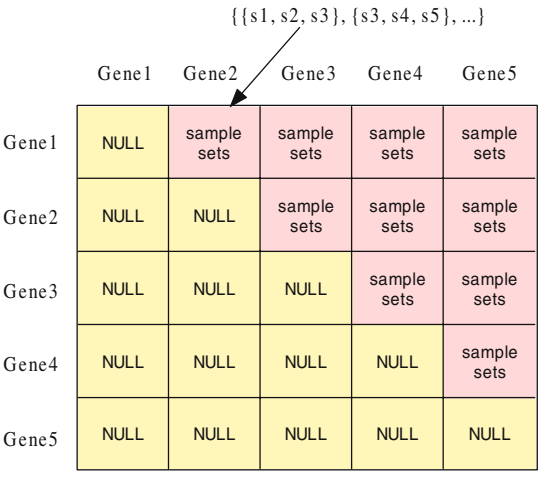


Fig. 3. Illustration of an $n \times n$ gene pairwise sample sets matrix

through a set of points in the original 2D space. Each line in the 2D image is a linear correlation between a pair of genes under a subset of samples. To allow for possible overlaps in the final bi-clusters, we let the Hough transform identify at most k non-reduplicative lines. Therefore, for each pair of genes, we can collect at most k sample sets over which the two genes are linearly correlated. After collecting sample sets for each gene pair, we obtain an $n \times n$ upper trianglular matrix, where each element contains at most k sample sets (See Figure 3 for an illustration). We denote each element in the matrix as S_{ij}. Note that for each 2D image, the horizontal lines and vertical lines in the 2D image are not eliminated since they might not represent linear correlations.

In the second step, for vector of sample sets, S_{iJ}, we count the samples that appear in each element of S_{iJ}. We then collect the top w voted samples into a sample pool and the corresponding genes who voted for these samples into a gene pool. The sample and gene pools thus constitute an initial bi-cluster. Then, for the remaining samples, we iteratively add them and their corresponding gene into the sample and gene pools, respectively, as long as by adding them the mean gene-gene correlation coefficient of the current bi-cluster remains above a threshold. The user specified parameter w should be greater than 3, because one can always draw a line between any 2 random points but the possibility that more than 2 random points lie on the same line is very small unless there is a linear correlation. In this step, each sample sets vector S_{iJ} will construct at most one bi-cluster that necessarily contains gene i.

In the third step, we remove redundancy in the bi-cluster sets generated in step two. If two bi-clusters share more than 60% identical elements, one of the two will be removed depending on which has more identical elements. Algorithm 1 describes the LCBD algorithm.

Algorithm 1. The LCBD Algorithm

Input: An $n \times m$ real value matrix $A(I, J)$, k, w.
Output: A set of bi-clusters $A(g_i, s_i)$, where $g_i \subseteq I$ and $s_i \subseteq J$.

for $i = 1$ to n **do**
  for $j = i + 1$ to n **do**
    Construct binary matrix $B_{i,j}$ for vectors $A(i, J)$ and $A(j, J)$;
    Do Hough transform based on $B_{i,j}$ and k to obtain a set of sample sets, S_{iJ};
  end for
end for

for $i = 1$ to n **do**
  Select the top w most voted samples in S_{iJ} as the initial sample pool s_i;
  Select the genes whose corresponding gene pair sample sets contain all the initial samples g_i;
  Construct the initial bi-cluster $A(g_i, s_i)$;
  while gene-wise mean correlation coefficient of $A(g_i, s_i) <$ threshold **do**
    Add the most voted sample in the leftover sample sets to the sample pool s_i;
    Add the corresponding gene into the gene pool g_i;
    Update bi-cluster $A(g_i, s_i)$;
  end while
end for

Remove redundant bi-clusters in the set $A(g_i, s_i)$ that has $> 60\%$ overlapping elements.
Output the set of bi-clusters $A(g_i, s_i)$;

3 Results

We tested our algorithm on both synthetic datasets and two real datasets *Saccharomyces cerevisiae* and *Arabidopsis thaliana*. For the synthetic datasets, we evaluate the algorithms based on how well they identify the real bi-clusters embedded in the expression matrix beforehand. We adopt the Prelić's match score function [15] as a quantified evaluation of merit: Let M_1, M_2 be two sets of bi-clusters. The gene match score of M_1 with respect to M_2 is given by the function

$$S_G^*(M1, M2) = \frac{1}{M_1} \sum_{(G_1,C_1)\in M_1} max_{(G_2,C_2)\in M_2} \frac{|G_1 \cap G_2|}{|G_1 \cup G2|}$$

$S_G^*(M1, M2)$ reflects the average of the maximum match scores for all bi-clusters in M_1 with respect to the bi-clusters in M_2. In our experiment, M_2 is one or more reference (optimal) bi-cluster(s) embedded in the expression matrix beforehand. For the parameter settings of the existing algorithms, we follow the previous works [12] and [15].

3.1 Results on Synthetic Data

Because most existing bi-clustering algorithms do not work on linear coherent bi-clusters, we select two bi-clustering algorithms OPSM and ISA that seek additive bi-cluster structures to compare to our LCBD algorithm, since an additive bi-cluster is a special case of a linear coherent bi-cluster. For the MSBE algorithm, we found in our testing that prior knowledge of a reference gene and reference sample for recovering a synthetic bi-cluster had a great effect on its final result, we therefore do not include the MSBE algorithm into the synthetic experiments because we assume that this prior knowledge is blind to all the algorithms tested.

Constant Bi-Cluster. To produce an expression matrix with an additive bi-cluster, we first randomly generated an 100×50 matrix. The values of the expression matrix obey either a normal distribution (with mean 0 ad SD 1) or a unique distribution (with minimum 0 and maximum 1), since a real data distribution could be either one of them [3,11,6]. Within the expression matrix, we randomly select a row and 10 columns to form a size 10 reference gene vector. We then randomly select 9 other row vectors under the same samples and re-calculate their expression values based on the equation $A(i, J_r) = m_i \times A(i_0, J_r) + b_i$, where $A(i_0, J_r)$ is the reference gene vector, m_i equals to 1, and b_i is a random constant. Random noise is then added to the synthetic bi-cluster: a certain percent of elements in the bi-cluster is randomly selected and replaced with random values which obey the same distribution as the background matrix. We tested noise levels of 0% to 25% with increasing steps of 5%. At each noise level we generated 50 synthetic matrices with bi-clusters and reported a final match score that is the mean over the 50 results. Figure 4 shows that our LCBD algorithm obtained the highest match scores for all noise levels and distributions, compared to the two additive bi-cluster type algorithms OPSM and ISA. As one can see, the LCBD algorithm is robust to noise even at noise level 25%. This occurs a line will be identified by the Hough transform as long as it passes through at least 3 points (samples) and during the majority sample voting. Although the expression value under some samples is destroyed, there are sufficiently many others that their expression values under these samples are not destroyed and thus these samples still obtain more votes than random samples that are not within the linear coherence bi-cluster.

Linear Coherent Bi-Cluster. Because the LCBD algorithm seeks bi-clusters of the linear coherent type, we can then test it directly on the linear coherent bi-clusters. In this experiment, we only use unique distribution expression matrix, since the normal distribution matrix shows similar results. To generate a linear coherent bi-cluster in a expression matrix, we use the same procedure as in the constant bi-cluster experiment, except that in the equation $A(i, J_r) = m_i \times A(i_0, J_r) + b_i$, the m_i's are no longer 1's but random values. The left part of Figure 5 shows the match scores of the LCBD algorithm under

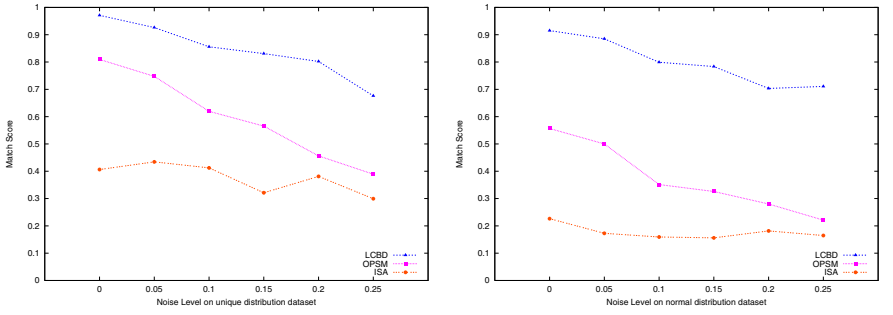


Fig. 4. Match scores of different additive model bi-clustering algorithms on synthetic dataset under unique distribution and normal distribution

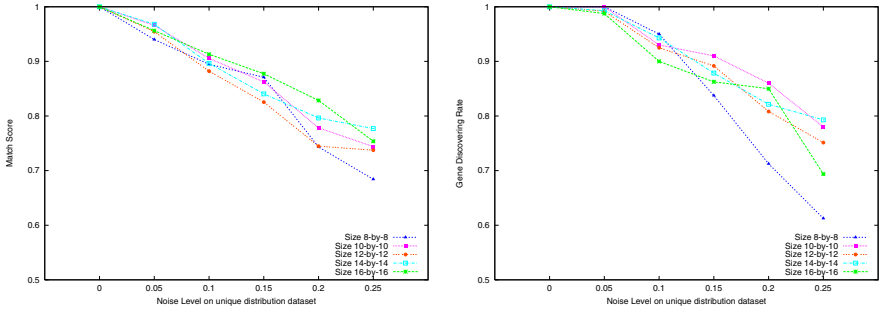


Fig. 5. Match score and gene discovering rate of the LCBD method on synthetic dataset of different bi-cluster size and noise level under unique distribution

different noise levels and different bi-cluster sizes; the right part of Figure 5 shows the corresponding gene discovery rates of the LCBD algorithm under the same noise level and bi-cluster size. From Figure 5, we can see that the match score and gene discovery rates are generally higher on larger bi-clusters. This is the case because whether a line can be identified during the Hough transform depends more on the absolute number of points that a line passes through than the proportion of points a line passes through. This suggests that the LCBD algorithm should be better at discovering large bi-clusters.

Overlapping Test. To test the LCBD algorithm on discovering multiple overlapping bi-clusters, we generated two linear coherent bi-clusters in the expression matrix and let them overlap to some degree. Figure 6 shows the mean match scores of the LCBD algorithm on discovering two overlapping bi-clusters at noise level 10%. For the overlapping elements, we replace their original values with the sum of the two overlapping values. The overlapping elements are not linear coherent elements and can be viewed as noise elements.

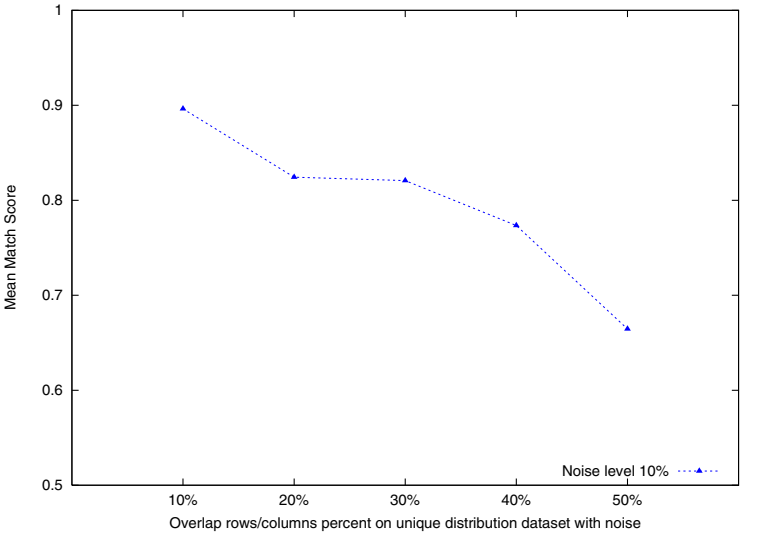


Fig. 6. Match score of the LCBD method of different bi-cluster overlapping rate on synthetic dataset under unique distribution

3.2 Results on Real Data

The documented descriptions of functions and processes that genes participate in has become widely available prior knowledge. The Gene Ontology Consortium in particular provides one of the largest organized collection of gene annotations. Following the idea in [19,15], we investigate whether the genes identified in bi-clusters produced by the different algorithms show significant enrichment with respect to a specific Gene Ontology annotation. We use two web-servers, FuncAssociate [2] and EasyGo [22], to evaluate the groups of genes produced in our bi-clustering results. The FuncAssociate computes the hypergeometric functional enrichment score, cf. [2], based on Molecular Function and Biological Process annotations. The resulting scores are adjusted for multiple testing by using the Westfall and Young procedure [21,2]. The EasyGo calculates the functional enrichment score in a similar way. In detail, based on availability, we tested the bi-clustering results from the *Saccharomyces Cerevisiae* dataset on the FuncAssociate web-server and the results from the *Arabidopsis thaliana* dataset on the EasyGo web-server. The *Saccharomyces Cerevisiae* dataset contains 2993 genes and 173 conditions and the *Arabidopsis thaliana* dataset contains 734 genes and 69 conditions. Figure 7 and Figure 8 show the proportion of gene groups of bi-clusters that are functionally enriched at different significance levels. The LCBD algorithm demonstrates the best results (all 100%) on the *Arabidopsis thaliana* dataset, compared to other seven algorithms; The LCBD results are also competitive to the best results derived from the MSBE algorithm on the *Saccharomyces Cerevisiae* dataset. These results on real datasets indicate that linear coherent bi-clusters are a useful form of bi-cluster structure to extract

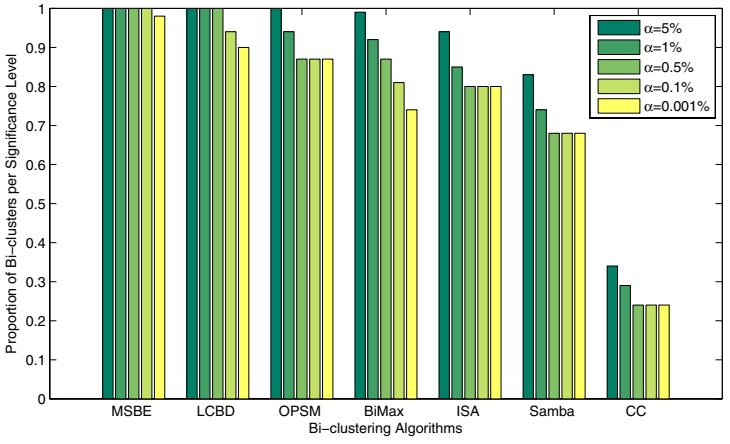


Fig. 7. Proportion of bi-clusters significantly enriched by any GO biological process category (S. Cerevisiae). α is the adjusted significant scores of the bi-clusters.

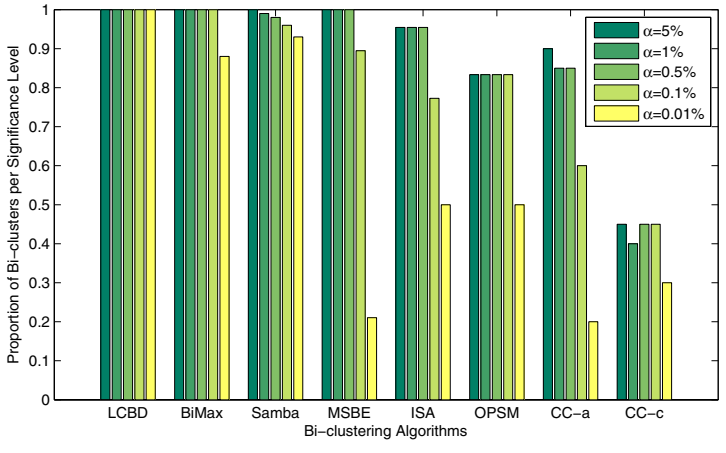


Fig. 8. Proportion of bi-clusters significantly enriched by any GO biological process category (A. thaliana). α is the adjusted significant scores of the bi-clusters.

from gene expression datasets, and could be a bi-cluster type that exists widely in other gene expression datasets.

4 Discussion and Conclusion

In this paper, we have developed a novel bi-clustering algorithm, the Linear Coherent Bi-cluster Discovering algorithm (LCBD), which seeks linear coherent bi-clusters in gene expression data. Our experimental results on the synthetic data

show that the LCBD algorithm can accurately discover additive and linear coherent bi-clusters, while being robust to the noise level and bi-cluster size. Our results on the two real datasets revealed that the linear coherent bi-clusters discovered by LCBD are functionally enriched and therefore biologically meaningful.

The drawback of using the traditional Hough transform technique for identifying linear correlations is that it can suffer from sparse data problems: even if some points lie perfectly on a common line, if the binary pixel matrix is too sparse, the traditional Hough transform might not find this line because different parameter values are needed to make the transform work appropriately for different sparsity levels. The sparse data problem may occur when the sample size of the expression matrix is small. However, the sparse data problem can be addressed by applying more advanced image analysis techniques such as the sparse resistant Hough transform or other feature recognition techniques.

The time complexity of the LCBD algorithm is worse than most algorithms mentioned in this paper, and improving its efficiency is an important direction for future work. It appears that selecting a set of representative genes, rather than all genes, to construct the $n \times n$ sample set matrix is a promising approach, since redundancies often occur if ones uses all genes. One intriguing aspect of the LCBD algorithm is that it can easily be extended to identify bi-clusters of more complex gene-gene correlations.

References

1. Ben-Dor, A., Chor, B., Karp, R., Yakhini, Z.: Discovering Local Structure in Gene Expression Data: The Order-Preserving Sub-Matrix Problem. In: Proc. of the 6th Annual International Conference on Computational Biology, pp. 49–57 (2002)
2. Berriz, G.F., King, O.D., Bryant, B., Sander, C., Roth, F.P.: Characterizing Gene Sets with FuncAssociate. BioInformatics 19, 2502–2504 (2003)
3. Causton, H.C., Quackenbush, J., Brazma, A.: Microarray Gene Expression Data Analysis: A Beginner's Guide. Blackwell Publishing, Malden (2003)
4. Cheng, Y., Church, G.M.: Biclustering of Expression Data. In: Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, pp. 93–103 (2000)
5. Eisen, M.B., Spellman, P.T., Brown, P.O., Botstein, D.: Cluster Analysis and Display of Genome-wide Expression Patterns. Proceedings of the National Academy of Sciences of the United States of America 95, 14863–14868 (1998)
6. Gan, X., Liew, A.W.-C., Yan, H.: Discovering Biclusters in Gene Expression Data based on High-dimensional Linear Geometries. BMC Bioinformatics 9, 209 (2008)
7. Hartigan, J.A.: Direct Clustering of a Data Matrix. Journal of the American Statistical Association 67, 123–129 (1972)
8. Hartigan, J.A., Wong, M.A.: A K-means Clustering Algorithm. Applied Statistics 28, 100–108 (1979)
9. Ihmels, J., Bergmann, S., Barkai, N.: Defining Transcription Modules Using Large Scale Gene Expression Data. Bioinformatics 20, 1993–2003 (2004)
10. Ihmels, J., Friedlander, G., Bergmann, S., Sarig, O., Ziv, Y., Barkai, N.: Revealing Modular Organization in the Yeast Transcriptional Network. Nature Genetics 31, 370–377 (2002)

11. Kluger, Y., Basri, R., Chang, J.T., Gerstein, M.: Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions. Genome Res. 13, 703–716 (2003)
12. Liu, X., Wang, L.: Computing the Maximum Similarity Bi-clusters of Gene Expression Data. Bioinformatics 23, 50–56 (2006)
13. Madeira, S.C., Oliveira, A.L.: Biclustering Algorithms for Biological Data Analysis: A Survey. Computational Biology and Bioinformatics 1, 24–45 (2004)
14. Mirkin, B.: Mathematical Classification and Clustering. Kluwer Academic Publishers, Dordrecht (1996)
15. Prelić, A., Bleuler, S., Zimmermann, P., Wille, A.: A Systematic Comparison and Evaluation of Biclustering Methods for Gene Expression Data. Bioinformatics 22(9), 1122–1129 (2006)
16. Sheng, Q., Moreau, Y., De Moor, B.: Biclustering Microarray Data by Gibbs Sampling. Bioinformatics 19, 196–205 (2003)
17. Sokal, R.R., Michener, C.D.: A Statistical Method for Evaluating Systematic Relationships. University of Kansas Science Bulletin 38, 1409–1438 (1958)
18. Tamayo, P., Slonim, D., Mesirov, J., Zhu, Q., Kitareewan, S., Dmitrovsky, E., Lander, E.S., Golub, T.R.: Interpreting Patterns of Gene Expression with Self-organizing Maps: Methods and Application to Hematopoietic Differentiation. Proceedings of the National Academy of Sciences of the United States of America 96, 2907–2912 (1999)
19. Tanay, A., Sharan, R., Shamir, R.: Discovering Statistically Significant Biclusters in Gene Expression Data. Bioinformatics 18, 136–144 (2002)
20. Tavazoie, S., Hughes, J.D., Campbell, M.J., Cho, R.J., Church, G.M.: Systematic Determination of Genetic Network Architecture. Nature Genetics 22, 281–285 (1999)
21. Westfall, P.H., Young, S.S.: Resampling-Based Multiple Testing. Wiley, New York (1993)
22. Zhou, X., Su, Z.: EasyGO: Gene Ontology-Based Annotation and Functional Enrichment Analysis Tool for Agronomical Species. BMC Genomics 8, 246 (2007)

Generalized Russian Cards Problem*

Zhenhua Duan and Chen Yang

Institute of Computing Theory and Technology, Xidian University
Xi'an 710071, China
{chyang,zhhduan}@mail.xidian.edu.cn

Abstract. This paper investigates Russian Cards problem for the purpose of unconditional secured communication. First, a picking rule and deleting rule as well as safe communication condition are given to deal with the problem with 3 players and 7 cards. Further, the problem is generalized to tackle n players and $n(n-1)+1$ cards. A new picking rule for constructing the announcement is presented, and a new deleting rule for players to determine each other's cards is formalized. In addition, the safe communication condition is also proved.

Keywords: Russian Cards Problem; picking rule; deleting rule.

1 Introduction

The security of cryptographic protocols generally depends upon several assumptions such as the agents are computationally limited and certain computational problems are intractable with these computational limits. In protocols based on public key encryption schemes such as RSA [11], for example, decryption of messages is tractable for the intended recipient but assumed to be impossible for an intruder, because it requires factoring a large product of primes, a problem assumed to be intractable [18]. There do exist, however, unconditionally secure protocols, whose security does not rely upon such assumptions. Some of such protocols have been studied recently in the cryptography and information theory community [6,9]. These protocols can be shown to be secure even against the adversaries with unlimited computational powers, because they ensure that the adversary cannot learn secrets for information theoretic rather than computational reasons.

'Russian Cards' problem was originally presented at the Moscow Mathematics Olympiad in 2000 with 3 players and 7 cards. The problem can be described as follows [14]:

From a pack of seven known cards two players each draw three cards and a third player gets the remaining card. How can the players with three cards openly (publicly) inform each other about their cards, without the third player learning from any of their cards who holds it?

* This research is supported by NSFC Grant No. 60433010 and 60873018, DPRP No.51315050105, SRFDP 200807010012, and SKLSE 20080713.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 85–97, 2009.

Typically, it is a communication protocol with three parties and seven numbers. The solution to the problem will imply a method to communicate information among parties in a distributed computing setting securely without using any encryption [6,10,8,1,15,12,16]. As a result, the communicating agents and adversaries can be modeled as players and the information to be communicated as the ownership of cards. It is generally believed that the above game gives unconditional security for communication protocols [14,6,10,8,17].

Concerning the initial Russian Cards problem, Ditmarsch [14] proposed a safe solution and gave 102 solutions according to different cards deals. However, he did not tell us how to figure out these solutions. Although some solutions can be found in the literature [14] for the problem, however, a generic safe communication protocol based on Russia Cards problem has not intensively been studied. Cyriac and Krishnan studied the Lower Bound for the Communication Complexity of the Russian Cards Problem [2]. They also pointed out that it is interesting to study the possible generalization of the problem to an n-player m-card game, and to derive a possible lower bound for announcement in this scenario. However, to the best of our knowledge, there is no published work on this generalization. Therefore, in this paper, we are motivated to investigate the Russian Cards problem with the following aspects: (1) formalizing the picking rule and deleting rule and safe communication condition with 3 players and 7 cards; (2) generalizing the problem to n players and $n(n-1)+1$ cards and further investigating the picking and deleting rules and safe communication condition.

In the paper, to deal with the initial Russian Cards problem, we first formalize two algorithms called picking rule and deleting rule to construct the announcement (a set of cards a announcer holds) and to manipulate the communication among players. Further, a safe communication condition is proposed and proved. Based on the mathematical analysis, an instance of Russian cards problem is given to show how the rules work. Further, we generalize the problem to n players and $n(n-1)+1$ cards so that a safe public communication protocol can be established. To do so, we randomly dispatch n cards from $n(n-1)+1$ cards to each party as his hand, and leave the remaining one card for intruder. In the communication with $n(n \geq 4)$ players, each party has to announce his hand by means of the announcement one by one. Each announcement is actually a matrix containing announcer's hand and other fake hands. Since we know the card deal and can construct the matrix for each party by means of a new picking rule. The i^{th} announcement is made after the $(i-1)^{th}$ announcement. The $(n-1)^{th}$ announcement gives the intruder's hand. After all of the announcements, all parties but intruder know each other's hand by means of a new deleting rule. We can guarantee the communication is safe since, after all $n-1$ announcements, all parties learn each other's hand, and the intruder knows nothing about any party's hand.

The rest of the paper is organized as follows. Section 2 models and analyzes the original Russia Cards problem. The picking rule and deleting rule as well as the safe communication condition are formalized. In section 3, we generalize the problem to a generic case with n players and $n(n-1)+1$ cards. By the

mathematical analysis, new picking and deleting rules are given. To prove the communication based on our approach is safe, some lemmas and theorems are proved in the appendix. Finally, the conclusion is drawn in Section 4.

2 Russian Cards Problem

2.1 Russian Cards Problem

The Russian cards problem was originally presented at the Moscow Mathematics Olympiad in 2000. Initially, the cards were named 0, ..., 6. Besides being public, all announcements are assumed to be truthful. For convenience, we use the following notations. According to game rules, when all cards are allocated to each player, we call the set of cards held by a player a **hand**, and the allocation of cards a **card deal**. Further, for a hand of cards such as {0, 1, 2}, we write 012 instead; and for a card deal such as 012, 345, 6, we write 012.345.6 to mean that the first player holds cards {0, 1, 2}, the second holds {3, 4, 5}, and the third holds {6}. Suppose the three players are Anne, Bill and Crow. We assume that 012.345.6 is the actual card deal. A solution to the Russian Cards problem is a sequence of secured announcements such that Anne and Bill know each other's hand without Crow learning any of cards from Anne and Bill. The following is an instance of solutions. [14]:

> *Anne says: "I have one of 012, 034, 056, 135, 246." After which Bill announces:"Crow has card 6."*

Once Bill receives Anne's announcement, he learns Anne's hand from the hand of his own. Accordingly, after having received both announcements, Crow knows just Anne has one of 012, 034, 135, but neither which one nor the hand of Bill. This sequence is safe. However, if we replace Anne's announcement by "I have one of 012, 034, 056, 134, 256" and keep other conditions, the result is different since although both Anne and Bill can learn each other's hand, Crow is also able to figure out some information (for instance, he can work out Bill holds card 5) about their cards because he holds card 6. So this sequence is unsafe.

2.2 What Is a Safe Communication?

We call the set of hands, such as 012,034,056,135 and 246 in the instance of solutions, appearing in any announcement a **hand set**. Observe the instance of solutions above, the set is generated in the way in which the first three hands cover all the cards with a sharing card (0 in the instance), and each of the remaining hands consists of three cards, coming from each of the first three hands excluding the sharing card. For convenience, we call this procedure **picking rule**.

Let $S = \{0, 1, 2, 3, 4, 5, 6\}$ denote the set of cards, and h_a, h_b and h_c represent the hand of Anne, Bill and Crow respectively. A matrix called hand matrix $B = (b_{i,j})_{3\times 3}$ is given below,

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}$$

denote a hand set, where each row represents a hand, and each of the last two columns also represents a hand. For simplicity, we assume Anne's hand is placed in the first row of B. Let R_i (resp. $[R_i]$) represent the i^{th} row (resp. set of cards from R_i), and C_j (resp. $[C_j]$) the j^{th} column (resp. set of cards from C_j) of matrix B. Formally, the picking rule is described in Algorithm 2.1.

Algorithm 2.1: (picking rule)

```
1  h := h_a                                              /* h is a copy of h_a */;
2  M := S − h
   /* extract a sharing card from h_a and fill the first column of B with it */
3  Let m ∈ h;
4  h := h − {m};
5  for i := 1 to 3 do
6      b_{i,1} := m;
7  end
8  Let m ∈ h; b_{1,2} := m; h := h − {m};
9  Let m ∈ h; b_{1,3} := m; h := h − {m};
10 for i := 2 to 3 do   /* place cards in other columns of the last two rows of B */
11     for j := 2 to 3 do
12         Let m ∈ M;
13         b_{i,j} := m; M := M − {m};
14     end
15 end
```

Algorithm 2.2: (deleting rule for Bill)

```
1  Let H_b := H;
2  for i := 1 to 3 do
3      if h_b ∩ [R_i] ≠ φ then
4          H_b := H_b − {[R_i]};
5      end
6  end
7  for j := 2 to 3 do
8      if h_b ∩ [C_j] ≠ φ then
9          H_b := H_b − {[C_j]};
10     end
11 end
```

Algorithm 2.3: (deleting rule for Crow)

```
1  Let H_c := H;
2  for i := 1 to 3 do
3      if h_c ∩ [R_i] ≠ φ then
4          H_c := H_c − {[R_i]};
5      end
6  end
7  for j := 2 to 3 do
8      if h_c ∩ [C_j] ≠ φ then
9          H_c := H_c − {[C_j]};
10     end
11 end
12 H_s := φ; M := H_c; h := |M|;
13 for k := 1 to h do                          /* obtaining all cards contained in M */
14     Let m ∈ M;
15     H_s := H_s ∪ m; M := M − {m};
16 end
```

The sequence of announcements, the first from Anne and the second from Bill, completes the communication procedure for the Russian Cards problem. So we call this sequence a **communication**. Note that, in a communication, once receiving the announcement from Anne, Bill and Crow can remove the hands containing cards in their own hands, from the hand set within the announcement. We call this procedure **deleting rule**. Let $H_a = \{[R_1], [R_2], [R_3], [C_2], [C_3]\}$ denote the set of all possible hands for Anne. The rule can be formalized as follows. The deleting rule for Bill is formally described in Algorithm 2.2.

Note that if

$$|H| = 1 \tag{2.1}$$

then Bill learns Anna's hand. One may think the above deleting rule can be used for Crow to determine Anne's hand as well. However, it is not enough to

guarantee a safe communication since Crow might learn what cards Anne has not held from his announcement. For instance, suppose Anne's announcement is $\{012, 034, 056, 134, 256\}$ and Bill still hold 345, by Algorithm 2.2, Bill learns Anne's hand is 012. Since both hands 056 and 256 contain card 6, by removing them from the hand set , Crow also learns Anne's hand is one of $\{012, 034, 134\}$. Although Crow cannot determine Anne's hand however he learns Anne does not hold card 5. This is not a safe communication. Therefore, to guarantee a safe communication, the set of hands generated by using deleting rule must cover all cards except for Crow's card. Formally, the deleting rule for Crow is given in Algorithm 2.3.

Where H_s represents the set of cards extracted from members of H (Crow figured out set of all possible hands for Anne). It is easy to see that, if

$$h_c = S - H_s \tag{2.2}$$

then Crow learns nothing about hands of Anne and Bill. Thus, after Anne's announcement, Bill and Crow complete the communication by the deleting rule, Bill learns Anne's hand, while Crow knows neither Anne's nor Bill's hand. Also, Crow does not know what cards Anne and Bill do not hold. Therefore, the communication is safe. In what follows, we call equations 2.1 and 2.2 the safe conditions for the communication. Therefore, we have the following conclusion.

Theorem 1. *For the Russian Cards problem, a communication based on the hand matrix constructed by the picking rule is safe.*

We have proved Theorem 1.

Back to the instance of solutions, Anne's hand 012 is one of the first three hands. So, Bill's hand contains three cards out of 3, 4, 5 and 6. There are $C_4^3 = 4$ possible hands 345, 346, 356 or 456 for Bill. Suppose his hand is 345. According to the deleting rule, Bill can remove four hands 034, 135, 246 and 056 from the hand set, and the left hand 012 should be Anne's hand. The intruder Crow holds card 6 and can remove hands 056 and 246 from the hand set, but the left hands 012, 034, 135 cover all the cards excluding card 6, so he cannot decide which cards Anne and Bill hold or do not hold. Accordingly, the communication in this instance of solutions is safe.

3 Generalization of Russian Cards Problem

The Russian cards problem with 3 players and 7 cards, written as $R_{(3)}$, can be generalized to n players and $n(n-1)+1$ cards, $\{0, 1, ..., n^2 - n\}$, written as $R_{(n)}$. The picking rule and deleting rule for $R_{(3)}$ can also be generalized for $R_{(n)}$. For convenience, we use the following notations. We call each player but the intruder a **party**, represented by P_i $(1 \leq i \leq n-1)$, and the party who announces his hand **announcer**. We randomly dispatch n cards from $n(n-1)+1$ cards to each party as his hand, and leave the remaining one card for intruder as his hand. In the communication protocol the information communicated among

parties actually are their hands. We use h_{P_i} and h_{in} to denote hands of party P_i and the intruder respectively. In the communication with $n(n \geq 4)$ players, each party has to announce his hand by means of the announcement one by one. With our approach, each announcement is actually a matrix containing announcer's hand and other fake hands. Since we know the card deal we can generate the matrix for each party. The i^{th} announcement is made after the $(i-1)^{th}$ announcement; and the $(n-1)^{th}$ announcement shows the intruder's hand. After all of $n-1$ parties have made their announcements, they know each other's hand. The communication is safe if, after all $n-1$ announcements, (i) all parties learn each other's hand; and (ii) the intruder knows nothing about any party's hand.

In the communication based on $R_{(n)}$, the picking rule is responsible for generating matrix and hiding hand of the announcer in it. As a matter of fact, announcer's hand can be placed in any rows or columns of the matrix or hidden with any way in the matrix. However, for simplicity, we assume the announcer's hand is placed in the first row. In the communication protocol the picking rule is actually to encrypt the announcer's hand by means of hiding it in the matrix. In the following, we discuss the picking rule for $R_{(n)}$.

1. Picking rule

Each announcer announces his hand by means of a matrix so that his hand can be hidden in it. So, we need construct a matrix for each announcer. For convenience, we use $B^k = (b^k_{i,j})_{n\times n}$ $(1 \leq k \leq n-2)$ to denote the matrix for announcer P_k in k^{th} announcement. The structure of B^k is as follows: the first row we call **hand row** holds all n cards of h_{P_k}. The first column which we call **sharing column** is filled with a card from h_{P_k} which we call **sharing card**. As shown in Fig.1, the announcer's hand is $h_{P_k} = \{b^k_{1,1}, b^k_{1,2}, ..., b^k_{1,n}\}$, and the sharing card is $b^k_{1,1}$, where $b^k_{1,1} = b^k_{2,1} = \cdots = b^k_{n,1}$. The intruder's card can be placed in any of the remaining places. We assume that the intruder holds card $b^k_{p,q}$. We call the row and column containing this card respectively **redundant row** and **redundant column**. The remaining part of B^k holds the other $n^2 - 2n + 1$ cards. However, how to place these cards into the matrix is a tricky job (we introduce the method later). In principle, to guarantee a successful matrix for the communication based on $R_{(n)}$, the above matrix has to satisfy the property (called covering property): each row and each column, apart from the hand row and sharing column, contains a card from each hand of all parties except for the announcer. In this way, any row of B^k is a possible hand of the announcer, and apart from the sharing column, any column is also a possible hand of announcer. Let R^k_i (resp. $[R^k_i]$) represent the i^{th} row (resp. set of cards from R^k_i), and C^k_j (resp. $[C^k_j]$) the j^{th} column (resp. set of cards from C^k_j) of matrix B^k. Thus, the covering property can be given as follows,

$$\forall i\, \forall j\, [(1 \leq i \leq n-1 \wedge i \neq k \wedge 2 \leq j \leq n) \rightarrow h_{P_i} \cap [R^k_j] \neq \phi \wedge h_{P_i} \cap [C^k_j] \neq \phi] \quad (3.1)$$

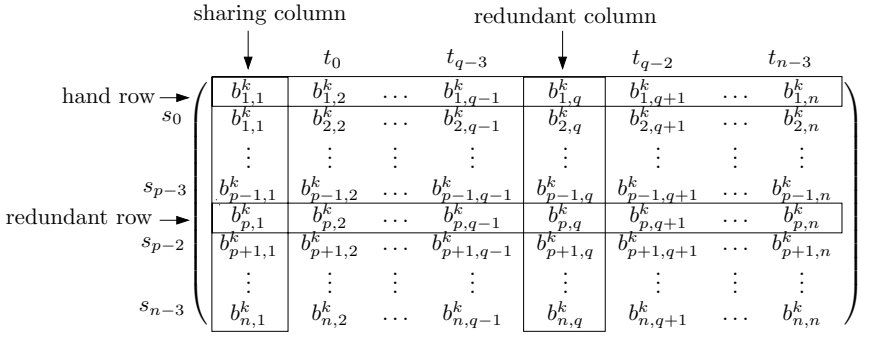


Fig. 1. Hand matrix of P_k

We call the procedure generating matrix B^k for P_k **picking rule**. Actually, we first generate B^1 for P_1, then we construct B^h for P_h $(2 \le h \le n-2)$ based on B^1. For clarity, the pick rule can be formalized as follows:

A. Constructing B^1

(a) Firstly, we chose a card from h_{P_1} as the *sharing card* to fill the *sharing column*. Then we place the remaining cards of h_{P_1} in the *hand row* in any order. Secondly we randomly place intruder's card in one of the remaining places of B^1. Suppose intruder's card is $b^1_{p,q}$, so p^{th} row and q^{th} column are respectively *redundant row* and *redundant column*. As shown in Fig.1, apart from the sharing column, redundant column, hand row and redundant row, indices of the remaining rows from top to bottom are respectively denoted by s_0, s_1, ..., s_{n-3}, and indices of the remaining columns from left to right are respectively denoted by t_0, t_1, ..., t_{n-3}. For convenience, let $S := \{s_0, ..., s_{n-3}\}$ and $T := \{t_0, ..., t_{n-3}\}$. Thus, the indices can be generated in Algorithm 3.1.

(b) We randomly divide each hand of the remaining parties P_k $(2 \le k \le n-1)$ into three parts such that $h_{P_k} = X_k \cup Y_k \cup Z_k$, $|X_k| = n-2$ and $|Y_k|=|Z_k|=1$. Formally, it is described in Algorithm 3.2

For convenience, we further define positive integers od_k and ed_k $(2 \le k \le n-1)$ as follows:

$$od_k = \begin{cases} 2k-4, & 2 \le k \le \frac{n+1}{2}; \\ 2k-n-2, & \frac{n+1}{2}+1 \le k \le n-1, \end{cases}$$

and

$$ed_k = \begin{cases} 0, & k = 2; \\ n-k, & 3 \le k \le \frac{n}{2}; \\ n-1-k, & \frac{n+2}{2} \le k \le n-2; \\ \frac{n-2}{2}, & k = n-1. \end{cases}$$

(c) We place all $n-2$ cards of X_2 into the remaining places of B^1 so that any s^{th} row, $s \in S$, and any t^{th} column, $t \in T$, can be occupied, namely $[R^1_s] \cap X_2 \neq \phi$

Algorithm 3.1: (step (a) of generating B^1)

```
1  for i := 2 to n do
2      if i < p then
3          s_{i-2} := i;
4      else if i=p then                                   /* skip the p^th row */
5          skip;
6      else
7          s_{i-3} := i;
8      end
9      if i < q then
10         t_{i-2} := i;
11     else if i=q then                                   /* skip the q^th column */
12         skip;
13     else
14         t_{i-3} := i;
15     end
16 end
```

Algorithm 3.2: (step (b) of generating B^1)

```
1  for k := 2 to n − 1 do
2      M := h_{P_k};                                      /* M is a copy of h_{P_k} */
3      for i := 1 to n − 2 do      /* pick n − 2 cards from M and put them into X_k */
4          Let c ∈ M;
5          M := M − {c}; X_k := X_k ∪ {c};
6      end
       /* put the remaining two cards from M into Y_k and Z_k respectively */
7      Let c ∈ M; M := M − {c}; Y_k := {c}; Z_k := M;
8  end
```

Algorithm 3.3: (step (c) of generating B^1)

```
1  Let H := {0, 1, ··· , n − 3};
2  for i := 0 to n − 3 do                                 /* place all cards of X_2 in B^1 */
3      Let c ∈ X_2; X_2 := X_2 − {c};                     /* pick a card out of X_2 */
4      Let d ∈ H; H := H − {d};
5      b^1_{s_d, t_i} := c;                /* card c is placed in s_d^th row and t_i^th column */
6      q_i := d;                          /* q_i keeps the index of row card c appears in */
7  end
   /* for every other party P_k, all cards of X_k are placed in B^1 */
8  for i := 0 to n − 3 do
9      for k := 3 to n − 1 do
10         Let c ∈ X_k; X_k := X_k − {c};
11         if n is odd then        /* card c chosen from X_k is placed in t_i^th column */
12             d := (q_i + od_k) mod (n − 2);
13         else
14             d := (q_i + ed_k) mod (n − 2);
15         end
16         b^1_{s_d, t_i} := c;            /* card c is placed in s_d^th row and t_i^th column */
17     end
18 end
```

Algorithm 3.4: (step (d) of generating B^1)

```
1  Let H := {0, 1, ··· , n − 3}; Let c ∈ Y_2; Let i ∈ H;
2  b^1_{p, t_i} := c;              /* in redundant row, card of Y_2 is placed in t_i^th column */
   /* for every other party P_k, card of Y_k is placed into redundant row */
3  for k := 3 to n − 1 do
4      Let c ∈ Y_k;
5      if n is odd then                    /* card c of Y_k is placed in redundant row */
6          d := (i + od_k) mod (n − 2);
7      else
8          d := (i + ed_k) mod (n − 2);
9      end
10     b^1_{p, t_d} := c;          /* in redundant row, card c is placed in t_d^th column */
11 end
12 Let H := {0, 1, ··· , n − 3}; Let c ∈ Z_2; Let i ∈ H;
13 b^1_{s_i, q} := c;              /* in redundant column, card of Z_2 is placed in s_i^th row */
   /* for every other parity P_k, card of Z_k is placed into redundant column */
14 for k := 3 to n − 1 do
15     Let c ∈ Z_k;
16     if n is odd then                    /* card c of Z_k is placed in redundant column */
17         d := (i + od_k) mod (n − 2);
18     else
19         d := (i + ed_k) mod (n − 2);
20     end
21     b^1_{s_d, q} := c;          /* in redundant column, card c is placed in s_d^th row */
22 end
```

and $[C_t^1] \cap X_2 \neq \phi$. Thus, in any t^{th} column there exists a card from X_2. Suppose the card is in ${s_i}^{th}$ row. For each of the remaining parties P_k $(3 \leq k \leq n-1)$ we randomly pick a card from X_k and place it in ${s_u}^{th}$ $(u = (i + od_k)\ mod\ (n-2))$ row if n is odd, or place it in ${s_v}^{th}$ $(v = (i + ed_k)\ mod\ (n-2))$ row if n is even. Formally, it is described in Algorithm 3.3.

(d) Since there exists only one card in Y_2 (resp. Z_2) we place it randomly in any column (resp. row) within the redundant row (resp. column). We assume the card is in ${t_i}^{th}$ column (resp. ${s_i}^{th}$ row), $t_i \in T$ (resp. $s_i \in S$) in the redundant row (resp. column). For each of the remaining parties P_k $(3 \leq k \leq n-1)$, if n is odd we place the only one card from Y_k (resp. Z_k) into ${t_u}^{th}$ $(u = (i+od_k)\ mod\ (n-2))$ column (resp. ${s_u}^{th}$ row) in the redundant row (resp. column), if n is even we place the only one card from Y_k (resp. Z_k) into ${t_v}^{th}$ $(v = (i + ed_k)\ mod\ (n-2))$ column (resp. ${s_v}^{th}$ row) in the redundant row (resp. column). Formally, it is described in Algorithm 3.4.

B. Constructing B^k, $2 \leq k \leq n-2$.

For convenience, we need three auxiliary matrices $A^k = (a_{i,j}^k)_{n\times n}$, $D^k = (d_{i,j}^k)_{n\times n}$ and $F^k = (f_{i,j}^k)_{n\times n}$. In the same way as B^k, indices of rows and columns of matrices A^k, D^k and F^k can respectively be represented by s_0, s_1, ..., s_{n-3} and t_0, t_1, ..., t_{n-3}. Let AR_i^k, DR_i^k and FR_i^k (resp. AC_i^k, DC_i^k and FC_i^k) be i^{th} rows (resp. columns) of matrices A^k, D^k and F^k respectively. Let $[DR_i^k]$ represent the set of cards from DR_i^k, and $[DC_j^k]$ the set of cards from DC_j^k. The procedure for constructing B^k is as follows

(i) constructing A^k from B^1 (see Algorithm 3.5)

We first copy the first row and p^{th} row of B^1 to the first row and p^{th} row of A^k respectively. Then, for the remaining rows, we copy ${s_i}^{th}$ row of B^1 to ${s_j}^{th}$ row of A^k, $j := (i - (k-1) + n - 2)\ mod\ (n-2)$.

(ii) constructing D^k from A^k (see Algorithm 3.6) We first copy the first column and q^{th} column of A^k to the first column and q^{th} column of D^k respectively. Then, for the remaining columns, we copy ${t_i}^{th}$ column of A^k to ${t_j}^{th}$ column of D^k, $j := (i - (k-1) + n - 2)\ mod\ (n-2)$.

(iii) constructing F^k and B^k (see Algorithm 3.7). We first swap cards of h_{P_1} with cards of h_{P_k} in F^k. Then, let $B^k := F^k$.

After each announcement B^h $(1 \leq h \leq n-2)$, apart from announcer P_h, any other party P_k $(1 \leq k \leq n-1, k \neq h)$ compares his hand with the announcement to determine the announcer's hand. Similarly, the intruder also compares his hand with the announcement to probe announcer's hand. In the communication protocol, a **deleting rule** for parties is required to decrypt the announcer's hand from matrix B^h, and might be used for the intruder to detect the announcer's hand.

2. Deleting rule

Suppose the current announcer is P_h $(1 \leq h \leq n-2)$. Let H_{P_k} be a set of hands not intersecting with h_{P_k} $(k \neq h)$. The deleting rule for P_k can be formally described in Algorithm 3.8.

Algorithm 3.5: (constructing A^k from B^1)

1 $AR_1^k := R_1^1$; /* copy the first row of B^1 to the first row of A^k */
2 $AR_p^k := R_p^1$; /* copy p^{th} row of B^1 to p^{th} row of A^k */
3 **for** $i := 0$ **to** $n-3$ **do** /* construct the remaining rows of A^k */
4 $j := (i-(k-1)+n-2)\ mod\ (n-2)$;
5 $AR_{s_j}^k := R_{s_i}^1$; /* copy $s_i{}^{th}$ row of B^1 to $s_{i-k+1}{}^{th}$ row of A^k */
6 **end**

Algorithm 3.6: (constructing D^k from A^k)

1 $DC_1^k := AC_1^k$; /* copy the first column of A^k to the first column of D^k */
2 $DC_q^k := AC_q^k$; /* copy q^{th} column of A^k to q^{th} column of D^k */
3 **for** $i := 0$ **to** $n-3$ **do** /* construct the remaining columns of D^k */
4 $j := (i-(k-1)+n-2)\ mod\ (n-2)$;
5 $DC_{t_j}^k := AC_{t_i}^1$; /* copy $t_i{}^{th}$ column of A^k to $t_{i-k+1}{}^{th}$ column of D^k */
6 **end**

Algorithm 3.7: (constructing F^k and B^k)

1 $F^k := D^k$; /* copy D^k to F^k */
2 **for** $j := 0$ **to** $n-3$ **do** /* locate the card of h_{P_k} in redundant row */
3 **if** $f_{p,j}^k \in Y_k$ **then**
4 break;
5 **end**
6 **end**
7 $g := f_{p,j}^k$; $f_{p,j}^k := f_{1,1}^k$;
8 **for** $i := 1$ **to** n **do** /* fill the sharing column with card g */
9 $f_{i,1}^k := g$;
10 **end**
/* swap the card of P_1 with card of P_k in every other column */
11 **for** $j := 2$ **to** n **do**
12 **for** $i := 0$ **to** $n-3$ **do** /* locate the card of h_{P_k} in j^{th} column */
13 **if** $f_{s_i,j}^k \in h_{P_k}$ **then** /* card of P_k is in j^{th} column */
14 break;
15 **end**
16 **end**
17 $g := f_{s_i,j}^k$; $f_{s_i,j}^k := f_{1,j}^k$; $f_{1,j}^k := g$; /* swap card $f_{s_i,j}^k$ with card $f_{1,j}^k$ */
18 **end**
19 $B^k := F^k$; /* copy F^k to B^k */

Algorithm 3.8: (deleting rule for P_k)

1 $H_{P_k} := \phi$;
2 **for** $i := 1$ **to** n **do**
3 **if** $h_{P_k} \cap [R_i^h] = \phi$ **then** /* $[R_i^h]$ is possibly the hand of P_h */
4 $H_{P_k} := H_{P_k} \cup \{[R_i^h]\}$;
5 **end**
/* if $h_{P_k} \cap [R_i^h] \neq \phi$, we ignore it */
6 **end**
7 **for** $j := 2$ **to** n **do**
8 **if** $h_{P_k} \cap [C_j^h] = \phi$ **then** /* $[C_i^h]$ is possibly the hand of P_h */
9 $H_{P_k} := H_{P_k} \cup \{[C_j^h]\}$;
10 **end**
/* if $h_{P_k} \cap [C_i^h] \neq \phi$, we ignore it */
11 **end**

Algorithm 3.9: (deleting rule for intruder)

1 $H_{in} := \phi$;
2 **for** $i := 1$ **to** n **do**
3 **if** $h_{in} \cap [R_i^h] = \phi$ **then**
4 $H_{in} := H_{in} \cup \{[R_i^h]\}$;
5 **end**
6 **end**
7 **for** $j := 2$ **to** n **do**
8 **if** $h_{in} \cap [C_j^h] = \phi$ **then**
9 $H_{in} := H_{in} \cup \{[C_j^h]\}$;
10 **end**
11 **end**
12 $H_{intr} := \phi$; $m := |H_{in}|$;
/* this is to generate the set of cards from all hands in H_{in} */
13 **for** $k := 1$ *to* m **do**
14 **Let** $g \in H_{in}$;
15 $H_{intr} := H_{intr} \cup g$; $H_{in} := H_{in} - \{g\}$;
16 **end**

Party P_k compares his hand with each row and each column (excluding the sharing column) of B^h, and records them into set H_{P_k} if the row or column does not intersect with his hand since, in this case, it is a possible hand of announcer P_h. Note that if

$$|H_{P_k}| = 1 \tag{3.2}$$

then P_k is aware of announcer's hand. Let H_{in} be a set of hands not intersecting with h_{in}, and H_{intr} be the set of cards from hands of H_{in}. The deleting rule for intruder can formally be described in Algorithm 3.9.

Similarly to a party, the intruder compares his hand with each row and each column (excluding the sharing column) of B^h and records them into set H_{in} if the row or column is a possible hand of announcer P_h. After that, he extracts all cards from hands in H_{in} and puts them into set H_{intr}. Subsequently, he checks if

$$h_{in} = \{0, 1, \ldots, n^2 - n\} - H_{intr} \tag{3.3}$$

If so, the intruder learns nothing about announcer's hand.

As a matter of fact, $R_{(3)}$ is a trivial problem of $R_{(n)}$. If we make the hand set into 3×3 matrix for Anne to announce, no matter what hand Bill holds, the matrix satisfies expression 3.1.

In the following we prove that matrix B^h generated according to the *picking rule* satisfies expression 3.1.

Lemma 1. *Matrix B^h $(1 \leq h \leq n-2)$ generated according to the picking rule satisfies expression 3.1.*

We have proved Lemma 1.

After h^{th} announcement, by the deleting rule, the intruder can only remove the redundant row and redundant column from B^h. Let

$$HS^h = \{[R_1^h], [R_{s_0}^h], \ldots, [R_{s_{n-3}}^h], [C_{t_0}^h], \ldots, [C_{t_{n-3}}^h]\}.$$

From the intruder's view, HS^h contains all candidates of hand of P_h.

After $(n-2)^{th}$ announcement, the intruder can obtain $n-2$ hand sets: $HS^k (1 \leq k \leq n-2)$. Since any two parties do not allow to share a card, the intruder may obtain a group of hand sets,

$$G = \{\{h_1, ..., h_{n-2}\} \in HS^1 \times, ..., HS^{n-2} | \ \forall i, j \ h_i \cap h_j = \phi, i \neq j, \ 1 \leq i, j \leq n-2\}.$$

Any hand set from G can be regarded as a candidate of card deal. Note that if $|G| = 1$ the intruder knows the card deal, and further learns the hand of any party. So, the communication is unsafe. Before we prove the communication with B^h $(1 \leq h \leq n-2)$ constructed according to the picking rule is safe, we need the following lemmas.

Lemma 2. $\forall s \, \forall t \, [(2 \leq s, t \leq n-2 \wedge s \neq t) \rightarrow (od_s - s + n - 1) \ mod \ (n-2) \neq (od_t - t + n - 1) \ mod \ (n-2) \wedge (ed_s - s + n - 1) \ mod \ (n-2) \neq (ed_t - t + n - 1) \ mod \ (n-2)]$.

We have proved Lemma 2.

Lemma 3. $\forall h\ \forall k\ \forall i\ [(1 \leq h, k \leq n-2 \wedge h \neq k \wedge 0 \leq i \leq n-3) \rightarrow [R^h_{s_i}] \cap [R^k_{s_i}] = \phi \wedge [C^h_{t_i}] \cap [C^k_{t_i}] = \phi]$.

We have exploited Lemma 2 to prove Lemma 3.

Theorem 2. *(Safe communication for $R_{(n)}$) Given a card deal for $R_{(n)}$, the communication based on B^k $(1 \leq k \leq n-2)$ generated by the picking rule is safe.*

We have exploited Lemma 1 and Lemma 3 to prove Theorem 2.

4 Conclusion

This paper investigated Russia cards problem. First, $R_{(3)}$ was studied and some solutions were given in detail. Then, the problem was generated to $R_{(n)}$, and the picking rule was developed to construct hand set while the deleting rule was designed to decide card deal. Based on $R_{(n)}$, an unconditionally secured protocol can further be developed to tackle n parties communication without public keys. This is a challenge to us in the future research. In addition, the verification of the protocol based on $R_{(n)}$ is also required. To do so, we will employ Propositional Projection Temporal Logic [3,4,5,13] to express the properties and use Promela [7] to describe the behavior of the protocol. Thus, the model checker SPIN [7] can be used to check the properties.

References

1. Albert, M.H., Aldred, R.E.L., Atkinson, M.D., van Ditmarsch, H.P., Handley, C.C.: Safe Communication for Card Players by Combinatorial Designs for Two-Step Protocols. Australasian Journal of Combinatorics 33, 33–46 (2005)
2. Cyriac, A., Krishnan, K.M.: Lower Bound for the Communication Complexity of the Russian Cards Problem. CoRR, vol. abs/0805.1974 (2008), www.arxiv.org
3. Duan, Z., Koutny, M.: A Framed Temporal Logic Programming Language. J. Comput. Sci. Technol. 19, 333–344 (2004)
4. Duan, Z., Tian, C., Li, Z.: A Decision Procedure for Propositional Projection Temporal Logic with Infinite Models. Acta Informatica 45(1), 43–78 (2008)
5. Duan, Z., Yang, X., Koutny, M.: Frammed Temporal Logic Programming. Science of Computer Programming 70(1), 31–61 (2008)
6. Fischer, M.J., Wright, R.N.: Bounds on Secret Key Exchange using a Random Deal of Cards. Journal of Cryptology 9(2), 71–99 (1996)
7. Holzmann, G.J.: The Spin Model Checker: Primer and Reference Manual. Addison-Wesley, Reading (2003)
8. Koichi, K., Takaaki, M., Takao, N.: Necessary and Sufficient Numbers of Cards for the Transformation Protocol. In: Chwa, K.-Y., Munro, J.I.J. (eds.) COCOON 2004. LNCS, vol. 3106, pp. 92–101. Springer, Heidelberg (2004)
9. Makarychev, K.: Logicheskie Voprosy Peredachi Informacii (Logical Issues of Information Transmission). Master's Thesis, Moscow State University, Diplomnaja rabota, part 1 (2001)
10. Ramanujam, R., Suresh, S.P.: Information based Reasoning about Security Protocols. Electronic Notes in Theoretical Computer Science 55(1), 89–104 (2001)

11. Rivest, R., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
12. Roehling, S.: Cards and Cryptography. Report in Partial Fulfilment of MSc in Computer Science. University of Otago (2005)
13. Tian, C., Duan, Z.: Model Checking Propositional Projection Temporal Logic Based on SPIN. In: Butler, M., Hinchey, M.G., Larrondo-Petrie, M.M. (eds.) ICFEM 2007. LNCS, vol. 4789, pp. 246–265. Springer, Heidelberg (2007)
14. van Ditmarsch, H.P.: The Russian Cards Problem. Studia Logica 75, 31–62 (2003)
15. van Ditmarsch, H.P.: The Case of the Hidden Hand. Journal of Applied Non-Classical Logics 15(4), 437–452 (2005)
16. van Ditmarsch, H.P., van der Hoek, W., Kooi, B.P.: Public Announcements and Belief Expansion. Advances in Modal Logic 5, 335–346 (2005)
17. van Ditmarsch, H.P., van der Hoek, W., van der Meyden, R., Ruan, J.: Model Checking Russian Cards. Electronic Notes in Theoretical Computer Science 149(2), 105–123 (2006)
18. Vasilenko, O.: Number-Theoretic Algorithms in Cryptography (Translations of Mathematical Monographs). American Mathematical Society, Boston (2006)

Computing the Transitive Closure of a Union of Affine Integer Tuple Relations

Anna Beletska[1], Denis Barthou[2], Wlodzimierz Bielecki[3], and Albert Cohen[4]

[1] INRIA Saclay, France
Anna.Beletska@inria.fr
[2] University of Versailles St. Quentin, France
Denis.Barthou@prism.uvsq.fr
[3] Technical University of Szczecin, Poland
WBielecki@wi.ps.pl
[4] INRIA Saclay, France
Albert.Cohen@inria.fr

Abstract. This paper proposes a method to compute the transitive closure of a union of affine relations on integer tuples. Within Presburger arithmetics, complete algorithms to compute the transitive closure exist for convex polyhedra only. In presence of non-convex relations, there exist little but special cases and incomplete heuristics. We introduce a novel sufficient and necessary condition defining a class of relations for which an exact computation is possible. Our method is immediately applicable to a wide area of symbolic computation problems. It is illustrated on representative examples and compared with state-of-the-art approaches.

Keywords: transitive closure, affine tuple relations.

1 Introduction

Computing the transitive closure of graphs is required in many algorithms, from CAD, software engineering, real-time process control, data bases to optimizing compilers. To cite a few of its applications in the domain of programming languages and compilation: redundant synchronization removal, testing the legality of iteration reordering transformations, computing closed form expressions for induction variables, iteration space slicing and code generation [3,9], model checking and in particular reachability analysis (see [7] and included references), testing equivalence between codes [1,2,10].

Graphs can be represented in different ways. One of possible representations of graphs is based on tuple relations. In this paper, we consider the class of parameterized and affine integer tuple relations whose constraints consist of affine equalities and inequalities. Such relations describe infinite graphs. There are many techniques for computing transitive closures for finite graphs, but to our best knowledge, techniques for computing the transitive closure of a parameterized affine integer tuple relation, that describes infinite graphs, were the subject of the investigation of a few papers only [5,6,7,9]. The solution presented by Kelly

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 98–109, 2009.

et al. [9] is based on heuristics that guarantee neither calculating the exact result nor its conservative approximation.

This paper presents a method to calculate the transitive closure of a relation R being a union of n integer tuple relations R_i, i=1,2,...,n. The domain and range of an integer tuple relation consists of integer tuples. Our approach is based on computing relation A[R^k], (symbolically) for all k, being either the exact R^k or its overapproaximation. We assume that power k of each individual relation R_i can be computed.

$R^k = \underbrace{R \circ R \circ ... \circ R}_{k\ times}$ represents the power k of relation R.

Computation of R^k itself is an important result. It opens the door for extracting fine-grained parallelism available in program loops, for example, by building the free-scheduling of loop statement instances [8] implying that each loop statement instance is executed as soon as its operands are ready. When R^k does not describe redundant synchronization [9], then to get the free-scheduling we form the set range(R^k), use the constraints of R^k to calculate the upper bound of k, k_{max}, defining the latency of the free-scheduling [8], and generate a nested loop whose outermost nest enumerates values of k from 0 to k_{max} while inner loops scan statement instances to be executed at time k (elements of the set range(R^k)).

Having represented the constraints of R^k as a Presburger formula, we can easily get a representation of the positive transitive closure of R, R^+, by making k in the formula for A[R^k] existentially quantified, i.e., by adding the quantifier "$\exists$" to k into the constraints of R^k.

Having found the positive transitive closure of relation R, R^+, the transitive closure of R, R^*, is calculated as follows

$$R^* = R^+ \cup I,$$

where I is the identity relation.

We propose a necessary and sufficient condition defining when positive transitive closure calculated according to the suggested technique corresponds to exact positive transitive closure. Further on in this paper transitive closure refers to positive transitive closure.

2 Background and Related Work

This section briefly presents some definitions necessary for the rest of the paper and describes related work for transitive closure computation.

An integer k-tuple is a point in Z^k. An integer tuple relation has the following general form {[*input_list*] →[*output_list*]: *constraints*}, where *input_list* and *output_list* are the lists of variables and/or expressions used to describe input and output tuples, and *constraints* is a Presburger formula describing the constraints imposed upon *input_list* and *output_list* [9].

The transitive closure of a directed graph G=(V, E) is a graph H=(V, F) with edge (v, w) in F if and only if there is a path from v to w in G. There is a path in G if and only if there is an edge (v, w) in H. A graph can be represented with

an integer tuple relation whose domain consists of integer k-tuples and whose range consists of integer k′-tuples, for some fixed k and k′.

We use the following standard operations on relations and sets: union ($\cup$), composition ($\circ$ - for a pair of relations, $\prod$ - for multiple relations), inclusion ($\subseteq$), domain(R) (S=domain(R), x$\in S$ iff $\exists$y s.t. {x$\rightarrow$y}$\in R$), range(R) (S=range(R), y$\in$S iff $\exists$x s.t. {x$\rightarrow$y}$\in$R).

Transitive closures of relations with arbitrary Presburger constraints are not computable in general [5,9]. Two approaches have been studied: finding a particular class of relations for which transitive closure can be computed exactly, and finding an approximation of transitive closure that is "good enough" for a particular class of relations. The general form of a relation is as follows:

$$\left\{ [x] \rightarrow [y] \mid \bigvee_{i=1}^{n} (\exists \alpha_i, A_i(x, y, \alpha_i) \geq b_i) \right\},$$

where x, y, and α_i are integer tuples, A_i is an integer matrix and b_i is a tuple of integer and symbolic values. A distinction is then made when relations contain a single clause (meaning $n = 1$) and are of the form:

$$\{[x] \rightarrow [y] \mid \exists \alpha, A(x, y, \alpha) \geq b\}$$

and when they contain multiple clauses ($n > 1$).

For single clause relations, the exact formulation of transitive closure has been given in particular cases. For relations of the form $R = \{[x] \rightarrow [Ax+b] \mid Cx \geq d\}$ where A and C are integer matrices, b and d are constant integer vectors, Boigelot in [5] (theorem 8.53, p.236) defines a sufficient condition for the computation of R^*. In briefly, this condition states that if there exists p such that A^p is diagonalizable with eigenvalues in $\{0, 1\}$ then R^* is computable in Presburger arithmetics. In particular, idempotent matrices correspond to transitive closures that are represented by periodic sets [6]. Kelly *et al.* [9] also define a simple class of relations, called d-forms, for which the transitive closure is easily computable. These d-form relations are single clause relations with constraints only on the difference of output and input tuples. Bielecki *et al.* [4] show how to compute the transitive closure of a single relation representing graphs of the chain topology. The exact transitive closure calculation is based on resolving a system of recurrence equations being formed from the input and output tuples of a dependence relation.

For multiple clause relations (or union of single clause relations), the computation of transitive closure is more complex. To our knowledge, it is not yet known, for instance, whether the transitive closure of a union of d-forms is computable. Approaches proposed by Boigelot in [5] or Kelly *et al.* [9] compute approximations of transitive closures for such relations. The union of polyhedra defined by multiple clauses is then approximated into the convex hull of the polyhedra. The multiple clauses are therefore approximated into a single clause. Kelly *et al.* propose a flexible approach based on very specific cases with recipes to compute the transitive closure of a union of relations. The conditions for which the computation is exact are not decided beforehand.

In the following sections, we present a new technique to compute the transitive closure for a union of single clause relations, assuming the power k of each individual relation can be computed.

3 Computing the Power k of a Union of Multiple Relations

In this section, we present an approach to calculate the power k of relation R being a union of single clause relations, denoted as A[R^k]. Given a set of $n \geq 2$ relations R_i, i=1,2,...,n, relations $\bar{R}_i$ are obtained by transforming the constraints of R_i so that their domains (and, respectively, ranges) are infinite. When relations $\bar{R}_i$ are commutative (this means that $\bar{R}_i \circ \bar{R}_j = \bar{R}_i \circ \bar{R}_i$ for all i, j), our approach to compute $R^k = (\bigcup_{i=1}^{n} R_i)^k$ is the following

$$A[R^k] = \{[x] \rightarrow [y] \mid x \in \text{domain}(R) \wedge y \in \text{range}(R) \wedge \exists k_i \geq 0, i = 1, 2, ..., n,\ s.t.\ y \in \prod_{i=1}^{n} \bar{R}_i^{k_i}(x) \wedge \sum_{i=1}^{n} k_i = k \wedge k > 0\}. \quad (1)$$

We specify the following classes of relations R_i, i=1,2,...,n, $n \geq 2$, for which the corresponding relations $\bar{R}_i$ are always commutative.

1. Uniform relations, that is, relations of the form $R = \{[x] \rightarrow [x+b] \mid\ Cx \geq d\}$, where C is the integer matrix, b and d are the constant integer vectors;
2. Non-uniform relations of the form $R = \{[x] \rightarrow [Ax] \mid\ Cx \geq d\}$, where A is the diagonal integer matrix, C is the integer matrix, d is the constant integer vector.
3. Relations of the form $R = \{[x] \rightarrow [OP(A,\ x,\ op)] \mid\ Cx \geq d\}$, where A is the diagonal integer matrix, C is the integer matrix, d is the constant integer vector, and OP is the operator producing the following output:

$$OP(A,\ x,\ op) = \begin{vmatrix} a_{11}\ op_1\ x_1 \\ a_{22}\ op_2\ x_2 \\ \quad ... \\ a_{nn}\ op_n\ x_n \end{vmatrix},$$

where

$$A = \begin{vmatrix} a_{11} & 0 & ... & 0 \\ 0 & a_{22} & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & a_{nn} \end{vmatrix},\ x = \begin{vmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{vmatrix},\ op = \begin{vmatrix} op_1 \\ op_2 \\ ... \\ op_n \end{vmatrix},\ op_i \in \{+, \times\},\ 1 \leq i \leq n,$$

op is the same for all relations R_i.

The two previous classes of relations are the special cases of the third class – when *op* is composed of only the addition operators, we deal with the first class, and when *op* is composed of only the multiplication operators, we deal with the second class.

To proove that relations $\bar{R}_1 = \{[x] \to [OP(A,\ x,\ op)]\}$ and $\bar{R}_2 = \{[x] \to [OP(B,\ x,\ op)]\}$ are indeed commutative for given A, B, x, and op, we compute

$$\bar{R}_1 \circ \bar{R}_2 = \left\{ \left| \begin{array}{l} x_1 \\ x_2 \\ \ldots \\ x_n \end{array} \right| \to \left| \begin{array}{ccccc} a_{11} & op_1 & b_{11} & op_1 & x_1 \\ a_{22} & op_2 & b_{22} & op_2 & x_2 \\ & & \ldots & & \\ a_{nn} & op_n & b_{22} & op_n & x_n \end{array} \right| \right\},$$

$$\bar{R}_2 \circ \bar{R}_1 = \left\{ \left| \begin{array}{l} x_1 \\ x_2 \\ \ldots \\ x_n \end{array} \right| \to \left| \begin{array}{ccccc} b_{11} & op_1 & a_{11} & op_1 & x_1 \\ b_{22} & op_2 & a_{22} & op_2 & x_2 \\ & & \ldots & & \\ b_{nn} & op_n & a_{22} & op_n & x_n \end{array} \right| \right\}.$$

Because the addition/multiplication operators represented by each op_i, $1 \leq i \leq n$, are commutative and associative, we can conclude that $\bar{R}_1 \circ \bar{R}_2 = \bar{R}_2 \circ \bar{R}_1$, that is, relations $\bar{R}_1$ and $\bar{R}_2$ are commutative.

For example, relations $\bar{R}_1$=$\{[i,j] \to [i+1, 2*j]\}$ and $\bar{R}_2$=$\{[i,j] \to [i+3, 5*j]\}$ are commutative (they belong to the third class with $op = [+\ *]$), while relations $\bar{R}_3$=$\{[i,j] \to [2*i, j+1]\}$ and $\bar{R}_4$=$\{[i,j] \to [i+3, 5*j]\}$ are not commutative.

For relations of the general form $R = \{[x] \to [Ax + b] \mid Cx \geq d\}$, the satisfaction of the commutativity condition introduced above depends on particular values of A and b and should be verified for each pair of relations R_i and R_j by checking whether $\bar{R}_i \circ \bar{R}_j$ is equal to $\bar{R}_i \circ \bar{R}_i$ using any tool permitting for operations on relations.

The following property guarantees that A$[R^k]$ defined by (1) is either the exact representation of R^k or its overapproximation.

Property 1. $R^k \subseteq A[R^k]$.

Proof. By definition, $R^k = \prod_{j=1}^{k} \bigcup_{i=1}^{n} R_i$. Since $R_1 \subseteq \bar{R}_1$, $R_2 \subseteq \bar{R}_2$, ... $R_n \subseteq \bar{R}_n$, the following is true:

$$\prod_{j=1}^{k} \bigcup_{i=1}^{n} R_i \subseteq \prod_{j=1}^{k} \bigcup_{i=1}^{n} \bar{R}_i.$$

Because relations $\bar{R}_1$, $\bar{R}_2$, ..., $\bar{R}_n$ commute, we can group all the occurrences of the same $\bar{R}_i$ together. It can be shown using properties of a commutative semiring on relations that

$$\prod_{j=1}^{k} \bigcup_{i=1}^{n} \bar{R}_i = \bigcup_{\substack{k_1, k_2, \ldots, k_n \\ \sum k_i = k,\, k_i \geq 0}} \bar{R}_1^{k_1} \circ \bar{R}_2^{k_2} \circ \ldots \circ \bar{R}_n^{k_n}.$$

It implies that ∀ x,y s.t. {x→y}∈ R^k,

$$\exists k_i \geq 0, i = 1, 2, ..., n,\ s.t.\ y \in \prod_{i=1}^{n} \bar{R}_i^{k_i}(x) \wedge \sum_{i=1}^{n} k_i = k \wedge k > 0.$$

Because {x→y}∈ R^k means that x∈ domain(R) and y∈ range(R), we can conclude that $R^k \subseteq$ A[R^k].

Note that for proving the property $R^k \subseteq$ A[R^k] we require that relations $\bar{R}_1$, $\bar{R}_2$, ..., $\bar{R}_n$ be commutative. Under this condition, for A[R^+] formed by making k in the constraints of A[R^k] existentially quantified, we can conclude that $R^+ \subseteq$A[R^+] meaning that A[R^+] is either the exact representation of R^+ or its overapproximation.

The following theorem provides a necessary and sufficient condition when A[R^k] = R^k.

Theorem 1. *A[R^k]=R^k if and only if the following condition is satisfied for all positive values of k:*

$$\forall x \in domain(R) \cup range(R)\ \wedge\ \forall k_i \geq 0, i = 1, 2, ..., n,\ s.t.\ \sum_i k_i = k\ \wedge\ k > 0,$$
$$(\prod_{i=1}^{n} \bar{R}_i^{k_i}(x) \in range(R)) \Rightarrow (\exists h_i \geq 0, i = 1, 2, ..., n,\ s.t.\ \sum_{i=1}^{n} h_i = k, \tag{2}$$
$$\prod_{i=1}^{n} \bar{R}_i^{k_i}(x) = \prod_{i=1}^{n} \bar{R}_i^{h_i}(x)\ \wedge\ x \in \bigcup_{i=1, h_i>0}^{n} domain(R_i)).$$

Proof. The proof consists of two steps. We first prove by induction that Condition (2) is the sufficient condition for A[R^k] to be the exact representation of R^k. Next, we prove that Condition (2) is the necessary condition.

Sufficient condition: (2) ⇒ A[R^k] ⊆ R^k.

Consider x,y s.t. {x→y}∈A[R^1]. The satisfaction of Condition (2) guarantees that ∃i s.t. y∈ $\bar{R}_i$(x) and x∈ domain(R_i). This means that {x→y}∈ R^1, i.e., A[R^1]⊆ R^1.

Now, assuming that A[R^k]=R^k, we want to prove that A[R^{k+1}]=R^{k+1}. Consider x,y s.t. {x→y}∈A[R^{k+1}]. By definition of A[R^{k+1}],

x∈domain(R), y∈range(R), ∃k_i ≥0, i=1,2,...,n,

s.t. $\sum_{i=1}^{n} k_i = k + 1$, y∈ $\prod_{i=1}^{n} \bar{R}_i^{k_i}(x)$.

The satisfaction of Condition (2) guarantees that

∃h_i ≥0, i=1,2,...,n, s.t. $\sum_{i=1}^{n} h_i = k$+1,

y∈ $\prod_{i=1}^{n} \bar{R}_i^{h_i}(x)$ ∈ range(R), x∈ $\bigcup_{i=1, h_i>0}^{n}$ domain(R_i).

In particular, $\exists j \in [1,2,...,n]$ such that $h_j > 0$, $x \in \text{domain}(R_j)$.
Since relation R_j commutes with the remaining relations, we have

$$y \in \prod_{i,i\neq j} \bar{R}_i^{h_i} \circ \bar{R}_j^{h_j}(x).$$

Consider $z \in \bar{R}_j(x)$ such that $y \in \prod_{i,i\neq j} \bar{R}_i^{h_i} \circ \bar{R}_j^{h_j-1}(z)$. Because $x \in \text{domain}(R_j)$ and $z \in \text{range}(R_j)$, the following is true: $\{x \to z\} \in R$. Moreover, because Condition (2) is satisfied for z, $\{z \to y\} \in A[R^k]$. By induction, $\{z \to y\} \in R^k$. Thus, $\{x \to y\} \in R^{k+1}$ and $A[R^{k+1}] \subseteq R^{k+1}$.

Necessary condition: $\mathbf{A}[R^k] \subseteq R^k \Rightarrow$ (2).

Let us now show that Condition (2) of Theorem 1 is the necessary condition for $A[R^k]$ to be the exact representation of R^k. Assume Condition (2) is not satisfied but $A[R^k] \subseteq R^k$.

Since $A[R^k] \subseteq R^k$, $\{x \to y\} \in R^k$. This implies that $x \in \text{domain}(R)$, $y \in \text{range}(R)$ and $\exists i_h \in [1,2,...,n]$, $h=1,2,...,k$, such that $y \in R_{i_1} \circ R_{i_2} \circ ... \circ R_{i_k}(x)$. Note that $x \in \text{domain}(R_{i_k})$. Since $R_1 \subseteq \bar{R}_1$, $R_2 \subseteq \bar{R}_2$, ..., $R_n \subseteq \bar{R}_n$, the following is true: $y \in \bar{R}_{i_1} \circ \bar{R}_{i_2} \circ ... \circ \bar{R}_{i_k}(x)$.

Because all relations R_i commute, we conclude that

$$\exists\ h_i \geq 0,\ i=1,2,...,n,\ \text{s.t.} \sum_{i=1}^{n} h_i = k,\ y \in \prod_{i=1}^{n} \bar{R}_i^{h_i}(x).$$

Such a redefinition of y as well as the knowledge that $x \in \text{domain}(R_{i_k})$ is in contradiction with the assumption that Condition (2) is not satisfied. This proves that indeed Condition (2) is the necessary condition for $A[R^k]$ to be the exact representation of R^k.

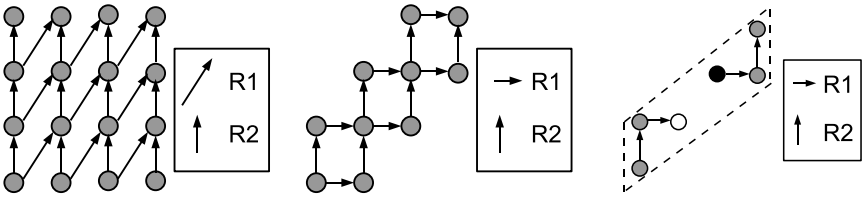


Fig. 1. Examples of graphs (i)

Figure 1 shows three examples of graphs. Condition 2 is satisfied for the first and second graphs, while for the third graph it is not valid: there exists the vertex x=(2,2) (it is shown in white) for which the condition does not hold. Indeed, for the pair x=(2,2) and y=(3,3) (it is shown in black), we can see that $y=\bar{R}_1 \circ \bar{R}_2(x)=\bar{R}_2 \circ \bar{R}_1(x)$, but $x \notin \text{domain}(R_1)$ and $x \notin \text{domain}(R_2)$.

Figure 2 presents two more examples of graphs. Condition (2) is satisfied for the left-hand side graph, while for the right-hand side graph the condition

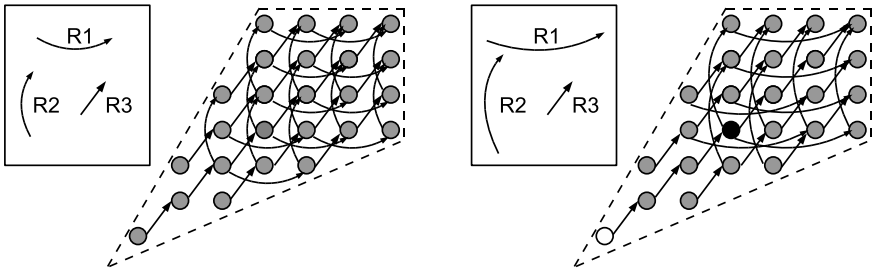


Fig. 2. Examples of graphs (ii)

does not hold: there exists the vertex x=(1,1) (it is shown in white) for which the condition is not satisfied. Indeed, for the pair x=(1,1) and y=(4,4) (it is shown in black), we have $\{x \to y\} \in A[R^2]$ ($y = \bar{R}_1 \circ \bar{R}_2(x) = \bar{R}_2 \circ \bar{R}_1(x)$), but $\{x \to y\} \notin R^2$ ($x \notin$ domain(R_1) and $x \notin$ domain(R_2)). In the matter of fact, $\{x \to y\} \in R^3$, because $y = R_3 \circ R_3 \circ R_3(x)$.

4 Illustrating Examples

In this section, we consider several examples illustrating the presented approach and compare our results with those yielded by approaches presented in paper [9]. We use the Omega syntax [11] to present relations and sets in our examples and the Omega calculator to carry out necessary calculations. Having computed $A[R^k]$, we want to check whether Condition (2) is satisfied. For this purpose, we calculate set X containing elements that satisfy the left-hand side of the implication of Condition (2) and do not satisfy the right-hand side of this implication. We build relation $X1$ whose domain containts elements satisfying the left-hand side of the implication. Next, we build relation $X2$ whose domain contains elements not satisfying the right-hand side of the implication. The computation of the domain$(X1 - X2)$ defines set X. If set X is empty, then we conclude that $A[R^k]$ corresponds to the exact representation of R^k.

Steps to check whether Condition (2) is valid are the following.

1. Compute $A[R^k]$ and $A[R^+]$.
2. Check whether $A[R^k] = R^k$. In the matter of fact,
   (a) form the following relation $X1$

$$X1 = \{x \to y \mid x \in \text{domain}(R) \cup \text{range}(R),\ y \in \text{range}(R),$$
$$\exists k_i \geq 0\ s.t.\ \sum_{i=1}^{n} k_i = k,\ y = \prod_{i=1}^{n} \bar{R}_i^{k_i}(x)\};$$

   (b) form the following relation $X2$

$$X2 = \{x \to y \mid x \in \text{domain}(R) \cup \text{range}(R),\ y \in \text{range}(R),$$
$$\exists k_i \geq 0\ s.t.\ \sum_{i=1}^{n} k_i = k,\ y = \prod_{i=1}^{n} \bar{R}_i^{k_i}(x),\ x \in \bigcup_{i=1, k_i > 0}^{n} \text{domain}(R_i)\};$$

(c) calculate the set $X = \text{domain}(X1 - X2)$. If set X is empty, $\text{A}[R^k] = R^k$ and, respectively, $\text{A}[R^+]=R^+$. The end. Otherwise, $\text{A}[R^k] \neq R^k$.

To illustrate the proposed approach, we start with the following example.

Example 1
R_1:={[i,j]→[i+1,j+1]: 1≤i<n & 1≤j<n},
R_2:={[i,j]→[i+1,j-1]: 1≤i<n & 2≤j≤n}.

Figure 3 illustrates the graph described with relations R_1 and R_2 for n=5. Our task is to compute relation $\text{A}[R^k]$ and to find $\text{A}[R^+]$ using $\text{A}[R^k]$. Relations $\bar{R}_1$ and $\bar{R}_2$ derived from relations R_1 and R_2 by transforming their constraints so that their domains (and, respectively, ranges) are infinite have the following form

$\bar{R}_1$:={[i,j]→[i+1,j+1]},
$\bar{R}_2$:={[i,j]→[i+1,j-1]}.

$\bar{R}_1$ and $\bar{R}_2$ are commutative because $\bar{R}_1 \circ \bar{R}_2 - \bar{R}_2 \circ \bar{R}_1 = \emptyset$ and $\bar{R}_2 \circ \bar{R}_1 - \bar{R}_1 \circ \bar{R}_2 = \emptyset$ (this is easy to check by means of the Omega calculator).

$\bar{R}_1^{k_1}$, $\bar{R}_2^{k_2}$ can be calculated easily using the formula presented in [9] and they are given below.

$\bar{R}_1^{k_1}$:={[i,j]→[i′,j′]: i′=i+k_1 & j′=j+k_1 & $k_1 \geq 0$ },
$\bar{R}_2^{k_2}$:={[i,j]→[i′,j′]: i′=i+k_2 & j′=j-k_2 & $k_2 \geq 0$ }.

The composition $\bar{R}_1^{k_1} \circ \bar{R}_2^{k_2}$ is the following

$\bar{R}_1^{k_1} \circ \bar{R}_2^{k_2}$:={[i,j]→[i′,j′]: i′=i+$k_1$+$k_2$ & j′=j+k_1-k_2 & $k_1 \geq 0$ & $k_2 \geq 0$ }.

Finally, we are able to calculate $\text{A}[R^k]$ that can be represented as

$\text{A}[R^k]$:={[i,j]→[i′,j′]: 1≤i<n & 1≤j≤n & 2≤i′ ≤n & 1≤j′ ≤n & Exists (k_1,k_2: i′=i+k_1+k_2 & j′=j+k_1-k_2 & $k_1 \geq 0$ & $k_2 \geq 0$ & $k_1 + k_2 = k$ & $k \geq 0$)}.

In order to check whether Condition (2) is satisfied, we form relations $X1$ and $X2$, and calculate set X.

$X1$:={[i,j]→[i′,j′]:

$$\underbrace{1 \leq \text{i}, \text{j} \leq n}_{x \in \text{domain}(R) \cup \text{range}(R)} \quad \& \quad \underbrace{2 \leq \text{i}' \leq n \;\&\; 1 \leq \text{j}' \leq n}_{y \in \text{range}(R)} \;\&$$

$$\text{Exists}(k_1, k_2 : \underbrace{\text{i}' = \text{i} + k_1 + k_2 \;\&\; \text{j}' = \text{j} + k_1 - k_2}_{y = \prod_{i=1}^{n} \bar{R}_i^{k_i}(x)} \;\&$$

$$k_1 \geq 0 \;\&\; k_2 \geq 0 \;\&\; k_1 + k_2 = k \;\&\; k > 0)\};$$

$X2$:={[i,j]→[i′,j′]:

$$\underbrace{1 \leq \text{i}, \text{j} \leq n}_{x \in \text{domain}(R) \cup \text{range}(R)} \quad \& \quad \underbrace{2 \leq \text{i}' \leq n \;\&\; 1 \leq \text{j}' \leq n}_{y \in \text{range}(R)} \;\&$$

$$\text{Exists}(k_1, k_2 : \underbrace{\text{i}' = \text{i} + k_1 + k_2 \ \&\ \text{j}' = \text{j} + k_1 - k_2}_{y = \prod_{i=1}^{n} \bar{R}_i^{k_i}(x)} \ \&$$

$$k_1 \geq 0 \ \&\ k_2 \geq 0 \ \&\ k_1 + k_2 = k \ \&\ k > 0 \ \&$$

$$(\underbrace{1 \leq \text{i}, \text{j} < n \ \&\ k_1 > 0}_{x \in \text{domain}(R_1),\ k_1 > 0} \ \text{OR} \ \underbrace{1 \leq \text{i} < n \ \&\ 2 \leq \text{j} < n \ \&\ k_2 > 0}_{x \in \text{domain}(R_2),\ k_2 > 0}))\};$$

$X := \text{domain}(X1 - X2) = \emptyset$.

Because set X is empty, Condition (2) is satisfied and $\text{A}[R^k] = R^k$ meaning that $\text{A}[R^k]$ found by means of our approach is the exact representation of R^k.

To get the positive transitive closure, R^+, of the union of R_1 and R_2, we make k in the formula for $\text{A}[R^k]$ to be existentially quantified, i.e.,

R^+:={[i,j]→[i′,j′]: 1≤i<n & 1≤j≤n & 2≤i′ ≤n & 1≤j′ ≤n & Exists (k_1, k_2, k: i′=i+k_1 + k_2 & j′=j+k_1 − k_2 & k_1 ≥0 & k_2 ≥0 & $k_1 + k_2 = k$ & k ≥0)}.

Applying the Omega calculator implementing the approach to compute transitive closure presented in paper [9], we yield a complex representation of transitive closure including the seven "union" operators, while the both forms (ours and produced by Omega) represent the exact transitive closure.

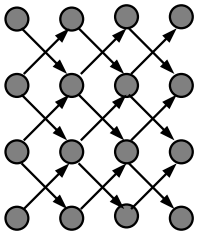


Fig. 3. Graph of Ex.1

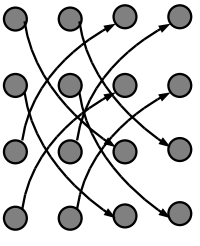


Fig. 4. Graph of Ex.2

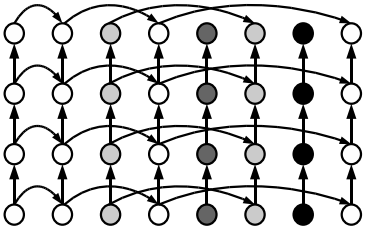


Fig. 5. Graph of Ex. 3

Let us now consider the following set of relations being a slight modification of the previous example. We will show that Omega is fragile to even the simplest modification of Example 1.

Example 2.
R_1:={[i,j]→[i+2,j+2]: 1≤i<n-1 & 1≤j<n-1},
R_2:={[i,j]→[i+2,j-2]: 1≤i<n-1 & 3≤j≤n}.

Figure 4 illustrates the graph described with relations R_1 and R_2 for n=5. Using the Omega calculator, it is easy to state that the corresponding relations $\bar{R}_1$ and $\bar{R}_2$ are commutative and Condition (2) is satisfied.

Applying our approach, we get
R^+:={[i,j]→[i′,j′]: 1≤i<n-1 & 1≤j≤n & 3≤i′ ≤n & 1≤j′ ≤n & Exists (k_1,k_2,k: i′=i+2 ∗ k_1 + 2 ∗ k_2 & j′=j+2 ∗ k_1 − 2 ∗ k_2 & k_1 ≥0 & k_2 ≥0 & k_1 + k_2 = k & k ≥0)},

while applying the approach implemented in the Omega calculator [9] it is not possible to obtain any result.

Let us now show how the proposed method can be applied to non-uniform relations.

Example 3.
R_1:={[i,j]→[2i,j]: 2≤2i≤n & 1≤j≤m},
R_2:={[i,j]→[i,j+1]: 1≤i≤n & 1≤j<m}.

Figure 5 illustrates the graph described with relations R_1 and R_2 for n=8 and m=4. Relations $\bar{R}_1$ and $\bar{R}_2$ derived from relations R_1 and R_2 by transforming their constraints so that their domains (and, respectively, ranges) are infinite have the following form

$\bar{R}_1$:={[i,j]→[2i,j]},
$\bar{R}_2$:={[i,j]→[i,j+1]}.

$\bar{R}_1$ and $\bar{R}_2$ are commutative because $\bar{R}_1 \circ \bar{R}_2 - \bar{R}_2 \circ \bar{R}_1 = \emptyset$ and $\bar{R}_2 \circ \bar{R}_1 - \bar{R}_1 \circ \bar{R}_2 = \emptyset$.

Because relation R_1 is non-uniform, we cannot compute $\bar{R}_1^{k_1}$ using Omega. But using Mathematica according to the approach described in [4] we yield:

$\bar{R}_1^{k_1}$:={[i,j]→[i′,j′]: (i′=i^{2*k_1} & k_1 ≥1 OR i′=i & k_1=0) & j′=j },

while $\bar{R}_2^{k_2}$ (computed either using Omega or according to [4]) is the following:

$\bar{R}_2^{k_2}$:={[i,j]→[i′,j′]: i′=i & j′=j+k_2 & k_2 ≥0 }.

The composition $\bar{R}_1^{k_1} \circ \bar{R}_2^{k_2}$ is of the form

$\bar{R}_1^{k_1} \circ \bar{R}_2^{k_2}$:={[i,j]→[i′,j′]: (i′=i^{2*k_1} & k_1 ≥1 OR i′=i & k_1=0) & j′=j+k_2 & k_2 ≥0}.

Finally, we are able to calculate A[R^k] that can be represented as

A[R^k]:={[i,j]→[i′,j′]: (1≤i≤n & 1≤j<m OR j=m & 1≤i & 2*i≤n) &
(1≤i′ ≤n & 2≤j′ ≤m OR Exists (alpha: 2*alpha = i′ & 2≤i′ ≤n & j′=1)) &
Exists (k_1,k_2: (i′=i^{2*k_1} & k_1 ≥1 OR i′=i & k_1=0) & j′=j+k_2 & k_2 ≥0 & k_1+k_2 = k & k ≥0)}.

Using Mathematica, it can be shown that set X is empty for this example. Thus, A[R^k]=R^k and A[R^+] found making k in the formula for A[R^k] to be existentially quantified is the exact representation of R^+, i.e., A[R^+]=R^+.

5 Conclusion and Future Work

We presented in this paper a novel approach for the computation of the transitive closure of a union of affine and parameterized relations. For relations that

commute when their domains and ranges are set to be infinite, we proposed a formulation for both the power k and the positive transitive closure of the union of $n \geq 2$ relations, for a symbolic k. The precise class of relations for which the computation is exact is defined by the necessary and sufficient condition presented in Section 3. This class includes in particular non-convex relations. For relations beyond this class, we proved that the approach still provides an overapproximation of transitive closure.

In comparison to the heuristic approach presented in [9], ours always permits the computation of the transitive closure of a union of multiple relations. A result can be either exact transitive closure or its overapproximation, depending on the satisfaction of the introduced condition.

In our future research we plan to derive approaches for calculating transitive closure for a union of $n \geq 2$ relations R_i, i=1,2,...,n, whose corresponding relations $\bar{R}_i$ are **not** commutative.

References

1. Alias, C., Barthou, D.: On domain specific languages re-engineering. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 63–77. Springer, Heidelberg (2005)
2. Barthou, D., Feautrier, P., Redon, X.: On the equivalence of two systems of affine recurrence equations. In: Monien, B., Feldmann, R.L. (eds.) Euro-Par 2002. LNCS, vol. 2400, pp. 309–313. Springer, Heidelberg (2002)
3. Beletska, A., Bielecki, W., San Pietro, P.: Extracting coarse-grained parallelism in program loops with the slicing framework. In: ISPDC 2007: Proceedings of the Sixth International Symposium on Parallel and Distributed Computing, p. 29. IEEE Computer Society Press, Washington (2007)
4. Bielecki, W., Klimek, T., Trifunovic, K.: Calculating exact transitive closure for a normalized affine integer tuple relation. Journal of Electronic Notes in Discrete Mathematics 33, 7–14 (2009)
5. Boigelot, B.: Symbolic Methods for Exploring Infinite State Spaces. PhD thesis, Université de Liège (1998)
6. Boigelot, B., Wolper, P.: Symbolic verification with periodic sets. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 55–67. Springer, Heidelberg (1994)
7. Comon, H., Jurski, Y.: Multiple counters automata, safety analysis and presburger arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998)
8. Darte, A., Robert, Y., Vivien, F.: Scheduling and Automatic Parallelization. Birkhaüser (2000)
9. Kelly, W., Pugh, W., Rosser, E., Shpeisman, T.: Transitive closure of infinite graphs and its applications. Int. J. Parallel Programming 24(6), 579–598 (1996)
10. Shashidhar, K.C., Bruynooghe, M., Catthoor, F., Janssens, G.: An automatic verification technique for loop and data reuse transformations based on geometric modeling of programs. Journal of Universal Computer Science 9(3), 248–269 (2003)
11. The Omega project, `http://www.cs.umd.edu/projects/omega`

Matching Techniques Ride to Rescue OLED Displays

Andreas Karrenbauer*

Institute of Mathematics, EPFL, Lausanne, Switzerland
andreas.karrenbauer@epfl.ch

Abstract. Combinatorial optimization problems have recently emerged in the design of controllers for OLED displays. The objective is to decompose an image into subframes minimizing the addressing time and thereby also the amplitude of the electrical current through the diodes, which has a direct impact on the lifetime of such a display. To this end, we model this problem as an integer linear program. Subsequently, we refine this formulation by exploiting the combinatorial structure of the problem. We propose a fully combinatorial separation routine for the LP-relaxation based on matching techniques. It can be used as an oracle in various frameworks to derive approximation algorithms or heuristics. We establish NP-hardness and hardness of approximation. Nevertheless, we are able to work around this issue by only focusing on a subsets of the variables and provide experimental evidence that they are sufficient to come up with near optimal solutions in practice. On this basis, one can derive custom-tailored solutions adapting to technical constraints such as memory requirements. By allowing the addressing of distributed doublelines, we improve the addressing time in cases where previous approaches fall short due to their restriction to consecutive doublelines.

1 Introduction

Organic **L**ight **E**mitting **D**iodes (**OLED**s) have been a hot topic on the display market in the last years as the sizes of commercially available displays increased significantly. Moreover, they provide many advantages over current technology, such as Liquid Crystal Display (LCD).The image and video displayed has a very high contrast and a viewing angle of nearly 180 degrees. It reacts within 10 microseconds, which is much faster than the human eye can catch and is therefore well suited for video applications. Moreover, the display is physically flexible.

There are two different OLED technologies called *active matrix* (AM) and *passive matrix* (PM). The former is more expensive but offers a longer lifetime than the latter. Their limited lifetime is one major reason why there are only small-sized displays on the mass market. For mobile phones or digital cameras,

* Supported by Deutsche Forschungsgemeinschaft (DFG) within Priority Programme 1307 "Algorithm Engineering".

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 110–122, 2009.

large state of the art OLED displays are either too expensive or suffer from insufficient lifetime.

While a lot of research is conducted on the material science side, the so-called *Consecutive Multiline Addressing Scheme* for passive matrix OLED displays [12] tackles the lifetime-problem from an algorithmic point of view. It is based on the fact that equal rows can be displayed simultaneously with a lower electrical current than in a serial manner [3,11]. Here we restrict ourselves to an informal description for self-containment.

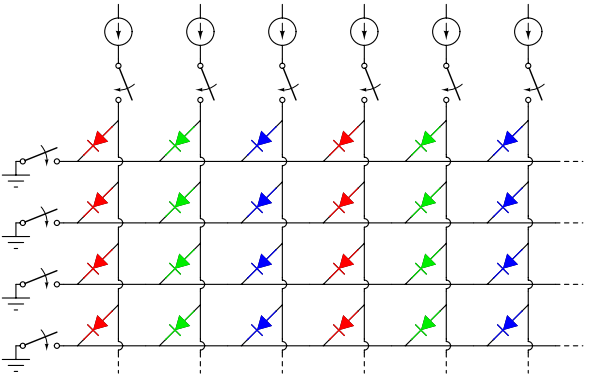


Fig. 1. Schematic electrical circuit of a display

A (passive matrix) OLED display has a matrix structure with n rows and m columns as depicted in Figure 1. At any crossover between a row and a column there is a vertical diode which works as a pixel. The image itself is given as an integral non-negative $n \times m$ matrix. For the sake of simplicity, we first consider the case of binary matrices, i.e. black/white images, and generalize to greyscale and colored images later on.

Consider the contacts for the rows and columns as switches. For the time the switch of row i and column j is closed, an electrical current flows through the diode of pixel (i, j) and it shines. Hence, we can not control each pixel directly. Therefore, such passive matrix displays are traditionally driven row by row in a round-robin fashion. At a sufficiently high framerate, say 50 Hz, the human eye perceives for each pixel only the average over time. Since, we may skip rows that are completely dark, the addressing time varies from to image to image. To maintain the brightness at the same level, we lower the amplitude of electrical current that is sourced into the columns. This procedure has two desired side effects: The power consumption is reduced and their lifetime is extended since high amplitudes of the electrical current are the major issues with respect to the lifetime of the diodes [10]. We can even save more time per frame, if we drive two rows simultaneously. However, this only works, if their content is equal as in the following example. As one can see, we need 5 units of time to display the image

in the traditional way, i.e. row by row, whereas 3 units of time are sufficient with so-called *Distributed Doubleline Addressing* (DDA).

$$\begin{pmatrix} 0&1&1&0 \\ 1&0&0&1 \\ 1&1&1&1 \\ 1&0&0&1 \\ 1&0&0&1 \end{pmatrix} = \begin{pmatrix} 0&1&1&0 \\ 0&0&0&0 \\ 0&1&1&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{pmatrix} + \begin{pmatrix} 0&0&0&0 \\ 1&0&0&1 \\ 1&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \end{pmatrix} + \begin{pmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 1&0&0&1 \\ 1&0&0&1 \end{pmatrix}$$

We thereby gain 40% of the time to display this image and as said before we may decrease the amplitude of the electrical current by that amount. Hence, it remains to find an algorithm that computes such a decomposition to benefit from Distributed Doubleline Addressing (DDA). We use the term *distributed* to distinguish from previous work where it is only allowed to combine *consecutive* lines. That is, we have to display the first matrix on the right-hand side of our example in two steps, which therefore only permits a reduction of the electrical current by 20% in that case.

To benefit from this decomposition in practice, we should adhere to the following design criteria. Since the algorithm has to be implemented on a driver chip attached to the display, it must have low hardware complexity allowing small production costs. Consequently it has to rely only on a small amount of memory and it should be *fully combinatorial*, i.e. only additions, subtractions, and comparisons are used. Though it has to solve or approximate the optimization problem, which is formally described in Section 2, in real-time. We do not fulfill all these requirements in one shot. We rather apply an algorithm engineering process that approaches these goals in several iterations.

Previous Work

Algorithmic questions on the restriction to *Consecutive Doubleline Addressing* (CDA) have been discussed by Eisenbrand et al. [3,4]. In these papers, the authors also considered to combine more than two lines simultaneously, but only consecutive ones. Other approaches based on *Non-negative Matrix Factorization* [6,5] have been outlined by Smith et al. [9] and Smith [8].

Contributions of this Paper

We describe an algorithm engineering process to develop efficient solutions for a real-world problem. That is, we first model the matrix decomposition problem of *Distributed Doubleline Addressing* as an integer program. On this basis, we improve the formulation by exploiting its combinatorial structure until we achieve a solution which is applicable in practice. On the theory side, we prove that computing optimal decompositions is NP-complete and also hard to approximate within a certain constant factor. To this end, we introduce the *Matchable Subset Problem* as a special case of our real-world problem. Though the complexity results sound discouraging, they give useful insight into the structure of

the problem and also hints to the applicable methods. That is, we adopt approximation techniques such as LP-rounding to come up with a promising method in practice. We derive two LP formulations of our problem: A concise one and one with exponentially many constraints. Though the former is of polynomial size, it is impractical and inferior to the latter. This interesting and on the first glance counter-intuitive behavior is due to the fact that we apply techniques known from b-matching to develop an efficient fully combinatorial algorithm for the separation problem of the exponentially many constraints. Finally, we propose parameterized heuristics to achieve a solution, which is applicable in practice. We conclude with a presentation of some computational results showing the improvement with respect to previous work. We thereby show that methods from combinatorial optimization are well-suited to tackle algorithmic challenges in the design of flat panel display drivers.

2 A Linear Programming Formulation

In this section, we will briefly introduce a linear programming model for DDA. The interested reader is referred to [3] for a more profound elaboration on the technical details. For the sake of simplicity, we restrict ourselves to the special case of black/white images given by binary matrices $R = (r_{ij}) \in \{0,1\}^{n \times m}$ for time being. Let the binary variables $f_j(i,k) \in \{0,1\}$ denote whether the switch for column j is closed while the switches of the rows i and k are closed. Note that if i equals k, then the corresponding variables represent a single line. Moreover, $f_j(i,k)$ and $f_j(k,i)$ represent the same switches and hence we implicitly require $f_j(i,k) = f_j(k,i)$ in the following. To get a lossless decomposition of the image R, the following constraints must hold.

$$\sum_{k=1}^{n} f_j(i,k) = r_{ij} \qquad \text{for all } i,j$$

Recall that our objective is to minimize the addressing time for each given image. Clearly, if we have for some pair $\{i,k\}$ that $f_j(i,k) = 0$ for all $j \in \{1,\ldots,m\}$, we can skip this doubleline (or singleline if i equals k). Hence, the total number of subframes is given by

$$\sum_{i \le k} \max\{f_j(i,k) : j = 1,\ldots,m\}.$$

We apply the standard trick to derive a linear programming formulation by replacing each maximum by an auxiliary variable $u(i,k)$. This yields

$$\begin{aligned} \min \quad & \sum_{i \le k} u(i,k) & & \\ \text{s.t.} \quad & \sum_{k=1}^{n} f_j(i,k) = r_{ij} & \text{for all } i,j & \qquad (1) \\ & 0 \le f_j(i,k) \le u(i,k) & \text{for all } i,j,k. & \end{aligned}$$

This LP formulation is not integral in general, which can be verified by an example with three rows and a single column with $R = (1, 1, 1)^T$. The fractional optimum of $\frac{3}{2}$ is attained at $u(1, 2) = u(2, 3) = u(1, 3) = \frac{1}{2}$, whereas the integer optimum is 2. This comes to no surprise regarding the hardness results of Section 4 for integer DDA. However, we stick to the LP formulation as it can be used with well-known approximation techniques, e.g. randomized rounding. We can also work around this issue on the technical side. To this end our display controller must work with a higher precision than the input data. Let us assume for now that it has an arbitrary precision. Then $u(i, k)$ denotes the fraction of time for which the row-switches i and k are simultaneously closed. Moreover, $f_j(i, k)$ is the fraction of time for which the column-switch j is closed while the switches for rows i, k are simultaneously closed. Now the generalization to greyscale images becomes straightforward by taking $R \in [0, 1]^{n \times m}$. Since the images usually have a fixed resolution, we may assume that the input data is scaled to integers in $\{0, \ldots, \varrho\}$ for some integer ϱ, e.g. $\varrho = 255$ for 8-bit resolution. Mathematically, there is no difference between greyscale and colored images. In the latter case, we just have differently colored OLEDs at the respective pixels.

Experimental Evaluation

This basic LP formulation permits a first evaluation of the DDA approach using standard software. Though it is clear that we will not be able to implement a general purpose LP-solver on a chip that drives such a display, we can use the LP-solutions as a benchmark for our further algorithms and heuristics. Although the formulation (1) is concise in the theoretical sense as only a polynomial number (with respect to the input size) of variables and constraints are used, the programs get huge for typical OLED displays. To give the reader a figure, we present the numbers for QQVGA resolution, e.g. subdisplays for mobile phones. There, we have $n = 120$ rows and $m = 3 \cdot 160 = 480$ columns. This means that we have $(m + 1)n(n + 1)/2 \approx 3.5 \cdot 10^6$ variables and about the same number of constraints in our LP. Hence, it comes to no surprise that CPLEX 10.0 takes for a much smaller LP of 30 rows already about 4 minutes on a 2.8 GHz Dualcore AMD Opteron with 16 GB RAM and the run for QQVGA did not finish within 300 hours. Clearly, we must have a deeper look at the theoretical properties of our problem to make any progress. To this end, we refine the formulation of our problem by exploiting its combinatorial structure.

3 Combinatorial Refinement

A closer look at the LP formulation (1) reveals that the objective only depends on the u-variables. Moreover, if those variables were fixed, then the problem would decompose into m independent parts. Hence, we wish to have an efficient method to solve the separation problem for the u-variables. That is, given an assignment to the u-variables, to decide whether all the independent parts are feasible, and if not, to return a violated inequality.

To this end, we introduce a combinatorial formulation of our problem. It is straightforward to consider an undirected graph $G = (V, E)$ where each vertex $i \in V$ corresponds to a row of the display and the edgeset E represents the pairs of row-switches. Note that we allow self-loops in G to model the singlelines. If no further restrictions are given, then G is the complete graph on n nodes.

In the following, we consider the column vectors of an image R as functions $r_j : V \to \mathbb{Z}_{\geq 0}$. A lossless decomposition is then considered as a *perfect r_j-matching problem* for each column $j = 1, \ldots, m$. That is, the set of feasible timings for column j is given by the polyhedron

$$P_j := \left\{ f \in \mathbb{R}^{|E|}_{\geq 0} : f(\delta(i)) = r_j(i) \quad \forall i \in V \right\}$$

where $\delta(\cdot)$ denotes the set of incident edges and $f(\delta(\cdot))$ means the sum over variables of these edges.

Recall that the timings for the row-switches is determined by the maxima over all columns. That is, we have a variable $u(e)$ for each edge $e \in E$. A row-timing $u : E \to \mathbb{Z}_{\geq 0}$ is feasible, if and only if for each column $j \in \{1, \ldots, m\}$ there is a feasible matching $f_j \in P_j$ with $f_j \leq u$. Hence, a row-timing u is feasible for a column j if and only if it is contained in the up-hull of P_j, i.e. $u \in P_j^{\uparrow} := P_j + \mathbb{R}^{|E|}_{\geq 0}$. Thus, the set of feasible row-timings is given by the polyhedron

$$P := \bigcap_{j=1}^{m} P_j^{\uparrow}.$$

The problem can now be divided into two parts and understood as follows.

1. Find a row-timing $u \in P$ that minimizes the sum $u(E)$.
2. For each column $j = 1, \ldots, m$, compute a u-capacitated perfect r_j-matching f_j representing the timings for the corresponding column switches.

Note that in the second step, the columns become independent and the matching problems could be solved in parallel. Moreover, there are several combinatorial algorithms known from literature to solve this task.

It remains to find a good characterization of P, e.g. by an efficient combinatorial algorithm for the separation problem. That is, given a vector u' determine an inequality that is valid for all elements of P but violated for u', or assert that no such inequality exists and hence u' is contained in P.

Theorem 1. *The polyhedron P is determined by the inequalities*

$$2u(E[X]) + u(\delta(X) \setminus \delta(Y)) \geq r_j(X) - r_j(Y) \tag{2}$$

for all $X \subseteq V$, $Y \subset N(X)$, $j = 1, \ldots, m$ where $E[X]$ denotes the inner edges of X excluding the self-loops, $\delta(\cdot)$ also contains the self-loops of X, and $N(X) \subset V$ is the neighborhood of X in G.

We sketch the proof in the following, since it reveals our implementation of the separation routine for these inequalities. We show first that each of the

Inequalities (2) is valid for P. To this end, arbitrarily fix $u \in P$, $X \subseteq V$, $Y \subset N(X)$, and $j \in \{1, \ldots, m\}$. Since $u \in P$, there is a $f \in P_j$ with $f \leq u$. Hence,

$$\begin{aligned} r_j(X) - r_j(Y) &= 2f(E[X]) + f(\delta(X)) - 2f(E[Y]) - f(\delta(Y)) \\ &\leq 2u(E[X]) + u(\delta(X) \setminus \delta(Y)). \end{aligned}$$

To prove sufficiency, we transform the problem to the uncapacitated case. To this end, we split each edge $e = \{i, k\} \in E$ by two new nodes e_1, e_2 such that we get a new graph $G' = (V', E')$ with

$$\begin{aligned} V' &:= V \quad \dot\cup \quad \{e_1, e_2 : e \in E\} \\ E' &:= \big\{\{i, e_1\}, \{e_1, e_2\}, \{e_2, k\} : e = \{i, k\} \in E\big\}. \end{aligned}$$

Note that the self-loops transform into 3-cycles as depicted in Figure 2.

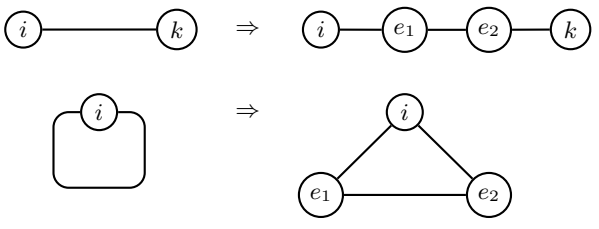


Fig. 2. Transformation to the uncapacitated case

Consider j to be fixed for time being. We define $b : V' \to \mathbb{Z}_{\geq 0}$ with $b(i) := r_j(i)$ for all $i \in V$, $b(e_1) := b(e_2) := u(e)$ for all $e \in E$ that are no self-loops, and $b(e_1) := b(e_2) := u(e)/2$ for all self-loops e. It is easy to verify that a fractional perfect b-matching in G' corresponds to a fractional u-capacitated perfect r_j-matching in G and vice versa. The value attributed to the middle segment in G' determines the slack of the corresponding edge in G. The transformation to the uncapacitated case allows us to use a well-known characterization of the existence of perfect $2b$-matchings (cf. Corollary 31.5a in [7]) that we can state in our context as follows.

Lemma 1. *There is a fractional perfect b-matching for G' if and only if*

$$b(N'(S)) \geq b(S)$$

for each stable set S of G' where N' denotes the neighborhood in G'.

Let S be a stable set in G'. Define the set $X := S \cap V$, i.e. the nodes of S that correspond to nodes in the original graph G. Moreover, let $Y := N'(S \setminus X) \cap V$ and $F \subseteq \delta(Y)$ such that its edges correspond to the nodes of $S \setminus X$ in G'. Hence,

$$b(S) = b(X) + b(S \setminus X) = r_j(X) + u(F)$$

and

$$b(N'(S)) = 2u(E[X]) + u(\delta(X) \setminus F) + u(F) + r_j(Y).$$

It follows that $b(N'(S)) \geq b(S)$ is equivalent to

$$2u(E[X]) + u(\delta(X) \setminus \delta(Y)) \geq r_j(X) - r_j(Y).$$

It remains to show that it is sufficient to consider only $Y \subseteq N(X)$ in the original graph. However, this is easy to see since any $y \in Y \setminus N(X)$ would weaken the right-hand side and would leave the left-hand side unchanged.

Algorithmically, a violated inequality for a given assignment $u : E \to \mathbb{Z}_{\geq 0}$ can be found by a further transformation of the uncapacitated perfect b-matching problem to a transportation problem. That is, we construct a bipartite graph G'' such that each part of the bipartition consists of a copy of V', say $V'' := V_1' \dot\cup V_2'$, and for each edge $\{v, w\} \in E'$ we have the two edges $\{v_1, w_2\}$ and $\{v_2, w_1\}$ in E''. By directing the edges from V_1 to V_2 and considering the nodes of V_1 as supplies and the nodes of V_2 as demands, the separation problem becomes a transshipment problem, which can be solved by a maximum flow computation. If and only if the value of the maximum flow equals $b(V')$, then there exist a fractional perfect b-matching in G'. If the value of the maximum flow, or by duality the minimum cut, is smaller, then the nodes of G'' constituting a minimum cut also represent a vertex cover $y : V'' \to \{0, 1\}$ of the same weight. By setting $z(v) := y(v_1) + y(v_2)$, we get a 2-vertex cover of G' with

$$\sum_{v \in V'} b(v)z(v) < b(V').$$

Moreover, the set $S := \{v \in V' : z(v) = 0\}$ yields a stable set, while $N'(S) = \{v \in V' : z(v) = 2\}$. Hence,

$$b(V') > 2b(N'(S)) + b(V' \setminus N'(S) \setminus S) = b(N'(S)) + b(V' \setminus S),$$

which gives the equivalent violated inequality $b(S) > b(N'(S))$.

3.1 Further Improvements

In principle, we could start the separation with the zero-vector. However, it is much more efficient to provide a starting set of valid inequalities which is of moderate size and easy to solve but still yields a dual solution which is not too far from the optimum.

It is straightforward to select the inequalities arising from the sets $X = \{i\}$ and $Y = \emptyset$ for each $i \in V$. This means, we simply bound the variables in the perfect matching constraints by the corresponding capacities and get

$$u(\delta(i)) \geq \overline{r}_i \tag{3}$$

where $\overline{r}_i := \max\{r_{ij} : j = 1, \ldots, m\}$.

For our application, it is easy to see that the optimal objective value of this partial solution is at least half of the optimum of the whole problem, since $\overline{u}(i,i) := \overline{r}_i$ and $u(i,k) := 0$ for all $i \neq k$ is a feasible solution and

$$2u(E) \geq \sum_{i=1}^{n} \overline{r}_i$$

holds by summing up the inequalities (3) and the non-negativity constraints for $u(i,i)$.

Note that these inequalities together with the non-negativity constraints determine the fractional $\overline{r}$-edge cover polytope. This has the following two consequences for us: Firstly, there is a fully combinatorial algorithm to compute a minimum fractional b-edge cover. Secondly, the integer $\overline{r}$-edge cover polyhedron is contained in the integer hull P_I of P. Hence, we may also add the valid inequalities

$$u(E[X] \cup \delta(X)) \geq \left\lceil \frac{\overline{r}(X)}{2} \right\rceil \qquad \text{for all } X \subseteq V \tag{4}$$

to the description of P_I. Recall that we made the temporary assumption that our display controller works with arbitrary precision. But in the real world, this is hardly possible since our digital circuit shall work with a fixed clock frequency. Hence, there is a minimum amount of time for which switches can be closed and opened again. Thus, we only have fixed precision, which is equivalent to require the variables to be integer by appropriate scaling.

Nevertheless, the separation routine has not become useless since we already get a very simple approximation algorithm by simply rounding each fractional variable to the next greater integer. Note that then the obtained integer solution has an objective value within an additive error of $|E|$. Moreover, the separation routine can be used within the framework of [3] to come up with fully combinatorial heuristics.

It is natural to ask whether there is a completely different approach to solve the problem exactly in polynomial time. But there is little hope because of the NP-completeness result of Section 4. Before we come to this section, we give a brief overview of the experimental results with respect to the combinatorial separation.

3.2 Experimental Evaluation

We implemented the combinatorial separation using the LEDA 6.1 library to solve the transportation problem by the built-in MaxFlow routine. We can use this separation to speed up the solution time of CPLEX. For example, the instance with 30 rows mentionend before is now solved in 14 seconds instead of 240 seconds. However, it is still not possible to solve the LP relaxation of a full QQVGA instance in timely manner.

4 Hardness Results

We show in this section that already the restriction to the black/white case, i.e. binary matrices $R \in \{0,1\}^{n \times m}$, is NP-complete and also hard to approximate. To this end, we define the *Matchable Subset Problem* and analyze its complexity.

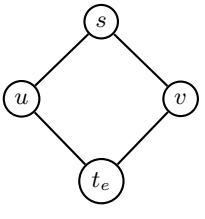
Definition 1 (Matchable Subset Problem). *Given an undirected graph $G = (V, E)$ and m subsets of the nodes $V_1, \ldots, V_m \subseteq V$, find an edgeset $\tilde{E} \subset E$ of minimum cardinality such that for each $j \in \{1, \ldots, m\}$ the set V_j is matchable in $\tilde{G} = (V, \tilde{E})$, i.e. there is a perfect matching in the subgraph of $\tilde{G}$ induced by V_j.*

Theorem 2. *The Matchable Subset Problem is NP-complete, even when restricted to complete graphs (with or without self-loops).*

Proof. Clearly, the problem is in NP. We show hardness by a reduction from vertex cover. Given an undirected graph $G = (V, E)$, we construct an undirected graph $G' = (V', E')$ as follows. Let

$$\begin{aligned} V' &:= \quad \{s\} \quad \dot\cup \quad V \quad \dot\cup \quad \{t_e : e \in E\} \\ E' &:= \quad \big\{\{s,u\} : u \in V\big\} \quad \dot\cup \quad \big\{\{u,t_e\},\{v,t_e\} : e = \{u,v\} \in E\big\} \end{aligned}$$

and let the matchable subsets be induced by the nodes $\{s, u, v, t_e\}$ for each original edge $e = \{u, v\} \in E$. An illustration is given in the left of Figure 3.



$$R = \begin{pmatrix} 1 & \cdots & 1 \\ & A & \\ 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$$

Fig. 3. The construction for the hardness-proof is illustrated on the left and the one for the decomposition problem on the right

Given an edgeset $\tilde{E} \subseteq E'$ such that for every $e = \{u, v\} \in E$ the set $\{s, u, v, t_e\}$ is matchable in the graph $(V', \tilde{E})$, we define the nodeset $C := \{u \in V : \{s, u\} \in \tilde{E}\}$. By construction, we have that $|C| = |\tilde{E}| - |E|$. We show next that C is a vertex cover in G. Let $e = \{u, v\}$ be an arbitrary edge. This implies that $\big\{\{s, u\}, \{s, v\}\big\} \cap \tilde{E} \neq \emptyset$, since $\{s, u, v, t_e\}$ has a perfect matching within $\tilde{E}$. Hence, $\{u, v\} \cap C \neq \emptyset$, which proves that C is a vertex cover in G.

Conversely, if C is a vertex cover in G, then we define $\tilde{E} \subseteq E'$ as follows. We distinguish the two cases if an edge of G is covered by one or two vertices in C. If $e = \{u, v\} \in E$ is covered by exactly one vertex in C, say $C \cup e = \{u\}$, we include the edges $\{s, u\}$ and $\{v, t_e\}$ in $\tilde{E}$. If both of ends of an edge $e = \{u, v\}$

are contained in the cover C, then we include $\{s,u\}$, $\{s,v\}$, and an arbitrary edge of $\{u,t_e\}$ and $\{v,t_e\}$. This yields $|\tilde{E}| = |C| + |E|$ and moreover for each edge $e = \{u,v\} \in E$ the set $\{s,u,v,t_e\}$ has a perfect matching within $\tilde{E}$.

Including also the edges $\{u,v\}$ and $\{s,t_e\}$ in E' does not help as they are only contained in one induced subgraph. Moreover, it is easy to see that self-loops are not suited to improve the solution. □

The relation to DDA is as follows. Consider the complete graph and subsets $V_1,\ldots,V_m$ of its nodes. For each $j = 1,\ldots,m$, let r_j be the characteristic vector of V_j. The shortest DDA timing for the matrix R made up by the column vectors r_j is then equal to the minimum-cardinality edgeset solving the Matchable Subset Problem. Hence, the vertex cover problem for a graph $G = (V,E)$ can be solved as follows. From the node-edge-incidence matrix $A \in \{0,1\}^{|V|\times|E|}$, we construct the image $R \in \{0,1\}^{(1+|V|+|E|)\times|E|}$ as shown on the right of Figure 3. The optimum number of timesteps to display the image using DDA is equal to the minimum size of a vertex cover of G plus the number of edges in G. Note that the constructed graph G' does not contain odd cycles and thus constraining the input to bipartite graphs is not a restriction. Furthermore, it does not make the problem easier if we also consider fractional perfect matchings.

Note that vertex cover is hard to approximate within a factor $\alpha > 1.36$ [2]. Moreover, hardness of approximation also holds for graphs with bounded degree [1]. Thus, we get the following theorem.

Theorem 3. *There is a constant β such that it is NP-hard to approximate the Matchable Subset Problem within a factor of β.*

Proof. An approximation algorithm for the Matchable Subset Problem with guarantee β yields an approximation algorithm for vertex cover with

$$|C| = |\tilde{E}| - |E| \leq \beta(OPT_{vc} + |E|) - |E| \leq [(\Delta+1)\beta - \Delta]OPT_{vc}$$

where Δ denotes the maximum degree of G. Hence, if it is hard to approximate vertex cover on a graph with maximum degree Δ within an approximation factor $\alpha(\Delta)$, then it is hard to approximate the matchable subset problem within an approximation factor

$$\beta = \frac{\alpha(\Delta) + \Delta}{\Delta + 1}$$

The proof is then finished. □

We get $\beta \geq \frac{261}{260} = 1.00385$ by the numbers from [1] and graphs with maximum degree 4. By the previous considerations, this also holds for the shortest DDA timing.

5 Cutting the Bandwidth

In the previous Section, we have seen that the problem is hard for complete graphs. Moreover, the experimental evaluations have shown that the large number of variables prevents ourselves from solving even the LP-relaxation in a

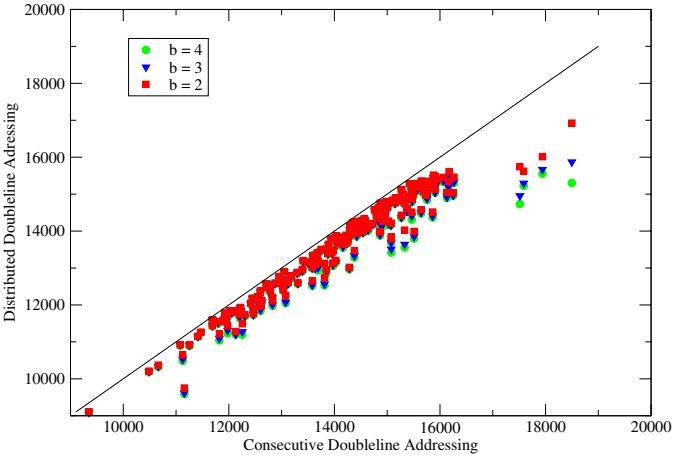


Fig. 4. Comparison between CDA ($b = 1$) and DDA with $b = 2, 3, 4$. The line marks the break even. The lower a symbol the better performs the algorithm.

timely manner. Hence, it is natural to dismiss some (or most) of them, i.e. to consider a suitable subgraph. From previous work [4], we know that Consecutive Doubleline Addressing (CDA) works pretty well in practice. A close look reveals that CDA is a special case of DDA, when we consider a path with n nodes (and self-loops at every node) as the corresponding graph. It is easy to see that this graph has bandwidth 1. If we take the square of this graph, i.e. inserting edges that skip one node on the path, we get bandwidth 2. The third power yields bandwith 3, and so on. Note that the n-th power, i.e. bandwidth n, is again the complete graph. We use the same testset of images as in [3] and [4] in QQVGA resolution (i.e. $n = 120$ and $m = 3 \cdot 160 = 480$) to compare the different bandwidths $b = 1, 2, 3, 4$. We do so with a small uncertainty by taking the mean of the LP-relaxation and the objective value after the naive rounding. This is justified since the error is negligible and the running time for solving the integer linear program is much higher, e.g. 27 seconds compared to 51.5 minutes. In fact, the error is so small that the error bars would not exceed the symbol size in Figure 4.

The reduction factor of the worst instance for CDA dropped from 63% via 58% and 54% to 52% for $b = 2, 3, 4$, respectively. As one can see in Figure 4, there is a saturation with growing bandwidth for real-world instances. However, for artificial images like icons, text, wallpapers for mobile displays, etc. where the mean reduction by CDA is only 63% [4], we strongly believe that DDA with small bandwidth, e.g. $b = 3$, will be of great interest. Moreover, it will decrease the number of bad instances.

Hence, future work includes the development of an efficient hardware implementation specialized for graphs with bounded bandwidth. To this end, it is usefull to investigate the black/white case. While the problem is NP-complete for general graphs as shown in Section 4, is is solvable in polynomial time for $b = 1$ [4]. Since the theoretical research to come up with the polynomial time

algorithm for $b = 1$ has lead to an efficient approximation algorithm in practice, the complexity of the Machable Subset Problem on graphs with bounded bandwidth $b > 1$ is a relevant open problem. A long term goal is to combine more than two distributed rows, but this is also more demanding from the technical point of view.

Since CDA has recently entered the market as part of Dialog Semiconductor's SmartXtend™ technology, which is included a 3" W-QVGA OLED displays of TDK, and the work presented in this paper further improves the addressing time, we strongly believe that DDA will follow with the next version of the display driver.

References

1. Chlebík, M., Chlebíková, J.: Inapproximability Results for Bounded Variants of Optimization Problems. In: Lingas, A., Nilsson, B.J. (eds.) FCT 2003. LNCS, vol. 2751, pp. 27–38. Springer, Heidelberg (2003)
2. Dinur, I., Safra, S.: The importance of being biased. In: Proc. of STOC, pp. 33–42 (2002)
3. Eisenbrand, F., Karrenbauer, A., Skutella, M., Xu, C.: Multiline Addressing by Network Flow. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 744–755. Springer, Heidelberg (2006)
4. Eisenbrand, F., Karrenbauer, A., Xu, C.: Algorithms for Longer OLED Lifetime. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 338–351. Springer, Heidelberg (2007)
5. Lee, D.D., Seung, H.S.: Algorithms for Non-Negative Matrix Factorization. In: Advances in Neural Information Processing Systems, vol. 13 (2001)
6. Paatero, P., Tapper, U.: Positive Matrix Factorization: A Non-Negative Factor Model with Optimal Utilization of Error Estimates of Data Values. Environmetrics 5, 111–126 (1994)
7. Schrijver, A.: Combinatorial Optimization - Polyhedra and Efficiency. Algorithms and Combinatorics 24 (2003)
8. Smith, E.C.: Total Matrix Addressing (TMA™). In: Morreale, J. (ed.) International Symposium Digest of Technical Papers, vol. XXXVIII, pp. 93–96 (2007)
9. Smith, E., Routley, P., Foden, C.: Processing Digital Data Using Non-Negative Matrix Factorization. Patent GB 2421604A (2005)
10. Soh, K.M., Xu, C., Hitzelberger, C.: Dependence of OLED Display Degradation on Driving Conditions. In: Proceedings of SID Mid Europe Chapter Fall Meeting (2006)
11. Xu, C., Karrenbauer, A., Soh, K.M., Wahl, J.: A New Addressing Scheme for PM OLED Display. In: Proc. of International Symposium Digest of Technical Papers, vol. XXXVIII, pp. 97–100 (2007)
12. Xu, C., Wahl, J., Eisenbrand, F., Karrenbauer, A., Soh, K.M., Hitzelberger, C.: Verfahren zur Ansteuerung von Matrixanzeigen. Patent 10 2005 063 159, Germany (2005)

On Open Rectangle-of-Influence Drawings of Planar Graphs

Huaming Zhang* and Milind Vaidya

Computer Science Department, University of Alabama in Huntsville
Huntsville, AL, 35899, USA
{hzhang, mvaidya}@cs.uah.edu

Abstract. We investigate open rectangle-of-influence drawings for irreducible triangulations, which are plane graphs with a quadrangular exterior face, triangular interior faces and no separating triangles. An open rectangle-of-influence drawing of a plane graph G is a type of straight-line grid drawing where there is no vertices drawn in the proper inside of the axis-parallel rectangle defined by the two end vertices of any edge. The algorithm presented by Miura and Nishizeki [8] uses a grid of size $\mathcal{W}+\mathcal{H} \leq$ *(n-1)*, where $\mathcal{W}$ is the width of the grid, $\mathcal{H}$ is the height of the grid and n is the number of vertices in G. Thus the area of the grid is at most $\lceil$(n-1)/2$\rceil$ $\times$ $\lfloor$(n-1)/2$\rfloor$ [8].

In this paper, we prove that the two straight-line grid drawing algorithms for irreducible triangulations from [4] and quadrangulations from [3,5] actually produce open rectangle-of-influence drawings for them respectively. Therefore, the straight-line grid drawing size bounds from [3,4,5] also hold for the open rectangle-of-influence drawings. For irreducible triangulations, the new asymptotical grid size bound is *11n/27* $\times$ *11n/27*. For quadrangulations, our asymptotical grid size bound *13n/27* $\times$ *13n/27* is the first known such bound.

1 Introduction

In this paper, we deal with the straight-line grid drawing of a graph G. We denote by n the number of the vertices in the graph. The grid size is denoted by $\mathcal{W} \times \mathcal{H}$, where $\mathcal{W}$ is called the *width* of the grid and $\mathcal{H}$ is called the *height* of the grid. The grid consists of $(\mathcal{W}+1)$ vertical grid lines and $(\mathcal{H}+1)$ horizontal grid lines.

A *planar graph* is a graph which can be drawn in the plane so that the edges do not intersect at any point other than their end vertices. A planar graph with a fixed planar embedding is called a *plane graph*. A plane graph divides the plane into regions which are called *faces*. The unbounded region is called *exterior face*, the other faces are called *interior faces*. An edge is an *exterior edge* if it belongs to the exterior face. A 3-cycle of a plane graph is a *separating triangle* if its removal separates the graph. A plane graph G is an *irreducible triangulation*, if G has a quadrangular exterior face, triangular interior faces and no separating

* Research supported in part by NSF Grant CCF-0728830.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 123–134, 2009.

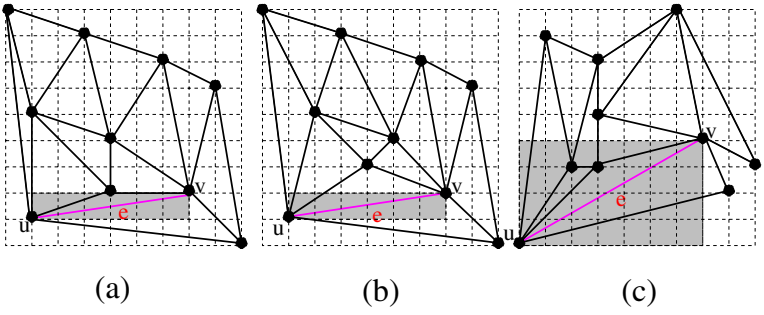


Fig. 1. Three straight-line grid drawings of a graph G: (a) an open rectangle-of-influence drawing, (b) a closed rectangle-of-influence drawing, (c) an ordinary straight-line grid drawing

triangles. An irreducible triangulation G is *internally 4-connected*. Namely, if we add a new vertex into the exterior face of G and connect it to all its four exterior vertices, the resulting graph is 4-connected.

A *straight-line planar* drawing of a planar graph G is a drawing in which all the vertices of the graph are represented by points and all the edges are represented by straight line segments without intersection, except at their common ends. A *straight-line grid drawing* is a straight-line planar drawing in which every vertex is placed on a grid point with integer coordinates. There are many results on straight-line grid drawings under additional constraints so that the drawings look nicer [2,10]. In this paper, we deal with a type of straight-line grid drawings under one additional constraint, known as the *open rectangle-of-influence* drawing; it is a straight-line grid drawing such that there is no vertex in the proper inside of the axis-parallel rectangle defined by the two ends of any edge. A rectangle-of-influence grid drawing is called *closed* if the axis-parallel rectangle defined by the two ends of any edge contains no vertices except its two ends on its boundary. See Figure 1 for three different straight-line grid drawings of a graph G. The axis-parallel rectangles defined by the two ends of the edge $e = (u, v)$ are shaded in Figure 1 (a), Figure 1 (b), and Figure 1 (c) respectively. In Figure 1 (a), the proper inside of the shaded rectangle contains no vertices of G. But there is a vertex $\neq u, v$ on its boundary. In Figure 1 (b), the shaded rectangle contains no vertices except its end vertices u and v, while in Figure 1 (c), the shaded rectangle contains two other vertices $\neq u, v$ in the proper inside of the rectangle. After examining all the other edges in the three drawings, we can easily see that Figure 1 (a) is an open rectangle-of-influence drawing of G, Figure 1 (b) is a closed rectangle-of-influence drawing of G, while Figure 1 (c) is just an ordinary straight-line grid drawing of G. A rectangle-of-influence grid drawing often looks pretty, since vertices are inclined to be separated from edges.

In this paper, we focus on open rectangle-of-influence drawing. From now on in this paper, rectangle-of-influence means open rectangle-of-influence. For a 4-connected n-vertex plane graph G with four or more exterior vertices, Miura et al. [8] presented a rectangle-of-influence drawing algorithm. It uses a grid

of width $\mathcal{W}$ and height $\mathcal{H}$ such that $\mathcal{W} + \mathcal{H} \leq$ *(n-1)*. Therefore, the area of grid used is $\mathcal{W} \times \mathcal{H}$, which is $\leq \lceil$(n-1)/2$\rceil \times \lfloor$(n-1)/2$\rfloor$ [8]. This algorithm is quite sophisticated and it relies on some particular ordering and grouping of the vertices called *4-canonical decomposition* [8]. This makes it difficult to implement and carry it out by hands.

Irreducible triangulations are associated with *transversal structures* [4,6]. Let G be an irreducible triangulation with four exterior vertices W, S, E, N in counterclockwise order. Roughly speaking, a transversal structure $\mathcal{T}(G)$ of G partitions the interior edges of G into two well patterned subsets of red edges and blue edges, where red edges are oriented from S to N and blue edges are oriented from W to E. The subgraph consisting of all the exterior edges and all the red edges (blue edges, respectively) is called the red map G_r (blue map G_b, respectively) of G. By applying the face counting approach [1,9] to G_r and G_b, Fusy [4] introduced a straight-line grid drawing algorithm for irreducible triangulations. It has been shown that for a random irreducible triangulation with n vertices, this algorithm asymptotically uses a grid of size *11n/27* $\times$ *11n/27*, with a high probability up to an additive error of $\mathcal{O}(\sqrt{n})$. In addition, the coordinates calculated by this approach carry clear combinatorial meanings.

Later on, Fusy proposed a related straight-line grid drawing algorithm for quadrangulations [3,5]. For a random quadrangulation with n vertices, the grid size is asymptotically with high probability *13n/27* $\times$ *13n/27* up to an additive error of $\mathcal{O}(\sqrt{n})$.

In this paper, we prove that the above two straight-line grid drawing algorithms by Fusy for irreducible triangulations and quadrangulations actually produce open rectangle-of-influence drawings for them respectively. Therefore, the above mentioned straight-line grid drawing size bounds also hold for the open rectangle-of-influence drawings. Since a 4-connected n-vertex plane graph with four or more exterior vertices can be made into a n-vertex irreducible triangulation by adding diagonals, therefore our new asymptotical grid size bound *11n/27* $\times$ *11n/27* notably improves the previous grid size bound $\mathcal{W}+\mathcal{H} \leq$ *(n-1)* from [8] for 4-connected plane graphs. For quadrangulations, our asymptotical grid size bound *13n/27* $\times$ *13n/27* is the first known bound.

The reminder of the paper is organized as follows. In the Section 2, we introduce some preliminaries. In the Section 3, we introduce the details of the transversal-structure-based straight-line grid drawing algorithm for irreducible triangulations. In Section 4, we prove that the straight-line grid drawing obtained using the transversal structure on irreducible triangulation is indeed a rectangle-of-influence drawing. We also prove that the related straight-line grid drawing on quadrangulation is also a rectangle-of-influence drawing.

2 Transversal Structure for Irreducible Triangulations

We now introduce the details about transversal structures [4,7]. An *orientation* of a graph G assigns a direction to every edge of G. An orientation is said to be *acyclic* if the graph does not contain any directed cycle in the graph. For

every acyclic orientation we can find a vertex with no ingoing edge as *source* and a vertex with no outgoing edge as *sink*. A *bipolar orientation* [4] is an acyclic orientation with a unique source s and a unique sink t. For a 2-connected plane graph G with bipolar orientation we have for any vertex $v \in V$ where $v \neq s, t$, the edges incident to v are partitioned into a non-empty consecutive block of ingoing edges and a non-empty block of outgoing edges around v. Each face f of G has two vertices s_f and t_f such that the boundary of the face f consists of two non-empty directed paths both originating at s_f and ending at t_f. These are called the *left-lateral path* and the *right-lateral path* of f respectively.

Let G be an irreducible triangulation. Let W, S, E and N be four exterior vertices in counterclockwise order. A transversal structure $\mathcal{T}(G)$ of G is a partition of its interior edges into two sets; say in red and blue edges, such that the following conditions are satisfied:

- Interior vertices: In clockwise order around each interior vertex v, its incident edges form a non empty interval of red edges entering v, a non empty interval of blues edges entering v, a non empty interval of red edges leaving v and a non empty interval of blue edges leaving v.
- Exterior vertices: All interior edges incident to N are red edges entering N, all interior edges incident to S are red edges leaving S, all interior edges incident to W are blue edges leaving W and all interior edges incident to E are blues edges entering E. Each of these blocks is non empty.

An *alternating 4-cycle* of a transversal structure $\mathcal{T}(G)$ is a cycle $\mathcal{C} = (e_1, e_2, e_3, e_4)$, where the edges e_1, e_2, e_3, e_4 form a cycle such that they are color alternating. In other words, two adjacent edges in the cycle have different colors [4]. Given a vertex v of the alternating cycle $\mathcal{C}$, we call *left edge* (*right edge*, respectively) of v the edge of $\mathcal{C}$ starting from v and having the exterior of $\mathcal{C}$ on its left (right, respectively). An alternating 4-cycle $\mathcal{C}$ in $\mathcal{T}(G)$ satisfies either of the following configurations [4]: (1) All edges inside $\mathcal{C}$ and incident to a vertex v of $\mathcal{C}$ have the same color of the left edge of v. Then $\mathcal{C}$ is called as a *left alternating cycle*. (2) All edges inside $\mathcal{C}$ and incident to a vertex v of $\mathcal{C}$ have the same color of the right edge of v. Then $\mathcal{C}$ is called as a *right alternating cycle*. A transversal structure $\mathcal{T}(G)$ is called the *minimum transversal structure* if $\mathcal{T}(G)$ has no right alternating 4-cycles. It will be denoted by $\mathcal{T}_m(G)$.

Figure 1 shows an example of an irreducible triangulation G and a transversal structure $\mathcal{T}(G)$ of G. There are two left alternating cycles $\mathcal{C}_1 = ((n, a), (a, d), (d, e), (e, n))$ and $\mathcal{C}_2 = ((p, i), (i, j), (j, k), (k, p))$. Consider the vertex n in $\mathcal{C}_1$. The edge (n, e) is the left edge of n. The edge (n, d) inside the alternating cycle and incident to n has the same color of the edge (n, e). (n, d) also has the same color of the left edge of the vertex d. Hence, $\mathcal{C}_1$ is a left alternating 4-cycle. Similary, one can check that $\mathcal{C}_2$ is also a left alternating 4-cycle. Hence $\mathcal{T}(G)$ has no right alternating 4-cycles; it is the minimum transversal structure $\mathcal{T}_m(G)$ of G.

We have the following three simple facts from [4]:

- **Fact 1:** Let G be an irreducible triangulation with a transversal structure $\mathcal{T}(G)$ of G. Then the oriented red edges form a bipolar orientation of the

plane graph G after removing W, E and all non red edges. Symmertrically, the oriented blue edges form a bipolar orientation of the plane graph G after removing S, N and all the non blue edges [4].

- **Fact 2:** The orientation of the interior edges given by the transversal structure $\mathcal{T}(G)$ is acyclic. The exterior vertices S and W are sources and the exterior vertices N and E are sinks [4].
- **Fact 3:** Let G be an irreducible triangulation. Then the minimum transversal structure $\mathcal{T}_m(G)$ is unique, and it can be computed in linear time [4].

3 A Straight-Line Grid Drawing Algorithm by Fusy

Consider an irreducible triangulation G with a transversal structure $\mathcal{T}(G)$ as defined above. The subgraph of G with all its red colored edges (blue colored edges, respectively) and all its four exterior edges is called the *red map* of G (*blue map* of G, respectively), it is denoted by G_r (G_b, respectively). For an interior vertex v in G_r (G_b, respectively), the edges entering v and the edges leaving v form two non empty contiguous blocks. When walking from u to v in G_r (G_b, respectively), if we always pick the first feasible outgoing edge (i.e., the outgoing edge could still lead to the vertex v) in counterclockwise direction at all its intermediate vertices in the path, then the path is unique and it is called the *rightmost* path from u to v. Similarly, the *leftmost* path always chooses the first feasible clockwise outgoing edge at all its intermediate vertices in the path. In particular, we use $\mathcal{P}_r^{in}(v)$ to denote the rightmost path from S to v and $\mathcal{P}_r^{out}(v)$ to denote the leftmost path from v to N in G_r. We use $\mathcal{P}_b^{in}(v)$ to denote the rightmost path from W to v and $\mathcal{P}_b^{out}(v)$ to denote the leftmost path from v to E in G_b. For any interior vertex v of G, let $P_r(v)$ denote the concatenated path of $\mathcal{P}_r^{in}(v)$ and $\mathcal{P}_r^{out}(v)$, it is called the *separating red path* for v. Similarly, let $P_b(v)$ denote the concatenated path of $\mathcal{P}_b^{in}(v)$ and $\mathcal{P}_b^{out}(v)$, it is called the *separating blue path* for v.

Consider an interior vertex v. The region enclosed by the path (S, W, N) and $P_r(v)$ will be denoted by $\mathcal{R}_{red}^{left}(v)$, the region enclosed by the path (S, E, N) and

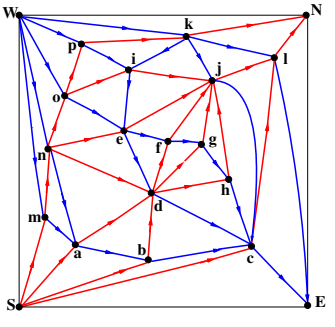


Fig. 2. An irreducible triangulation G and the minimum transversal structure $\mathcal{T}_m(G)$ of G

$P_r(v)$ will be denoted by $\mathcal{R}_{red}^{right}(v)$. The region enclosed by the path (W, S, E) and $P_b(v)$ will be denoted by $\mathcal{R}_{blue}^{right}(v)$, and the region enclosed by the path (W, N, E) and $P_b(v)$ will be denoted by $\mathcal{R}_{blue}^{left}(v)$. Let $x(v)$ be the number of faces in G_r which are within the region $\mathcal{R}_{red}^{left}(v)$, $y(v)$ be the number of faces in G_b which are within the region $\mathcal{R}_{blue}^{right}(v)$.

Let $x(\mathcal{T}(G))$ be the total number of interior faces of G_r and $y(\mathcal{T}(G))$ be the total number of interior faces of G_b. For the exterior vertices, we define $x(W) = 0$, $y(W) = y(\mathcal{T}(G))$, $x(E) = x(\mathcal{T}(G))$, $y(E) = 0$, $x(N) = x(\mathcal{T}(G))$, $y(N) = y(\mathcal{T}(G))$, $x(S) = 0$, and $y(S) = 0$.

For example, Figure 3 (a) presents G_r and Figure 3 (b) presents G_b. Consider the vertex f. In G_r, $P_r(f)$ for f is $P_r(f) = (S, b, d, f, j, l, N)$. In G_b, $P_b(f)$ for f is $P_b(f) = (W, o, e, f, g, h, c, E)$. Therefore, we have $x(f) = 6$ and $y(f) = 4$.

We have the following lemma from [3]:

Lemma 1. *Let G be an irreducible triangulation, W, S, E, N be its four exterior vertices in counterclockwise order.Let $\mathcal{T}(G)$ be a transversal structure of G. Applying $\mathcal{T}(G)$, for each vertex v, embed it on the grid point $(x(v), y(v))$. For each edge of G, simply connect its end vertices by a straight line. Then the drawing is a straight-line grid drawing for G. Its drawing size is $x(\mathcal{T}(G)) \times y(\mathcal{T}(G))$. This drawing is computable in linear time.*

Figure 4 (a) shows the straight-line grid drawing of the graph G in Figure 2, using the transversal structure $\mathcal{T}(G)$ in Figure 3. Observe that there are unused vertical or horizontal grid lines in the drawing plane in Figure 4 (a). It is easy to see that, after deleting all the unused vertical and horizontal grid lines and shifting other vertical grid lines and horizontal grid lines accordingly, we can obtain more compact straight-line grid drawing for G. See Figure 4 (b) for an illustration of the compaction. For an irreducible triangulation G, and a transversal structure $\mathcal{T}(G)$, we will denote the drawing obtained from Lemma 1 by $\mathcal{D}(G)$ and the resulting compact drawing by $\mathcal{D}_c(G)$. The compaction can be understood as a pair of functions from the coordinates of the grid point $(x(u), y(u))$ for a vertex u in drawing $\mathcal{D}(G)$ to the corresponding coordinates of the grid point $(x_c(u), y_c(u))$ for u in the drawing $\mathcal{D}_c(G)$, where we use $x_c(u)$ and $y_c(u)$ to

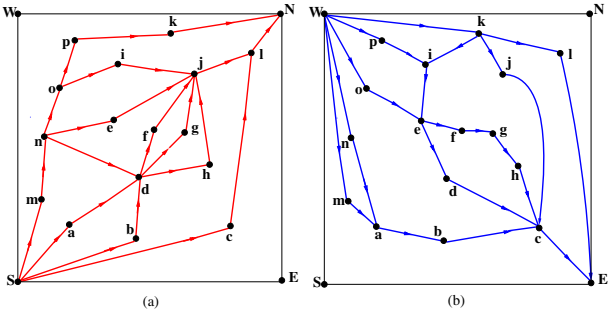


Fig. 3. The G_r and G_b for the transversal structure $\mathcal{T}_m(G)$ of G in Figure 2

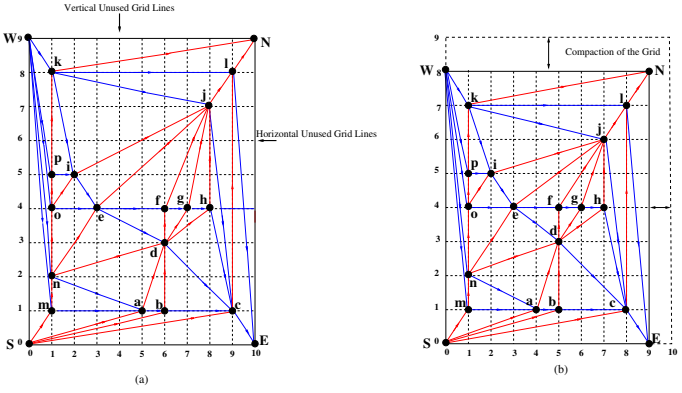


Fig. 4. Straight-line grid drawings of the graph G in Figure 2

denote the $x-$ and $y-$ coordinates for u in $\mathcal{D}_c(G)$. It is easy to observe that, the function from $x(u)$ to $x_c(u)$ and the function from $y(u)$ to $y_c(u)$ are both strictly monotonically increasing. (Note, the functions only consider feasible coordinate values. An integer which is not a coordinate value for any vertices is not in the domain of the function. Take the function from the set of $x(u)$ to the set of $x_c(u)$ for an example. Different coordinate values from the set of $x(u)$ are mapped into different coordinate values in the set of $x_c(u)$. Therefore, this function is strictly monotonically increasing.)

When using the minimum transversal structure $\mathcal{T}_m(G)$, we have the following lemma about the sizes of the drawings from [3]:

Lemma 2. *Let G be an irreducible triangulation with n vertices taken uniformaly at random. Let $\mathcal{W} \times \mathcal{H}$ be the grid size of $\mathcal{D}(G)$ and $\mathcal{W}_c \times \mathcal{H}_c$ be the grid size of $\mathcal{D}_c(G)$ respectively. Then the following results hold asymptotically with high probability, up to fluctuations of order $\sqrt{n}$:*

$$\mathcal{W} \times \mathcal{H} \approx \frac{n}{2} \times \frac{n}{2}, \mathcal{W}_c \times \mathcal{H}_c \approx \frac{11n}{27} \times \frac{11n}{27}.$$

Let us mention that the so-called ϵ-*formulation* holds here: For any $\epsilon > 0$, the probability that $\mathcal{W}$ or $\mathcal{H}$ are outside of $[(1/2-\epsilon)n, (1/2+\epsilon)n]$ and the probability that $\mathcal{W}_c$ and $\mathcal{H}_c$ are outside of $[(11/27-\epsilon)n, (11/27+\epsilon)n]$ are asymptotically exponentially small.

We have the following simple observations regarding the coordinate monotonicity of the above grid drawing algorithms in Lemma 1 and Lemma 2.

Observation 1:

1. If (u, v) is a red path directed from u to v in the red map G_r of $\mathcal{T}(G)$, then $x(u) \leq x(v)$, $y(u) < y(v)$. $x_c(u) \leq x_c(v)$, and $y_c(u) < y_c(v)$.
2. If (u, v) is a blue path directed from u to v in the blue map of G_b of $\mathcal{T}(G)$, then $y(u) \geq y(v)$, $x(u) < x(v)$. $y_c(u) \geq y_c(v)$, and $x_c(u) < x_c(v)$.

First sentences of both Statement 1 and Statement 2 of the above observation are from [11]. Second sentences of both Statement 1 and Statement 2 are also valid because of first sentences of both Statement 1 and Statement 2 and the monotonicity of the function from $x(u)$ to $x_c(u)$ and the monotonicity of the function from $y(u)$ to $y_c(u)$.

4 Rectangle-of-Influence Drawing

In this section, we prove that the above two drawings $\mathcal{D}(G)$ and $\mathcal{D}_c(G)$ of G are actually rectangle-of-influence drawings for the irreducible triangulation G. First we have the following technical lemma:

Lemma 3. *Let G be an irreducible triangulation. Let $\mathcal{T}(G)$ be a transversal structure for G. Let $\mathcal{D}(G)$ and $\mathcal{D}_c(G)$ for G be the two straight-line grid drawings for G as obtained above from $\mathcal{T}(G)$. Then:*

1. *Let w be a vertex and (u, v) be an edge in G. If w is not in the proper inside of the rectangle defined by the two end vertices u and v in $\mathcal{D}(G)$, then w is not in the proper inside of the rectangle defined by the two end vertices u and v in $\mathcal{D}_c(G)$.*
2. *If $\mathcal{D}(G)$ is a rectangle-of-influence drawing for G, then $\mathcal{D}_c(G)$ is also a rectangle-of-influence drawing for G.*

Proof. 1. **Case 1:** If $x(u) = x(v)$ or $y(u) = y(v)$, then after compaction, $x_c(u) = x_c(v)$ or $y_c(u) = y_c(v)$, then the rectangle defined by $(x_c(u), y_c(u))$ and $(x_c(v), y_c(v))$ has no proper inside. Therefore, w cannot be in the proper inside of the rectangle defined by the two end vertices u and v in $\mathcal{D}_c(G)$,
Case 2: Without loss of generality, let's assume that $x(u) < x(v)$ and $y(u) < y(v)$. Consider the vertex w. If w is not in the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$ in $\mathcal{D}(G)$, then using the monotonicity property for the functions from the coordinates in $\mathcal{D}(G)$ to the coordinates in $\mathcal{D}_c(G)$, we have the following: if $x(w) < x(u)$, then $x_c(w) < x_c(u)$; if $x(w) > x(v)$, then $x_c(w) > x_c(u)$; if $y(w) < y(u)$, then $y_c(w) < y_c(u)$; if $y(w) > y(v)$, then $y_c(w) > y_c(u)$.

Therefore, w is not in the proper inside of the rectangle defined by $(x_c(u), y_c(u))$ and $(x_c(v), y_c(v))$ in $\mathcal{D}_c(G)$.

2. It is trivial from the above statement and the definition of the rectangle-of-influence drawing for a graph.

Therefore, we only need to prove that $\mathcal{D}(G)$ is a rectangle-of-influence drawing for G in the following. Next we will have our key technical lemma:

Lemma 4. *Let (u, v) be an edge of G, $(x(u), y(u))$ and $(x(v), y(v))$ be the coordinates of u and v in $\mathcal{D}(G)$. Then there is no vertex in the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$.*

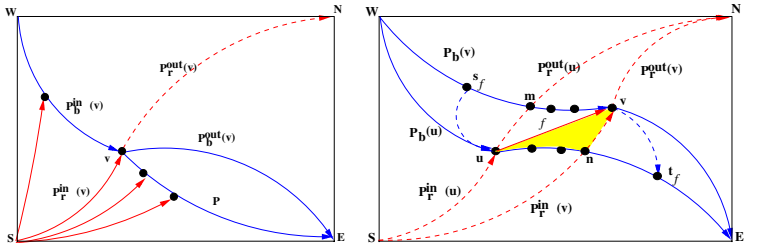


Fig. 5. Proof of Lemma 4

Proof. We have the following several cases for the edge (u, v):

Case 1: (u, v) is an exterior edge. Then either $x(u) = x(v)$ or $y(u) = y(v)$. Therefore the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$ is empty. Hence, no vertex can lie in its interior.

Case 2: (u, v) is a red edge directed from u to v, and u is the exterior vertex S. See Figure 5 (a) for an illustration. First observe that none of the other three exterior vertices can be in the proper inside of the rectangle defined by $(x(S), y(S))$ and $(x(v), y(v))$. Consider $P_b(v)$ and $P_r(v)$, and the rightmost path in G_b from v to E, which is denoted by P. Consider an interior vertex $w \neq u, v$ of G. We have three subcases: Case 2a: w is in the interior of the region $\mathcal{R}^{left}_{blue}(v)$, which is enclosed by $P_b(v)$ and the exterior path (W, N, E). Then $y(w) > y(v)$. Case 2b: w is in the interior of the region $\mathcal{R}^{right}_{red}(v)$, which is enclosd by $P_r(v)$ and the exterior path (S, E, N). Then $x(w) > x(v)$. Case 2c: w is in the region enclosed by (W, S), $P^{in}_r(v)$ and $P^{in}_b(v)$. Clearly, this region contains no vertices in its inside. Therefore, w must be on $P^{in}_b(v)$. Since w is in $P^{in}_b(v)$, there is a directed blue path from w to v. According to Observation 1, $y(w) \geq y(v)$. Therefore, w can not be in the proper inside of the rectangle defined by $(x(S), y(S)) = (0, 0)$ and $(x(v), y(v))$.

Case 3: (u, v) is a red edge directed from u to v, and v is the exterior vertex N. This case is similar to Case 2 and can be proved similarly.

Case 4: Both u and v are interior vertices, and (u, v) is red. Note that, G_r is 2-connected and (u, v) is inside a face of G_b. Denote this face in G_b by f. Let s_f and t_f be the source and sink of the face f in G_b. (u, v) is directed from u to v, therefore, according to the coloring and edge orientation property of a transversal structure, u is on the right-lateral path of f and v is on the left-lateral path of f, and neither u nor v can be s_f or t_f. See Figure 5 (b) as an illustration.

According to the first statement and second statement of Observation 1, we have $x(u) \leq x(v)$ and $y(u) < y(v)$.

Consider the separating blue paths $P_b(u)$ and $P_b(v)$. u is on the right-lateral path of the blue face f, hence the leftmost outgoing blue path from u in G_b must pass through all the vertices in the right-lateral path of f which are after

the vertex u. Denote the subpath of the right-lateral path from u to t_f by $(u, \cdots, n, \cdots, t_f)$. It is contained in ${P_b}^{out}(u)$. Here, the vertex n denotes the last vertex in the path $(u, \cdots, n, \cdots, t_f)$ which has a red outgoing edge to v. In a degenerated case, n could be u.

Consider the separating red path $P_r(v)$. The vertex v is on the left-lateral path of the blue face f and v is not s_f or t_f. Hence according to the local edge coloring and orientation property for the vertex v, all the red ingoing edges to v can only be from the vertices in the right-lateral path of f. (Of course, it is not possible for s_f and t_f to have red outgoing edges to v.) Note that n denotes the last vertex in the path $(u, \cdots, n, \cdots, t_f)$ which has a red outgoing edge to v. Therefore, each vertex between u and n has an outgoing red edge to v. Because the block of the ingoing red edges for v is consecutive. Therefore, for the vertices in the path $(u, \cdots, n, \cdots, t_f)$ which are after n, they cannot have red outgoing edges to v. Hence, n must be contained in the path $P_r^{in}(v)$.

Similary, when we consider $P_r(u)$, the leftmost outgoing red path from u in G_r must pass through a vertex m in the left-lateral path of f. And in a degenerated case, m could be v.

Now we have four subcases according to whether $u = n$ or $m = v$.

Case 4a: $u = n$ and $m = v$. Then clearly $P_r(u)$ is identical to $P_r(v)$. Hence $x(u) = x(v)$. Therefore the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$ is empty. Hence, no vertex can lie in its proper inside.

Case 4b: $u \neq n$ and $m \neq v$. See Figure 5 (b) for an illustration. For any vertex w other than u and v, w must fall in one of the following scenarios:

- w is an exterior vertex. Then clearly w cannot lie in the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$.
- w is contained in the interior of the region $\mathcal{R}_{red}^{left}(u)$, which is enclosed by $P_r(u)$ and the exterior path (S, W, N). Then clearly $x(w) < x(u)$. Because $x(u) \leq x(v)$, so w can not be in the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$.
- w is contained in the interior of the region $\mathcal{R}_{red}^{right}(v)$, which is enclosed by $P_r(v)$ and the exterior path (S, E, N). Then clearly $x(w) > x(v)$. Note that $x(u) \leq x(v)$, so w can not be in the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$.
- w is contained in the interior of the region $\mathcal{R}_{blue}^{right}(u)$, which is enclosed by $P_b(u)$ and the exterior path (W, S, E). Then clearly $y(w) < y(u)$. Because $y(u) < y(v)$, so w can not be in the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$.
- w is contained in the interior of the region $\mathcal{R}_{blue}^{left}(v)$, which is enclosed by $P_b(v)$ and the exterior path (W, N, E). Then clearly $y(w) > y(v)$. Because $y(u) < y(v)$, so w can not be in the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$.
- w is in the blue path $(u, \cdots, n) - \{u\}$, which is directed from u to n. Then according to Observation 1, $y(u) \geq \cdots \geq y(w)$. Note that $y(u) < y(v)$, so w

can not be in the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$.
- w is in the blue path $(m, \cdots, v) - \{v\}$, which is directed from m to v. Then according to Observation 1, $y(w) \geq \cdots \geq y(v)$. Note that $y(u) < y(v)$, so w can not be in the proper inside of the rectangle defined by $(x(u), y(u))$ and $(x(v), y(v))$.

Case 4c: $u = n$ and $m \neq v$. It is the same as Case 4b except that the blue path $(u, \cdots, n) - \{u\}$ is empty. The proof for Case 4b is still valid.

Case 4d: $u \neq n$ and $m = v$. It is the same as Case 4b except that the blue path $(m, \cdots, v) - \{v\}$ is empty. The proof for Case 4b is still valid.

Case 5: (u, v) is a blue edge directed from u to v, and u is the exterior vertex W. This case is symmetrical to Case 2 and can be proved similarly.

Case 6: (u, v) is a blue edge directed from u to v, and v is the exterior vertex E. This case is symmetrical to Case 3 and can be proved accordingly.

Case 7: Both u and v are interior vertices, and (u, v) is blue. This case is symmetrical to Case 4 and can be proved symmetrically.

The following main theorem is a direct result of Lemma 1, Lemma 2, Lemma 3 and Lemma 4:

Theorem 1. *Let G be an irreducible triangulation with n vertices taken uniformly at random. Then G has a rectangle-of-influence drawing whose grid size is asymptotically $\frac{11n}{27} \times \frac{11n}{27}$ with high probability, up to fluctuations of order $\sqrt{n}$. This drawing can be obtained in linear time.*

Transversal structures have been defined in [4,6] in the case where all interior faces are triangles. However, it is easy to replicate transversal structure definition to *partially triangulated quadrangulations*, where every face is a triangle or a quadrangle, and the exterior face is a quadrangle. The definition for the transversal structure for partially triangulated quadrangulations is exactly the same as for the irreducible triangulations. The only difference is that we do not know a precise condition of existence of transversal structures for partially triangulated quadrangulations. Fusy proved that if a partially triangulated quadrangulation is endowed with a transversal structure, then using the same way to calculate coordinates, Lemma 1 can easily be replicated [3,5]. The compaction method can also be easily replicated. Hence Lemma 3 and Lemma 4 can also be replicated.

When given a quadrangulation G with n vertices, Fusy proved that we can smartly insert certain edges into G to make it a partially triangulated quadrangulation G' and during the process of inserting edges, a transversal structure $\mathcal{T}(G')$ is of G' is also constructed. This process can be done in linear time [3,5]. Applying this transversal structure, a rectangle-of-influence drawing can be obtained in linear time for G'. After deleting the inserted edges, a rectangle-of-influence drawing of G is thus obtained. In addition, for a random quadrangulation with n vertices, the grid size produced by the above approach is asymptotically with high probability *13n/27* $\times$ *13n/27* up to an additive error of $\mathcal{O}(\sqrt{n})$. Therefore, we can have the following theorem:

Theorem 2. *Let G be an quadrangulation with n vertices taken uniformly at random. Then G has a rectangle-of-influence drawing obtainable in linear time whose grid size is asymptotically with high probability* $13n/27 \times 13n/27$ *up to an additive error of* $\mathcal{O}(\sqrt{n})$.

References

1. Bonichon, N., Felsner, S., Mosbah, M.: Convex Drawings of 3-Connected Plane Graphs. Algorithmica 47, 399–420 (2007)
2. Chiba, N., Onoguchi, K., Nishizeki, T.: Drawing Planar Graphs Nicely. Acta Informatica 22, 187–201 (1985)
3. Fusy, E.: Straight-Line Drawing of Quadrangulations. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 234–239. Springer, Heidelberg (2007)
4. Fusy, E.: Transversal Structures on Triangulations: Combinatorial Study and Straight Line Drawing. Ecole Polytechnique (2006)
5. Fusy, E.: Combinatoire des Cartes Planaires et Applications Algorithmiques. PhD dissertation, Ecole Polytechnique (2007)
6. He, X.: On Finding the Rectangular Duals of Planar Tringulated Graphs. SIAM Journal on Computing 22, 1218–1226 (1993)
7. Kant, G., He, X.: Regular Edge Labeling of 4-Connected Plane Graphs and its Applications in Graph Drawing Problems. Theoretical Computer Science 172, 175–193 (1997)
8. Miura, K., Nishizeki, T.: Rectangle of Four-Connected Plane Graphs. In: Asia-Pacific Symposium on Information Visualization (APVIS), Sydney, Australia. Conferences in Research and Practice in Information Technology, vol. 45 (2005)
9. Schnyder, W.: Planar Graphs and Poset Dimension. Order 5, 323–343 (1989)
10. Tutte, W.T.: Drawing Planar Graphs Nicely. Proceedings of London Math. Soc. 13, 743–768 (1963)
11. Zhang, H.: Planar Polyline Drawings vis Graph Transformation. Algorithmica (to appear)

An Effective Hybrid Algorithm for the Circles and Spheres Packing Problems

Jingfa Liu, Yonglei Yao, Yu Zheng, Huantong Geng, and Guocheng Zhou*

Computer and Software Institute
Nanjing University of Information Science and Technology
Nanjing 210044, China
{jfliu,ylyao,yzheng,htgeng}@nuist.edu.cn

Abstract. Given a fixed set of equal or unequal circular objects, the problem we deal with consists in finding the smallest container within which the objects can be packed without overlap. Circular containers are considered. Moreover, 2D and 3D problems are treated. Lacking powerful optimization method is the key obstacle to solve this kind of problems. The energy landscape paving (ELP) method is a class of heuristic global optimization algorithm. By combining the improved ELP method with the gradient descent (GD) procedure based on adaptive step length, a hybrid algorithm ELPGD for the circles and spheres packing problems is put forward. The experimental results on a series of representative circular packing instances taken from the literature show the effectiveness of the proposed algorithm for the circles packing problem, and the results on a set of unitary spherical packing instances are also presented for the spheres packing problem for future comparison with other methods.

Keywords: Circular packing, Spherical packing, Combinatorial optimization, Energy landscape paving, Gradient descent method.

1 Introduction

Given n objects and a container, each with given shape and size, the packing problem is concerned with how to pack these objects into the container with no overlap of any pair of them. The packing problem has a wide spectrum of applications [3,5,13]. It is encountered in a variety of real world applications including production and packing for the textile, apparel, naval, automobile, aerospace, and food industries. Many various optional features exist on this problem, e.g., the container can be circle,sphere, rectangle, cuboid or polygon, and the objects can be circular, rectangular, spherical or irregular. This paper discusses the circles packing problem (CPP) where the objects and container are circular and the spheres packing problem (SPP) where the objects and container are spherical. These problems have proven to be NP-hard [3], i.e., no procedure is able to

* This work is supported by the Foundation of Nanjing University of Information Science and Technology, Qing Lan Project and China Postdoctoral Science Foundation funded project (20080431114).

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 135–144, 2009.

exactly solve them in polynomial time unless P=NP. So, some heuristic methods are generally proposed to solve them.

For the problem of packing equal circles, Birgin, et al. [3], Dowsland [5], Lubachevsky and Graham [13] developed various heuristics to generate approximate solutions. The most recent of these works is due to Mladenović et al. [14], who gave a general reformulation descent heuristic. Some authors have studied the problem of packing unequal circles as well. Based on quasi-physical model, Huang et al. [10,8] and Wang et al. [17] developed heuristics inspired from human packing strategies. Zhang and Deng [18] combined Huang et al.'s approach with simulated annealing to explore the neighborhood of the current solution, and taboo search to be used jump out of local minima. Hifi and M'Hallah [7] proposed a three-phase approximate heuristic. Huang et al. [9] presented two greedy algorithms which use a maximum hole degree rule and a self look-ahead search strategy. Lü et al. [12] incorporated the strategy of maximum hole degree into the PERM scheme. The basic idea of their approach is to evaluate the benefit of a partial configuration using the principle of maximum hole degree, and use the PERM strategy to prune and enrich branches efficiently. Akeb et al. [1] gave an adaptive beam search algorithm that combines the beam search, the local position distance and the dichotomous search strategy.

Stoyan et al. [16] studied the problem of packing various solid spheres into a parallelepiped with minimal height, where a clever approach to jump from a local minimizer to a better one, interchanging the positions of two non-identical circles, is devised, and numerical results with up to 60 spheres show that this strategy yields high-quality solutions.

The energy landscape paving (ELP) [6] is a new global optimization algorithm and an improvement on Monte Carlo method. The ELP method has been successfully applied to solving many protein folding problems [15,11]. By combining the improved ELP method with the gradient descent (GD) procedure based on adaptive step length, an effective hybrid algorithm ELPGD for the circles and spheres packing problem is put forward. The experimental results on a series of representative circular or spherical instances show the effectiveness of the proposed algorithm.

2 The Mathematical Models of the Problems

This paper discusses the circles and spheres packing problems. For n objects (circles or spheres) $c_i, i \in N = \{1, \cdots, n\}$ with given shape and size, the problem consists of finding the smallest container (larger containing circle or sphere) c_0 so that all the objects can be packed into the container without overlapping each other. The problems are stated formally as follows.

Let (x_i, y_i) and $(x_i, y_i, z_i), i \in N = \{1, \cdots, n\}$, denote respectively the coordinates of the centers of n circular or spherical objects c_i with radii $r_i, i = 1, \cdots, n$. The circles packing problem (CPP) or the sphere packing problem (SPP) is

equivalent to finding the minimal radius r_0 of the larger containing circle or sphere c_0 and the coordinates of every circle or sphere c_i so that no two circles or spheres c_i and $c_j, i \neq j$, overlap. The mathematical models of the problems are as follows:

$$(CPP)\begin{cases} \min \quad r_0 \\ s.t.(x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j)^2, i \neq j \in N & (1) \\ \quad x_i^2 + y_i^2 \leq (r_0 - r_i)^2, \qquad i \in N, & (2) \end{cases}$$

$$(SPP)\begin{cases} \min \quad r_0 \\ s.t.(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \geq (r_i + r_j)^2, i \neq j \in N & (3) \\ \quad x_i^2 + y_i^2 + z_i^2 \leq (r_0 - r_i)^2, \qquad i \in N & (4) \end{cases}$$

Constraints (1) and (3) denote that each object in the container should not extend outside the container. Constraints (2) and (4) denote that any pair-wise objects placed in the container cannot overlap each other.

Definition 1. *At any moment, if the positions of n objects in the container are fixed, then we call them a configuration, denoted by X.*

Based on the quasi-physical strategy [8,17], we imagine the all n objects and the container as smooth elastic solids. For a given configuration X, if $d(c_i, c_j) < r_i + r_j$, where $d(c_i, c_j)$ denotes the Euclidean distance between the objects c_i and c_j, then we call two objects $c_i, c_j (i \neq j)$ embed each other, otherwise the objects c_i and c_j do not embed each other. If two objects embed each other, then the embedding depth d_{ij} between them is $d_{ij} = r_i + r_j - d(c_i, c_j), i, j \in N$. Otherwise, $d_{ij} = 0$. Similarly, if $d(c_j, c_0) > r_0 - r_j$, then we call the container c_0 and the object c_j embed each other, otherwise the container c_0 and the object c_j do not embed each other. If the container c_0 and the object c_j embed each other, then the embedding depth d_{0j} between them is $d_{0j} = d(c_j, c_0) + r_j - r_0, j \in N$. Otherwise, $d_{0j} = 0$. When two different objects are imbedded into each other, the extrusive elastic potential energy between them is proportional to the square of the depth of their mutual embedment according to elasticity mechanism. So the extrusive elastic potential energy of the whole systems is

$$E(X) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} k d_{ij}^2$$

Here k is a physical coefficient. Generally, we set $k = 1$. Obviously, $E(X) \geq 0$. For a given radius r_0 of the container(larger containing circle or sphere) , if there exists a configuration X^* subject to $E(X^*) = 0$, then X^* is a feasible solution of CPP or SPP, whereas if $E(X^*) > 0$, then X^* is not a feasible solution of CPP or SPP. Thus, if we find an efficient algorithm to solve optimization problem minimize $E(X)$, we can solve the original CPP or SPP by using some search strategies, for example dichotomous search, to reach the minimal radius of the container. Consequently, we will concentrate our discussion on the optimization

problem minimize $E(X)$ and propose a global optimization algorithm ELPGD to solve it in the following section.

3 Algorithms

In this section, we first introduce the theory and method of the energy landscape paving (ELP) and propose a critical improvement on ELP. Then we give the gradient descent (GD) procedure based on adaptive step length. At last, a hybrid algorithm ELPGD which combines the improved ELP method with the GD procedure is developed.

3.1 Energy Landscape Paving Method and Its Improvement

The energy landscape paving (ELP) method [6] is an improved Monte Carlo (MC) global minimization method, and combines ideas from energy landscape deformation [2] and taboo search [4]. ELP has been applied successfully to rough energy landscape of proteins [6,15,11] for finding low-energy configurations. As all good stochastic global optimizers, it is designed to explore low-energy configurations while avoiding at the same time entrapment in local minima. This is achieved by performing low-temperature MC simulations with a modified energy expression designed to steer the search away from regions that have already been explored. This means that if a configuration X is hit, the energy $E(X)$ is increased by a "penalty" and replaced by energy $\overline{E}(X) = E(X) + f(H(q,t))$, in which the "penalty" term $f(H(q,t))$ is a function of the histogram $H(q,t)$ in a prechosen "order parameter" q . In the present paper we assume that $f(H(q,t)) = kH(q,t)$, where k is a constant. The most simple choice of order parameter q is the energy itself, i.e., $q = E$. The histogram $H(E,t)$ is a function of the energy and the iterative step number t. In the course of execution of ELP, we update the histogram by adding 1 to the corresponding bin if only the found energy $E(X)$ falls into a certain bin, where a "bin" denotes an entry of the histogram, i.e., a small energy region, and the sizes of all bins in the histogram are equal. The statistical weight for a configuration X is defined as

$$\omega(\overline{E}(X)) = \exp(-\overline{E}(X)/k_BT)$$

where k_BT is the thermal energy at the (low) temperature T, and k_B is Boltzmann constant.

In ELP minimization, the sampling weight of a local minimum configuration decreases with the time that the system stays in that minimum, and consequently the probability to escape the minimum increases. However, in ELP, there exists a technical flaw: to update the old configuration X_1, the simulation accepts the new configuration X_2 only when satisfying the condition expression

$$\mathrm{Ran}(0,1) < \exp\{[\overline{E}(X_1,t) - \overline{E}(X_2,t)]/k_BT\},$$

where $\mathrm{Ran}(0,1)$ denotes a random number between 0 and 1. Consider an attempted move in ELP which yields a new lower energy minimum that has never

been visited before and happens to fall into the same bin which contains other energies previously visited in the earlier steps. Undesirably, the likelihood of accepting this new energy minimum X_2 becomes small as the result of the penalty term $kH(E,t)$, i.e., the ELP minimization may miss this new lower energy configuration X_2 near X_1. To overcome this shortcoming, we give an improvement on ELP. In the improved ELP, the acceptability of the new configuration X_2 is determined by a comparison between $E(X_1)$ and $E(X_2)$, where three cases are possible: (a) $E(X_2) < 10^{-8}$; (b) $E(X_2) < E(X_1)$ and (c) $E(X_2) \geq E(X_1)$. For case (a), the simulation stops with success; for case (b), the simulation unconditionally accepts the new configuration X_2 and starts a new round of iteration; for case (c), if the new configuration X_2 satisfies the condition expression $\mathrm{Ran}(0,1) < exp\{[\overline{E}(X_1,t) - \overline{E}(X_2,t)]/k_BT\}$, then the simulation accepts X_2 and starts a new round of iteration, otherwise does not accept X_2 and restores X_1 as the current configuration.

3.2 Configuration Update Strategy

Definition 2. *In the configuration X, define the relative degree of pain RDP_i of the object c_i as the extrusive elastic potential energy E_i (which is $\sum_{j=0,j\neq i}^{n} d_{ij}^2$) possessed by itself at this moment divided by its radius r_i : $RDP_i = E_i/r_i$.*

Definition 3. *In the configuration X, suppose that the object c_i is the next object to be picked and moved. Define the taboo region as the region where all packed objects (except the object c_i) occupy at this moment in the container and the vacant region as the remainder except the taboo region at this moment in the container.*

In ELP method, each iterative step must update the current configuration, we use the following configuration update strategy to update configuration.

• For the current configuration X_1, we pick out the object c_i with the maximum relative degree of pain RDP_i and randomly produce a point within the container and accept it if only it belongs to the vacant region. Then put the center of c_i at this point, and gain a new configuration X_2.

This strategy can generally pick out the object in most wrong position, i.e., select the object with maximum RDP_i to jump out of local minima. In addition, to put the center of the picked object into the vacant region can also improve the efficiency of ELP. However, during the process of jumping out of local minima, cycling is possible, namely, an object may be selected repeatedly. In order to improve further the efficiency of the exploration process and avoid cycling, we give the following taboo strategy.

• If an object c_i with maximum RDP_i has been put randomly four times within the vacant region in the container and every time the updated configuration X_2 is not accepted, we take its RDP_i as zero in the next jumping time, i.e., the object c_i is forbidden one time.

3.3 Gradient Descent Procedure Based on Adaptive Step Length

In order to search for the local or global optimal minimum near the updated configuration X_2, we introduce the gradient descent (GD) minimization. In the following GD procedure, *acceptenergy* denotes a threshold. If $E(X_2) \geq acceptenergy$, we use GD to search for the local minimum near the updated configuration X_2. If $E(X_2) < acceptenergy$, we use GD to search for the global (or new local) optimal configuration near X_2. The details of GD are given as follows:

GD(X_2)

Begin
  Let$h = 1$.
  Set $\varepsilon = 10^{-8}$.
  Let $X_1 = X_2$, and compute$X_2 = X_1 - \nabla E(X_1) \cdot h$.
  //Here $\nabla E(X_1)$ denotes gradient vector of $E(X)$ at X_1.
  If $E(X_2) \geq acceptenergy$ **then**
    Repeat
      If $E(X_2) \geq E(X_1)$ **then** $h = h \cdot 0.8$;
      $X_1 = X_2$;
      $X_2 = X_1 - \nabla E(X_1) \cdot h$;
    Until $|E(X_1) - E(X_2)| < 0.01$
  If $E(X_2) < acceptenergy$ **then**
    Repeat
      If$E(X_2) \geq E(X_1)$ **then** $h = h \cdot 0.8$;
      $X_1 = X_2$;
      $X_2 = X_1 - \nabla E(X_1) \cdot h$;
    Until $E(X_2) < \varepsilon$ or $h < 10^{-2}$
  Return X_2.
End

3.4 Description of the Hybrid Algorithm

Our hybrid algorithm ELPGD is based on a combination of the gradient descent (GD) procedure and the improved ELP method. After random initialization of the all n objects' coordinates, the computational procedure mainly includes the following two steps, repeated iteratively. The first step is identical to that in ELP. Based on the current configuration X_1 computed from the previous iteration, we consider an attempted move by displacing the configuration X_1 to a new configuration X' by the configuration update strategy described in subsection 3.2 . The configuration X' is then used as the initial guess in a deterministic minimization algorithm, GD, to locate the nearest local or global optimal minimum at X_2. Then, the acceptability of the new configuration X_2 is determined by a comparison between $E(X_1)$ and $E(X_2)$ according of the improved strategy of ELP described in subsection 3.1. The details of the ELPGD algorithm are described as follows:

Program ELPGD

Begin

1. Randomly give n points in the container c_0 with the origin as the center and r_0 as the radius, whereby the initial configuration X_1 is determined.
   Set $T := 5$, $k := 20$. Let $t = 1$ and initialize $H(E, t)$.
   Set acceptenergy $\in [0.01, 1]$.
2. Compute $E(X_1, t)$ and $\overline{E}(X_1, t)$.
3. Pick out the object c_i with maximum RDP_i, set $j := 1$.
4. Randomly produce a point within the container and accept it if only it belongs to the vacant region. Then put the center of c_i at this point, and gain a new configuration X'.
5. Set $X_2 := GD(X', acceptenergy)$.
6. Compute $E(X_2, t)$ and $\overline{E}(X_2, t) = E(X_2, t) + kH(E(X_2), t)$.
7. **If** $E(X_2, t) < 10^{-8}$ **then** save X_2 and stop with success;
   Else if $E(X_2, t) < E(X_1, t)$ **then** accept X_2, i.e.,
   set $X_1 := X_2$, $E(X_1, t) := E(X_2, t)$, go to Step 9;
   Else If $\mathrm{Ran}(0, 1) < \exp\{[\overline{E}(X_1, t) - \overline{E}(X_2, t)]/k_B T\}$
   then accept X_2, go to Step 9;
   Else do not accept X_2, and restore X_1 as the current configuration, $j := j+1$, go to Step 8.
8. **If** $j > 4$ **then** set RDP_i=0, go to Step 3;
   Else go to Step 4.
9. **If** $t < 100,000$ **then** $t := t + 1$, go to Step 3;
   Else stop with failure.

End

4 Experimental Results

To evaluate the performance of our algorithm for the circles packing problem (CPP) and the spheres packing problem (SPP), we implement the hybrid algorithm ELPGD which is coded in Java language. All experiments are run on a Pentium IV 1.6GHz and 512M RAM.

For the CPP, we test all seven instances listed in [8]. These instances include two equal circular packing instances and five unequal circular packing ones. These benchmark instances vary from regular to non-regular, from identical to non-identical, and from small to large problems, which are typical representatives. For each instance, ELPGD is run for 5 times independently. The obtained smallest radius of the larger containing circle over five independent runs and the running time (in second) under this smallest radius are listed in Table 1, in comparison with those obtained by the improved quasi-physical algorithm (IQP). In Table 1, column 2 denotes the number n of circles to be packed, column 3 denotes the radius r_i of each of the n circles, column 4 tallies the larger containing circle radius r_0^{IQP} obtained when applying IQP, and columns 5 and 6 display the smallest radius r_0^{ELPGD} of the larger containing circle obtained

by ELPGD and the running time under this smallest radius, respectively. It is noteworthy that we find new smaller radii (which are reported in boldface) of the larger containing circles missed in [6] for six circular packing instances, while the result of another instance agrees well that obtained by IQP. The geometric configurations of the solutions obtained by ELPGD are shown in Fig.1.

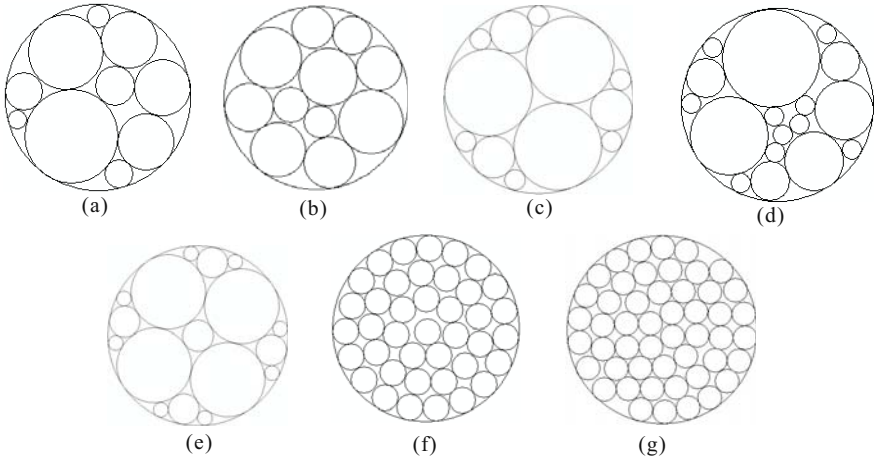


Fig. 1. Geometric configurations of seven circular packing instances obtained by ELPGD: (a) Solution for instance 1 with $r_0 = 99.885$; (b) Solution for instance 2 with $r_0 = 57.095$; (c) Solution for instance 3 with $r_0 = 215.470$; (d) Solution for instance 4 with $r_0 = 49.188$; (e) Solution for instance 5 with $r_0 = 241.421$; (f) Solution for instance 6 with $r_0 = 142.578$; (g) Solution for instance 7 with $r_0 = 158.950$

Table 1. Comparisons between IQP and ELPGD for the circles packing problem

No.	n	$r_i, i = 1, \cdots, n$	r_0^{IQP}	r_0^{ELPGD}	time(s)
1	10	$r_1 = 10, r_2 = 12, r_3 = 15, r_4 = 20, r_5 = 21,$ $r_6 = r_7 = r_8 = 30, r_9 = 40, r_{10} = 50$	99.92	**99.885**	10.082
2	11	$r_1 = 10, r_{i+1} = r_i + 1, i = 1, \cdots, 10$	57.65	**57.095**	134.802
3	12	$r_1 = r_2 = r_3 = 100, r_4 = r_5 = r_6 = 48.26,$ $r_7 = \cdots = r_{12} = 23.72$	215.47	215.470	0.375
4	17	$r_1 = \cdots = r_{10} = 5, r_{11} = r_{12} = r_{13} = 10,$ $r_{14} = r_{15} = 15, r_{16} = 20, r_{17} = 25$	50.00	**49.188**	220.384
5	17	$r_1 = \cdots = r_8 = 20, r_9 = \cdots = r_{13} = 41.415,$ $r_{14} = \cdots = r_{17} = 100$	241.43	**241.421**	0.693
6	40	$r_1 = \cdots = r_{40} = 20$	142.77	**142.578**	1.234
7	50	$r_1 = \cdots = r_{50} = 20$	159.32	**158.950**	85.702

For the SPP, we test fifty equal spherical packing instances, each of which is characterized by n unitary spheres, where $n = 1, 2, 3, \cdots, 50$. For each instance,

ELPGD is run three times independently. The minimal radius OurResult obtained by ELPGD and runtime (in second) under this radius are listed in Table 2 for future comparison with other methods.

Table 2. Results of ELPGD for equal spherical packing instances

n	OurResult	time(s)	n	OurResult	time(s)
1	1.0000006104	19.078	26	3.7544099284	44.485
2	1.9999903454	3952.813	27	3.8232278317	78.328
3	2.1546167514	429.657	28	3.8515149358	117.453
4	2.2246708262	9.063	29	3.8839304098	61.547
5	2.4141344424	0.687	30	3.9261491500	122.335
6	2.4141532067	6.157	31	3.9586163857	196.313
7	2.5911976306	52.203	32	3.9702237305	73.234
8	2.6452760424	14.484	33	4.0297789249	9.344
9	2.7320006394	27.172	34	4.0597699668	136.078
10	2.8324158104	230.313	35	4.0892793950	91.485
11	2.9020670076	9.250	36	4.1228778440	102.703
12	2.9020703080	26.500	37	4.1599920917	518.625
13	3.0012182585	0.578	38	4.1665004961	34.234
14	3.0920038218	34.500	39	4.2333222860	211.078
15	3.1416013295	16.360	40	4.2610782845	226.390
16	3.2156404405	454.438	41	4.2992680183	191.109
17	3.2725342088	9.984	42	4.3090119020	58.907
18	3.3189507406	94.250	43	4.3628504653	132.781
19	3.3861842506	26.484	44	4.3847055183	1257.891
20	3.4771886832	21.610	45	4.4168820245	77.296
21	3.4863138754	18.422	46	4.4490860772	23.594
22	3.5798162793	457.375	47	4.4834917689	258.703
23	3.6274969467	162.125	48	4.5080735101	770.891
24	3.6862536176	130.796	49	4.5211083524	29.359
25	3.6875554317	27.250	50	4.5604157703	49.813

5 Conclusions

This paper solves the circles and spheres packing problems using a new effective hybrid algorithm ELPGD, which combines the improved energy landscape paving (ELP) method with the gradient descent (GD) procedure based on adaptive step length. The energy landscape paving method is mainly introduced to escape from local minima and enhance diversification, and the gradient descent method is used for searching for the local or global optimal minima near an updated configuration. The experimental results on a series of representative equal or unequal circular packing instances taken from the literature show the effectiveness of the proposed algorithm for the circles packing problem, and the results on a set of unitary spherical packing instances are also presented for the spheres packing problem for future comparison with other methods.

References

1. Akeb, H., Hifi, M., M'Hallah, R.: A Beam Search Algorithm for the Circular Packing Problem. Computers and Operations Research 36(5), 1513–1528 (2009)
2. Besold, G., Risbo, J., Mouritsen, O.G.: Efficient Monte Carlo Sampling by Direct Flattening of Free Energy Barriers. Computational Materials Sciences 15(3), 311–340 (1999)
3. Birgin, E.G., Martinez, J.M., Ronconi, D.P.: Optimizing the Packing of Cylinders into a Rectangular Container: A Nonlinear Approach. European Journal of Operational Research 160, 19–33 (2005)
4. Cvijovic, D., Klinowski, J.: Taboo Search: An Approach to the Multiple Minima Problem. Science 267(5198), 664–666 (1995)
5. Dowsland, K.A.: Palletisation of Cylinders in Cases. OR Spektrum 13, 171–172 (1991)
6. Hansmann, U.H.E., Wille, L.T.: Global Optimization by Energy Landscape Paving. Physical Review Letters 88(6), 068105 (2002)
7. Hifi, M., M'Hallah, R.: Adaptive and Restarting Techniques-based Algorithms for Circular Packing Problems. Computational Optimization and Applications 39(1), 17–35 (2008)
8. Huang, W.Q., Kang, Y.: A Short Note on a Simple Search Heuristic for the Disk Packing Problem. Annals of Operations Research 131, 101–108 (2004)
9. Huang, W.Q., Li, Y., Li, C.M., Xu, R.C.: New Heuristics for Packing Unequal Circle into a Circular Container. Computers and Operations Research 33, 2125–2142 (2006)
10. Huang, W.Q., Xu, R.C.: Two Personification Strategies for Solving Circles Packing Problem. Science in China (Series E) 29(4), 347–353 (1994)
11. Liu, J.F., Huang, W.Q.: Studies of Finding Low Energy Configuration in Off-lattice Protein Models. Journal of Theoretical and Computational Chemistry 5(3), 587–594 (2006)
12. Lü, Z.P., Huang, W.Q.: PERM for Solving Circle Packing Problem. Computers and Operations Research 35(5), 1742–1755 (2008)
13. Lubachevsky, B.D., Graham, R.L.: Curved Hexagonal Packing of Equal Circles in a Circle. Discrete & Computational Geometry 18, 179–194 (1997)
14. Mladenović, N., Plastria, F., Urosević, D.: Reformulation Descent Applied to Circle Packing Problems. Computers and Operations Research 32, 2419–2434 (2005)
15. Schug, A., Wenzd, W., Hansmann, U.H.E.: Energy Landscape Paving Simulation of the Trp-cage Protein. Journal of Chemistry Physics 122(19), 194711 (2005)
16. Stoyan, Y.G., Yas'kov, G.N., Scheithauer, G.: Packing Spheres of Various Radii into a Parallelepiped. Technical Report Math-NM-15-2001, TU Dresden (2001)
17. Wang, H.Q., Huang, W.Q., Zhang, Q.A., Xu, D.M.: An Improved Algorithm for the Packing of Unequal Circles within a Larger Containing Circle. European Journal of Operational Research 141(2), 440–453 (2002)
18. Zhang, D.F., Deng, A.S.: An Effective Hybrid Algorithm for the Problem of Packing Circles into a Larger Containing Circle. Computers and Operations Research 32(8), 1941–1951 (2005)

Variable-Size Rectangle Covering

Francis Y.L. Chin*, Hing-Fung Ting**, and Yong Zhang

Department of Computer Science, The University of Hong Kong, Hong Kong
{chin,hfting,yzhang}@cs.hku.hk

Abstract. In wireless communication networks, optimal use of the directional antenna is very important. The *directional antenna coverage (DAC) problem* is to cover all clients with the smallest number of directional antennas. In this paper, we consider the *variable-size rectangle covering (VSRC) problem*, which is a transformation of the DAC problem. There are n points above the base line $y = 0$ of the two-dimensional plane. The target is to cover all these points by minimum number of rectangles, such that the dimension of each rectangle is not fixed but the area is at most 1, and the bottom edge of each rectangle is on the base line $y = 0$. In some applications, the number of rectangles covering any position in the two-dimensional plane is bounded, so we also consider the variation when each position in the plane is covered by no more than two rectangles. We give two polynomial time algorithms for finding the optimal covering. Further, we propose two 2-approximation algorithms that use less running time ($O(n \log n)$ and $O(n)$).

1 Introduction

Let R be a region above the *base*, i.e., $y = 0$, of a two-dimensional plane. An h-rectangle is a rectangle with its lower edge touching the base, and with its height $h > 0$ and width w, such that $w \cdot h \leq 1$, i.e., the area of an h-rectangle is bounded by 1. An *h-rectangle* (h, x^ℓ, x^r) is defined by its height, the position of its left edge and right edge. We say *an h-rectangle is at q* if its left edge is at $x^\ell = q$.

The *Variable-Size Rectangle Covering (VSRC) problem* is to cover a given set of points with the minimum number of h-rectangles. Note that this VSRC problem differs from the traditional set covering problem in several ways. Besides the points in a two-dimensional region to be covered by h-rectangles, the dimensions of the h-rectangles can vary and the lower sides of the h-rectangles have to be grounded (i.e., touching the base $y = 0$). For example, consider the set of points $P = \{(0, 0.05), (3, 0.3), (4, 0.35), (5, 0.45), (6, 0.1), (7, 0.12), (8, 0.09)\}$. P can be covered by the following three h-rectangles (as shown in Figure 1(a)): $(0.3, 0, 3)$, $(0.45, 4, 6)$ and $(0.12, 7, 8)$, or alternatively, by two h-rectangles (as shown in Figure 1(b)): $(0.12, 0, 8)$ and $(0.45, 3, 5)$. The formulation of the problem has some immediate applications; e.g., points can be assumed as dirty stains on a wall, the

* Supported by HK RGC grant HKU-7113/07E.
** Supported by HK RGC grant HKU-7045/02E.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 145–154, 2009.

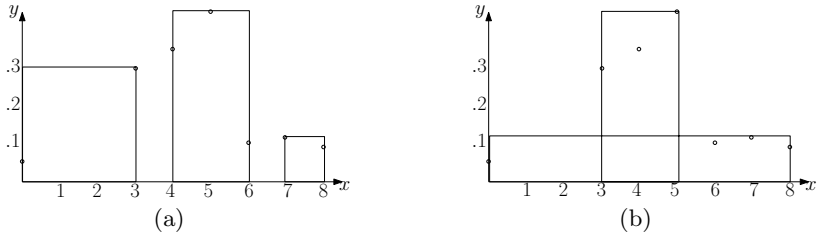


Fig. 1. Two coverings for $P = \{(0, 0.05), (3, 0.3), (4, 0.35), (5, 0.45), (6, 0.1), (7, 0.12), (8, 0.09)\}$

VSRC problem is equivalent to covering these dirty stains with the minimum number of rectangular cardboards standing on the floor and leaning against the wall, and whose dimensions are not fixed but the area of each cardboard is bounded.

An important application of the VSRC problem is to optimize the use of a directional antenna in a wireless network [11,12,13]. Traditionally, a wireless network uses an antenna that is *omnidirectional*, i.e., the signals are sent and received in all directions. In recent years, the use of directional antennas has become more common. A *directional antenna* is one whose signal is concentrated in a certain direction. Comparing with omnidirectional ones, it is far more efficient in terms of the frequency bandwidth and energy. In some antenna designs, multiple beams pointing at different directions can be used simultaneously, e.g., the *multi-beam adaptive array* (MBAA) [1]. For example, consider a set of clients on a two-dimensional plane. There is a base station which provides wireless coverage to these clients. The base station uses a directional antenna to send "beams" to cover the clients so as to provide wireless coverage (as shown in Figure 2).

In an abstract model, the coverage area of a beam can be represented by a circular sector with angle θ and radius r. Typically the covered area of a beam is bounded since the transmission energy of each base station is bounded, which means the wider the beam angle, the shorter the range [2]. For efficiency, antennas may dynamically adjust their orientation, and/or beam angle (and hence the range) [9]. The *directional antennas coverage (DAC) problem* is to cover all clients with the smallest number of beams.

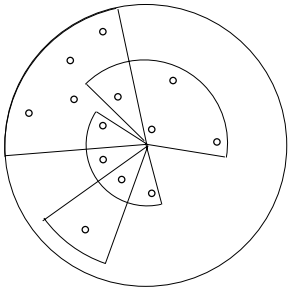


Fig. 2. Clients covered by beams from directional antennas

The DAC problem can be transformed to the VSRC problem as follows. Each client at position (r_i, θ_i) can be converted to point $p_i = (x_i, y_i)$, where $x_i = k\theta_i$ and $y_i = r_i$. Similarly each beam with radius r and angle $\theta(r)$ at a direction θ_0 is equivalent to an h-rectangle (h, x^ℓ, x^r), where $h = r$, $x^\ell = k\theta_0$ and $x^r = k(\theta_0 + \theta(r))$. The bounded area of each beam can be converted to the bounded area of each h-rectangle. Furthermore, since the beam is sent by an antenna at the origin which corresponds to $r = 0$, this would imply that each h-rectangle would have to touch the base $y = 0$ when capturing the points (clients). Let the *wrapped around line* for the VSRC problem be the line $x = 2k\pi$. In this way, the DAC problem can be reduced to the VSRC problem with the points in the region being wrapped around and minimizing the number of beams used would be equivalent to minimizing the number of h-rectangles. From Lemma 1 in Section 2.1, we have the good property of optimal covering, i.e., any two h-rectangles are either disjoint or nested. If we have an algorithm for the VSRC problem without wrapped around point, we can modify the algorithm to deal with wrapped around case (DAC) as follows:

- For each point, cut the region at its x-coordinate then attach the right part to the left of the left part (as shown in Figure 3) and apply the algorithm for finding the optimal covering;
- Return the one with minimal number of h-rectangles.

Thus, we can reduce the algorithm for the VSRC problem without wrapped around point for the DAC problem. On the other hand, given the algorithm for the DAC problem, we can apply it for the VSRC problem without wrapped around point as follows:

- Find the point of the lowest height h;
- Apply the algorithm for VSRC with wrapped around points (DAC) and the wrapped around line is at $x = x_r + 1/h + 1$, where x_r is the x-coordinate of the rightmost point.

In this way, any two points cannot be covered by an h-rectangle crossing the wrapped around line. Thus, the algorithm for the DAC problem can be reduced to the algorithm for the VSRC problem without wrapped around point.

In most applications on wireless communications, to avoid interference, each position in the plane can be covered by a bounded number of antennas. In other applications, this constraint is still reasonable, e.g., in the above example

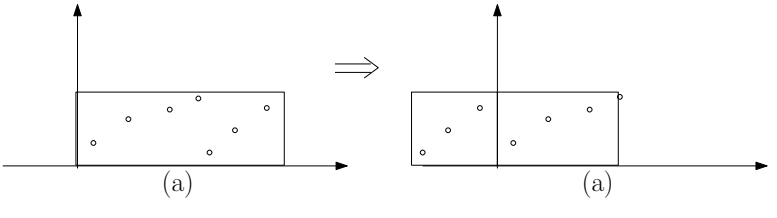


Fig. 3. Cut at the 4th point then attach the right part to the left of the left part

of covering dirty stains on the wall, the number of cardboards covering each position on the wall cannot be too many, that is because the total thickness of cardboards on each point cannot be too thick. In this paper, we consider the variation that any position in two-dimensional plane can be covered by at most two h-rectangles.

The problem of covering a set of points in a two-dimensional plane with the minimum number of unit disks or squares is well studied. It is NP-hard [5] to find the optimal solution, and polynomial time approximation schemes are known [7] (even for any higher but fixed dimensions). Berman et al. [2] used bin packing [8,10] and other techniques to give a 3-approximation algorithm for the capacitated version of DAC problem, i.e., the number of clients covered by a beam is bounded. They also considered the case where the radii of all antennas are fixed and equal. This essentially reduces the problem to one dimension. They [2] gave a tight 1.5-approximation algorithm for this case.

To find the optimal covering for the VSRC problem, we give two polynomial time algorithms: an $O(n^4)$ time algorithm and an $O(n^3)$ time algorithm for the variations without and with the constraint that each position in the plane is covered by no more than two h-rectangles. With less running time ($O(n \log n)$ and $O(n)$), we propose two 2-approximation algorithms.

In Section 2, we give two optimal algorithms for two variations of the VSRC problem, we also propose two 2-approximation algorithms for this problem by using less running time. The conclusion and future research are discussed in Section 3.

2 Algorithms for the VSRC Problem

In this section, we show that because all h-rectangles have to be grounded, the VSRC problem can be solved in polynomial time by dynamic programming,

2.1 An $O(n^4)$ Algorithm

Firstly, we give a polynomial time algorithm for solving the VSRC problem without the constraint that each position in two-dimensional plane is covered by at most a bounded number of h-rectangles.

Consider any two h-rectangles $r_1 = (h_1, x_1^\ell, x_1^r)$ and $r_2 = (h_2, x_2^\ell, x_2^r)$ in the optimal covering. Assuming $x_1^l \leq x_2^l$, there are at most three cases of the relationship between r_1 and r_2.

- $x_1^l < x_1^r \leq x_2^l < x_2^r$, i.e., these two h-rectangles are *disjoint*, which is shown in Figure 4(a)
- $x_1^l \leq x_2^l < x_1^r \leq x_2^r$, i.e., these two h-rectangles are *overlapped*, as shown in the left part of Figure 4(b). In this case, we may transform these two overlapped h-rectangles to two disjoint h-rectangles, as shown in the right part of Figure 4(b), which still cover the same area.
- $x_1^l \leq x_2^l < x_2^r \leq x_1^r$, i.e., these two h-rectangles are *nested*, as shown in Figure 4(c)

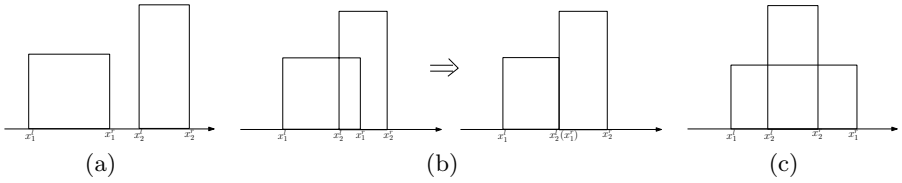


Fig. 4. The relationship between two h-rectangles in the optimal covering

Thus, we have the following lemma for the optimal covering:

Lemma 1. *In the optimal covering, any two h-rectangles are either disjoint or nested.*

Consider the set of points $P = \{p_1, p_2, ..., p_n\}$, where $p_i = (x_i, y_i)$, and $x_1 \leq x_2 \leq ... \leq x_n$. Define $N(i, j, h)$ be the minimum number of h-rectangles needed to cover the points $Q = \{p_k \in P | i \leq k \leq j \text{ and } y_k > h\}$.

According to Lemma 1, the optimal covering for Q must be in one of these forms (i) disjoint optimal covering for $\{p_i ... p_k\} \cap Q$ and $\{p_{k+1} ... p_j\} \cap Q$ for some $i \leq k \leq j$; or (ii) nested covering with an h-rectangle R_{ij} with height $h' = 1/(x_j - x_i)$ covering p_i and p_j and some other points in P and with height no more than h', and another optimal covering for $Q' = \{p_k \in Q | i < k < j \text{ and } y_k > h'\}$.

From the above analysis, we have the recursive formula for the minimum number of h-rectangles covering all the points.

$$N(i,j,h) = \begin{cases} 1 & \text{if all points can be covered by one h-rectangle} \\ \min\Big\{ \min\limits_{i \leq k \leq j} \{N(i,k,h) + N(k+1,j,h)\}, N(i,j,h') + 1 \Big\} & \text{otherwise} \end{cases} \tag{1}$$

As there are $O(n^3)$ terms of $N(i, j, h)$ and from the above formula, it is easy to see that each term takes $O(n)$ time to compute. Thus, we have the following result.

Theorem 1. *The VSRC problem can be solved in $O(n^4)$ time, if each position of the two-dimensional plane can be covered by an unbounded number of h-rectangles.*

This preliminary result shows that the VSRC problem is not NP-hard.

2.2 2-Approximation Algorithms with Less Running Time

In this section, we give two simple algorithms, whose solutions is at most twice of the optimal solution, i.e., 2-approximation. The idea of the first algorithm is from [2].

In any covering strategy, the highest point $p = (x, y)$ must be covered by an h-rectangle with $h \geq y$, and the width of this h-rectangle is at most $1/h$. Note that if $h > y$, the coverage by the h-rectangle higher than y is wasted. Thus, we can assume that $h = y$ and use two disjoint h-rectangles to cover p, on the left and right side of p. After all the covered points are removed, the next highest

point will then be selected and processed in the same way until all points are covered. Formally, the algorithm can be described as follows.

Input: $P = \{(x_i, y_i) | 1 \leq i \leq n\}$
Algorithm **VSRC1**

1: **while** $P \neq \phi$ **do**
2: Find the point (x, y) with the highest height y in P.
3: Create two h-rectangles $(y, (x - 1/y), x)$ and $(y, x, (x + 1/y))$
4: **if** these two h-rectangles overlap with previous h-rectangles **then**
5: Trim the width of these two h-rectangles until there is no overlap.
6: **end if**
7: Remove from P those points covered by these two h-rectangles.
8: **end while**

Theorem 2. *Algorithm VSRC1 is 2-approximation with running time* $O(n \log n)$.

Proof. Let q_i be the highest point selected in round i. We claim that in any covering strategy, any two points q_i and q_j, with $i < j$, must be covered by two disjoint h-rectangles. Otherwise, $|x_i - x_j| \cdot y_i \leq 1$, that means point q_j should be covered and removed in round i, this leads to contradiction.

As the minimum number of h-rectangles needed to cover all the points is no less than the number of highest points selected from each round. Thus, the number of rounds is the lower bound for the optimal covering. Since we create two h-rectangles at each round, VSRC1 is a 2-approximation algorithm.

As updating the set of uncovered points and finding the highest point take $O(\log n)$ time and the number of rounds is at most n, the running time of algorithm VSRC1 is $O(n \log n)$. □

Consider an example with points $p_i = (h, i)$ for $1 \leq i \leq n$ and $h > 1$. It can be shown easily that VSRC1 outputs $2n$ h-rectangles while the optimal covering needs n h-rectangles.

Can we further improve the running time while having the same approximation ratio? The answer is positive. Algorithm VSRC2 is very simple, just scan all the points from left to right once, and create h-rectangle with area at most 1 to cover the leftmost uncovered points. Formally,

Input: $p_i = (x_i, y_i)$ for $1 \leq i \leq n$ and $x_1 \leq x_2 \leq ... \leq x_n$
Algorithm **VSRC2**

1: $i = 1$
2: **while** $i \leq n$ **do**
3: Find the largest k, s.t. points p_i until p_{i+k} can be covered by one h-rectangle with area no more than 1.
4: Create an h-rectangle to cover the points $p_i, \cdots, p_{i+k}$
5: $i = i + k + 1$
6: **end while**

Theorem 3. *Assume* $\{p_i\}$ *where* $x_1 \leq x_2 \leq ... \leq x_n$*, Algorithm VSRC2 is 2-approximation with running time* $O(n)$*.*

Proof. At each round, we create one h-rectangle to cover all those points whose x-coordinates are within an interval, thus, there is no overlap between any two h-rectangles.

Consider the optimal covering OPT, we modify OPT to another covering OPT' without overlapping h-rectangles. For any interval $[l, r]$ covered by more than one h-rectangles in OPT, we use the h-rectangle with the highest height to cover this interval. For example, in the optimal covering, the interval $[l, r]$ is covered by three h-rectangles: (h_1, l_1, r_1), (h_2, l_2, r_2) and (h_3, l, r), where $h_1 \leq h_2 \leq h_3$, after the modification, the interval $[l, r]$ is only covered by one h-rectangle (h_3, l, r). Note that some h-rectangles with lower height in OPT may be split to several h-rectangles in OPT', as shown in Figure 5. In such modification, if two h-rectangles are nested in the optimal covering, the h-rectangle with lower height will be split to more than one h-rectangles.

Consider the example shown in Figure 5, in the optimal covering as shown in Figure 5(a), there are four h-rectangles 1, 2, 3 and 4, h-rectangle 1 is of the lowest height and overlaps with the other three h-rectangles of higher heights. After modification, h-rectangle 1 is split to three h-rectangles, say $1'$, $1''$ and $1'''$, with less widths. Any split occurs from its overlap with an h-rectangle of higher height. We can associate each part, except the leftmost one, of a split to the right edge of the overlapping h-rectangle with higher height. In this example, h-rectangles $1''$ and $1'''$ are associated to the right edges of h-rectangles 3 and 4 respectively. It is easy to see that the right edge of each h-rectangle in OPT can associate with at most one split part in OPT'. Thus, the number of disjoint h-rectangles in OPT' is at most twice to the number of h-rectangles in OPT.

In each round of Algorithm VSRC2, an h-rectangle is created to cover as many points as possible. We can prove by induction from the leftmost point to the rightmost point that the number of h-rectangles created by VSRC2 is no greater than the number of h-rectangles of OPT'. Thus, we can say that VSRC2 is 2-approximation.

At each round, as the points are in order with respect to x_i, finding the largest k such that point p_i until p_{i+k} can be covered by one h-rectangle takes $O(k)$

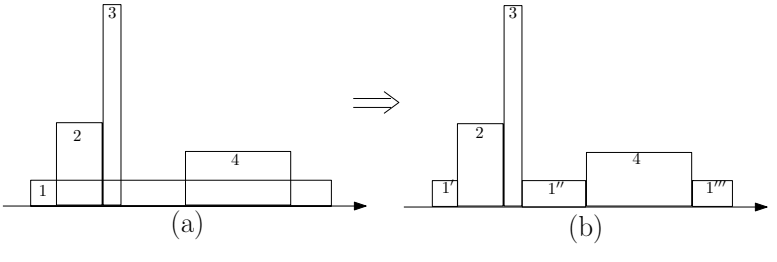


Fig. 5. Example of modify the optimal covering OPT to disjoint covering OPT'

time. As we can create an h-rectangle in each round in constant time, the total running time of VSRC2 is bounded by $O(n)$. □

Consider the example shown in Figure 6, there are 10 points (a to j) to be covered in the plane. The heights of these points are in three levels: a, b, i and j are of the same lowest height; c, d, g and h are of the same middle height; while e and f are of the same highest height. The points of the same height can be covered by an h-rectangle, but any two leftmost points of any two levels cannot be covered by an h-rectangle. Similarly, any two rightmost points of any two levels cannot be covered by an h-rectangle. For example, points a and c, c and e, f and h, h and j cannot be covered by an h-rectangle. To deal with this example, the optimal covering uses three h-rectangles (1, 2 and 3) while Algorithm VSRC2 uses 5 h-rectangles (1, 2, 3, 2' and 3').

We can generalize this example, force each h-rectangle except the highest one from the optimal covering covers 4 points of the same height, while the highest h-rectangle covers 2 points. When applying VSRC2, all h-rectangles except the one with highest height will be split to two h-rectangles with less width. In this way, we can force the optimal covering uses k h-rectangles while Algorithm VSRC2 uses $2k-1$ h-rectangles. Thus, the approximation ratio 2 is tight for VSRC2.

2.3 Each Position Covered by at Most 2 h-Rectangles

In the previous covering strategy, some positions in two-dimensional plane may be in the overlapped area of more than 2 h-rectangles.

Note that Lemma 1 still holds for the optimal covering if each position can be covered by at most 2 h-rectangles. If two h-rectangles $r_1 = (h_1, x_1^l, x_1^r)$ and $r_2 = (h_2, x_2^l, x_2^r)$ overlap, they must be nested, w.l.o.g., $x_1^l < x_2^l < x_2^r < x_1^r$. We can say that r_2 only overlaps with r_1, otherwise, some position in the plane will be covered by more than two h-rectangles. Therefore, if there are more than one h-rectangles nested in an h-rectangle r, all these h-rectangles except r are disjoint.

Define $N(i, j)$ to be the minimum number of h-rectangles for covering all points between x_i and x_j. Similar to the description in Section 2.1, the optimal covering consists of either two optimal coverings from x_i to x_k and from x_{k+1} to x_j, or an h-rectangle r covering x_i and x_j combined with the optimal covering for those points higher than r. Based on the above analysis, we have the following recurrence formula for the optimal covering of the VSRC problem.

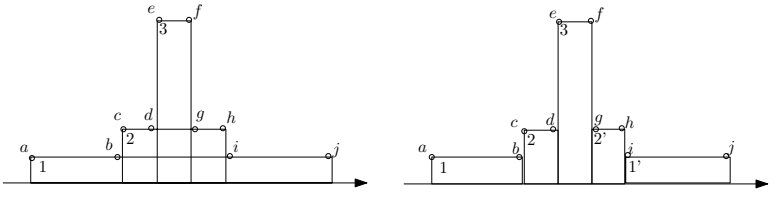


Fig. 6. Tight example of algorithm VSRC2

$$N(i,j)=\begin{cases}1 & \text{if all points can be covered by one h-rectangle}\\ \min\left\{\min_{i\le k\le j}\{N(i,k)+N(k+1,j)\}, N^1(i,j)+1\right\} & \text{otherwise}\end{cases} \quad (2)$$

where $N^1(i,j)$ is the minimal number of disjoint h-rectangles for covering those points whose x-coordinates are between x_i and x_j, and with their heights higher than $h = 1/(x_j - x_i)$. From the above analysis, as the h-rectangles must be disjoint, we can use Algorithm VSRC2 to find $N^1(i,j)$ as VSRC2 would give the optimal covering if the h-rectangles are not allowed to overlap. Thus, $N^1(i,j)$ can be computed in $O(j-i)$ time, and the computation of $N(i,j)$ can be finished in $O(n)$ time. As there are $O(n^2)$ entries of $N(i,j)$ and $N^1(i,j)$, we have the following theorem.

Theorem 4. *Assume $p_i = (x_i, y_i)$ for $1 \le i \le n$ and $x_1 \le x_2 \le ... \le x_n$ and each position can be covered by no more than two h-rectangles, the VSRC problem can be solved in $O(n^3)$ time.*

3 Concluding Remark

We have considered the variable-size rectangle covering problem and proposed several algorithms for finding the optimal covering or approximation covering of some variations of the problem. There are still many covering problems unsolved. In our future research, we will focus on the following directions:

- If the number of points covered by an h-rectangle is bounded and the weight of each point is fractional, it is NP-hard and we can directly use the algorithm from [2] to achieve 3-approximation. But if each point is of unit weight, can we achieve the optimal covering in polynomial time?
- The online version of VSRC problem is a very interesting problem. For one-dimensional case, there are several results on some similar problems [4,3,14,10]. For two-dimensional case, there are many results on covering points by rectangles if the bottom edges of rectangles do not have to be grounded at the base.

References

1. Bao, L., Garia-Luna-Aceves, J.J.: Transmission Scheduling in Ad Hoc Networks with Directional Antennas. In: Proceedings of the 8th International Conference on Mobile computing and networking, pp. 48–58 (2002)
2. Berman, P., Jeong, J., Kasiviswanathan, S., Urgaonkar, B.: Packing to Angles and Sectors. In: Gibbons, P.B., Scheideler, C. (eds.) SPAA 2007, pp. 171–180 (2007)
3. Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental Clustering and Dynamic Information Retriveal. SIAM J. Comput. 33(6), 1417–1440 (2004)
4. Chan, T.M., Zarrabi-Zadeh, H.: A randomize Algorithm for Online Unit Clustering. In: Erlebach, T., Kaklamanis, C. (eds.) WAOA 2006. LNCS, vol. 4368, pp. 121–131. Springer, Heidelberg (2007)
5. Fowler, R.J., Paterson, M.S., Tanimoto, S.L.: Optimal Packing and Covering in the Plane are NP-Complete. Information Processing Letters 12(3), 133–137 (1981)

6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide o the Theory of NP-Completeness. W.H. Freeman, New York (1979)
7. Hochbaum, D.S., Mass, W.: Approximation Scheme for Covering and Packing Problems in Image Processing and VLSI. JACM 32, 130–136 (1985)
8. Karmarkar, N., Karp, R.M.: An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, pp. 312–320 (1982)
9. Roy, S., Hu, Y., Peroulis, D., Li, X.Y.: Minimum-Energy Broadcast Using Practical Directional Antennas in All-Wireless Networks. In: Proc. of the 25th INFORCOM, pp. 1–12 (2006)
10. Seiden, S.: On the Online Bin Packing Problem. JACM 49(5), 640–671 (2002)
11. Spyropoulos, A., Raghavendra, C.: Energy Efficient Communication in Ad Hac Networks Using Directional Antennas. In: Proc. of the 21st INFORCOM, pp. 220–228 (2002)
12. Voipio, V., Vainikainen, P.: Narrowbeam Cylindrical Antenna Array with Sparse Antenna Spacing. In: Proc. of the 48th IEEE Vehicular Technology Conference, vol. 1, pp. 465–469 (1998)
13. Yi, S., Pei, Y., Kalyanaraman, S.: On the Capacity Improvement of Ad Hoc Wireless Networks Using Directional Antennas. In: Proceedings of the 9th International Conference on Mobile computing and networking, pp. 108–116 (2003)
14. Zarrabi-Zadeh, H., Chan, T.M.: An Improved Algorithm for Online Unit Clustering. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 383–393. Springer, Heidelberg (2007)

On-Line Multiple-Strip Packing

Deshi Ye[1,⋆], Xin Han[2], and Guochuan Zhang[1,3]

[1] College of Computer Science, Zhejiang University, Hangzhou 310027, China
yedeshi@zju.edu.cn
[2] Department of Mathematical Informatics, Graduate School of Information and Technology, University of Tokyo, Tokyo 113-8656, Japan
xhan@mist.i.u-tokyo.ac.jp
[3] State Key Lab of CAD & CG, Zhejiang University
zgc@zju.edu.cn

Abstract. We study the multiple-strip packing problem, in which the goal is to pack all the rectangles into m vertical strips of unit widths such that the maximum height among strips used is minimized. A number of on-line algorithms for this problem are proposed, in which the decision of delivering the rectangles to strips as well as packing the rectangles in strips must be done on-line. Both randomized and deterministic on-line algorithms are investigated, and all of them are guaranteed to have constant competitive ratios.

Keywords: Strip packing; approximation algorithms; many-core scheduling.

1 Introduction

We consider packing a set of rectangles into multiple two-dimensional strips of unit width such that no rectangles are intersected with each others and the sides of the rectangles are parallel to the strip sides. Rectangles are not allowed to rotate. The objective is to minimize the maximum height of the strips used to pack a given list of rectangles.

The classical strip packing problem, that uses exactly one strip, is known to be NP-hard [1]. It arises in many real-world applications, e.g., VLSI design and stock cutting problem for packing newspaper commercials. When we extend it to multiple strips, it is not only of interests in the theoretical study, but also has many real applications. For example, in operating systems, multiple-strip packing arises in the computer grid [6] and server consolidation [11]. In the system supporting server consolidation on many-core chip multiprocessors (CMP), multiple server applications are deployed on to virtual machines (VMs). Every virtual machine is allocated several processors and each application might require a number of processors simultaneously. Hence, a virtual machine can be regarded as a strip and server applications can be represented as rectangles whose heights and widths

⋆ Research was supported by National 973 Fundamental Research Project of China (No. 2007CB310900) and NSFC (No. 10601048).

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 155–165, 2009.

are equal to the running time and the required number of processors, respectively. Similarly, in the distributed virtual machines environment, each physical machine can be regarded as a strip and virtual machines are represented as a rectangle. It is quite natural to investigate the packing algorithm by minimizing the maximum height of the strips. This is related to the problem maximizing the throughput, which is commonly used in area of operating systems.

In this paper we study on-line multiple-strip packing. In the *on-line* version, the items (rectangles) appear one by one and the placement decision for the incoming rectangle must be immediately and irrevocability made without any knowledge of subsequence items. Since we have multiple strips, the decision consists of assigning a strip and packing to the strip. In our model both assigning and packing have been done before the next items is seen. If we assign items immediately (on-line) but pack them later (off-line), it becomes semi on-line.

Performance measures. For any input list I and a deterministic approximation (on-line) algorithm A, we denote by *OPT(I)* and *A(I)*, respectively, the maximum height of strips used by an optimal algorithm and the maximum height of strips used by the algorithm A to pack the list I. The approximation (competitive) ration of Algorithm A is defined to be $R_A = \sup_I\{A(I)/OPT(I)\}$.

If algorithm A is a randomized on-line algorithm, then $A(I)$ is random variable. We define the competitive ratio by the expected value of this random variable with oblivious adversary, that is

$$R_A = \sup_I\{E[A(I)]/OPT(I)\}.$$

Related Work. The classical online strip packing problem has been extensively studied. Baker and Schwarz [2] developed the first class of on-line algorithms named shelf algorithms. They showed that next fit shelf algorithm (NFS) has a competitive ratio at most 7.46 and first fit shelf algorithm (FFS) has a competitive ratio at most 6.99. They also gave a lower bound of 2. Hurink and Paulus [10] recently improved the lower bound to 2.43. To the best of our knowledge, the best upper bound by far is 6.6623 [14,9]. There is a lot of work concerning about the asymptotic performance of on-line algorithms [2,4].

A closely related problem to the strip packing is parallel jobs (or multiprocessor tasks) scheduling [5,9,12,15]. Parallel jobs might require to execute on more than one processor at the same time. If we consider that the widths as the resource requirement and the heights as the time, the strip packing is essentially a version of parallel jobs scheduling with additional constraint that a job must be scheduled on consecutively numbered processors. Any algorithms for strip packing can be migrated to parallel job scheduling problem, however vice versa may not. Schwiegelshohn et al. [12] studied the parallel jobs scheduling in grids, where the jobs arrive over time but do not need to be allocated to processors immediately at its submission time. They presented a grid scheduling algorithm that guarantees a competitive ratio at most 5.

The multiple-strip packing was first considered by Zhuk [16]. The author proved that the problem does not admit an approximation algorithm of a ratio

Table 1. Our results on on-line strip packing problem

	Randomized algorithms	Deterministic algorithms
One strip	RLS-NF: UB ≤ 6.58 RLS-FF: UB ≤ 6.1 RLS-RFF: UB ≤ 5.75	NFS UB=7.46 [2] FFS UB=6.99 [2] RFFS UB=6.6623 [14,9]
m identical strips	RLS-NF: UB $\leq 2r + \frac{1+m(r-1)}{m \ln r}$ RLS-FF: UB $\leq 1.7r + \frac{1+m(r-1)}{m \ln r}$ RLS-RFF: UB $\leq \begin{cases} 2 + \frac{(2m-1)r-(m-1)}{m \ln r}, & r \leq \frac{4}{3}; \\ \frac{3}{2}r + \frac{(2m-1)r-(m-1)}{m \ln r}, & r > \frac{4}{3}. \end{cases}$	LS-NF: UB $\leq (\sqrt{3m}+1)^2/m$ LS-FF: UB $\leq (\sqrt{2.7m}+1)^2/m$ LS-RFF: $UB \leq \begin{cases} \frac{1}{m} + \frac{5}{2} + \frac{10}{\sqrt{10m}}, & m \leq 3; \\ \frac{10}{3} + \frac{4}{m}, & 4 \leq m \leq 8; \\ 3 + \frac{2}{\sqrt{m}} + \frac{1}{m}, & m \geq 9. \end{cases}$

Table 2. Some numeric results on on-line strip packing problem

m	R_{LS-NF}	R_{RLS-NF}	R_{LS-FF}	R_{RLS-FF}	R_{LS-RFF}	$R_{RLS-RFF}$
1	7.4641	6.5949	6.9863	6.0937	6.6623	5.7512
2	5.9495	5.4566	5.5238	5.0155	5.2361	4.7158
3	5.3333	4.9741	4.9307	4.5594	4.6591	4.2783
4	4.9821	4.6931	4.5932	4.2939	4.3333	4.0238
5	4.7492	4.5041	4.3697	4.1155	4.1333	3.8539
6	4.5809	4.3661	4.2083	3.9854	4.0000	3.7381
7	4.4522	4.2598	4.0850	3.8851	3.9048	3.6553
8	4.3497	4.1746	3.9869	3.8048	3.8333	3.5932

strictly small than 2 unless $P = NP$ even if there are only two strips. In that paper, a semi-online algorithm was studied, i.e., the jobs arrive on-line and must be distributed to the strips in the on-line manner, but the packing of items in a strip is off-line since the packing is performed only after all the items have been deployed to the strips. The author adopted the Bottom-Left Decreasing algorithm [1] which requires to order the items (in a strip) in the order of decreasing width. It was shown that the competitive ratio of this semi-online algorithm is exactly 10.

Our Contribution. In this paper we study the online problem by designing both randomized and deterministic on-line algorithms. The competitive ratios beat the previous bound 10 that works for the *semi-online* problem [16]. When the number m of strips is arbitrarily large, the algorithms can achieve much better competitive ratios. The detailed competitive ratios are given in Table 1, and some numeric results are illustrated in Table 2, where m is the number of machines, LS-NF, LS-FF and LS-RFF are our deterministic algorithms and RLS-NF, RLS-FF and RLS-RFF are our randomized algorithms.

At the end, we give some remarks on the off-line version. We present a $(2+\varepsilon)$-approximation algorithm for any small number of $\varepsilon > 0$, and thus it is nearly optimal.

2 Randomized Online Algorithms

Before presenting our algorithms, some global notations are introduced here. We are given m vertical strips of width 1. The rectangles are characterized by their widths and heights. Denote by $T = \{T_1, \ldots, T_n\}$ the list of rectangles. Let $w(T_j)) \leq 1$ and $h(T_j)$ be the width and the height of rectangle T_j, respectively.

To get an online algorithm, we have to answer two questions upon arrival of a new item: i) which strip it goes to and ii) where it is placed inside the strip? The basic idea is: using List Scheduling(LS) [7] to select a strip for the incoming item and using Shelf Algorithm [2] to pack the item into the selected strip.

In a shelf packing, the strip is gradually divided level by level. A shelf is the rectangular bin formed by two consecutive levels and the distance of the two levels gives the height of the shelf. Our randomized or deterministic algorithms consist of two steps. At the first step, if the incoming item can be packed into some existing shelf among all strips, then pack the item in such a shelf with a shelf packing algorithm. Else go to the second step, open a new shelf for the item in the strip with the least load. The load of a strip is defined to be the current height of the strip used.

We will employ several packing algorithms in the first step. The next fit shelf algorithm [2] packs each item as far to the left as possible on the current active shelf (the first on the top) that has the required height until the next item is too wide to fit. A new such shelf is created and becomes the active shelf of that height once there is no room. In the first-fit approach [2], the item is packed to the lowermost shelf that it can fit. If there is no shelf of the required height, or none of the appropriate shelves have a sufficient room, then a new shelf of that height is created. Moreover, we will also use the revised shelf algorithm [14].

On the other hand, there are lots of strategies to assign the newly created shelf to a strip. Actually, the second step can be regarded as the classical scheduling problem. The height of a shelf can be regarded as processing time and strips are regarded as machines. In this paper, we use list scheduling(LS) [7] in the second step to assign the shelf to a strip. Algorithm LS always schedules the current job as early as possible.

2.1 Randomized LS-NF Algorithm

The idea of the algorithm is to partition items to a shelf and use list scheduling algorithm to pack the shelf. The height of a shelf is classified with a random variable, however, it is not too much random because it is fixed at the first step, and then only deterministic algorithm is applied.

Algorithm RLS-NF

1: Choose a uniform random real number $\varepsilon \in [0, 1)$ and a constant real number $r > 1$, and let $d = r^{\varepsilon}$.
2: For an incoming item with height p_j, find an integer k such that $d \cdot r^k < p_j \leq d \cdot r^{k+1}$. Then pack the item by the NF algorithm to a shelf with height $d \cdot r^{k+1}$ if there exists available space, otherwise create a new shelf with height $d \cdot r^{k+1}$.

3: Finally, the newly created shelf is assigned to a strip with the smallest load. The load of a strip is defined to be the height of the strip used.

Remark. The first line of RLS-NF gives the randomization. And the algorithm is *oblivious* randomized algorithm, i.e., the optimal algorithm does not know our randomized choice. When the algorithm NF is replaced by FF or RFF, then RLS-FF and RLS-RFF follow.

Consider the final packing, it consists of shelves. The shelves with the same height $d \cdot r^k$ are grouped to be class k. For each k, the last one shelf is called *sparse*, and the remaining shelves are *dense*. Let H_S and H_D be the total height of sparse shelves and dense shelves over all k. Denote by h the maximum height among all the shelves and by μ the maximum height among all the rectangles, then $\mu = \max_j h(T_j) \leq Opt$, where T_j is the j-th rectangle and $h(T_j)$ is its height. Note that the maximum difference of two strips is at most h. Then we have the following lemmas.

Lemma 2.1. *Let Alg be the value by online algorithm RLS-NF. Then* $Alg \leq \frac{H_S+H_D-h}{m} + h$.

Proof. Let h_i be the height of the i-th strip, where $1 \leq i \leq m$. So, we have

$$Alg = \max_i\{h_i\}.$$

By the List Scheduling algorithm, we have $|h_i - h_j| \leq h$ for any $1 \leq i, j \leq m$, where h is the maximum height over all the shelves. So, we have

$$\sum_{i=1}^{m} h_i + (m-1)h \geq m * \max_i\{h_i\} = m * Alg.$$

Since $\sum_{i=1}^{m} h_i = H_S + H_D$, this lemma holds. □

Remark. For algorithms RLS-FF and RLS-RFF, we also have the similar result.

Lemma 2.2. *The expectation of the random variable* h/μ *is* $\frac{r-1}{\ln r}$.

Proof. Similar as the randomized technical for online bidding problem [3], we know that $h/r < \mu \leq h$ and h/μ is a random variable of r^X, where X is uniformly chosen from $[0, 1)$.

The expectation of $E[h/\mu]$ can be computed directly from its definition.

$$E[h/\mu] = \int_0^1 r^x dx = \int_1^r y\frac{1}{y \ln r} dy = \frac{r-1}{\ln r}. \tag{1}$$

□

Theorem 2.3. *For any* $r > 1$, *the competitive ratio of RLS-NF algorithm is*

$$R_{RLS-NF} = \min_{r>1}\{2r + \frac{1 + m(r-1)}{m \ln r}\}. \tag{2}$$

Proof. Denote by Alg and Opt the value of algorithm RLS-NF and the value of optimal off-line algorithm, respectively.

By the NF algorithm, we know that in a density shelf, averagely the width is at least half occupied. By the shelf algorithm, we know in any shelf, the height is at least $1/r$ used, so we have

$$m * Opt \geq 1/2 * 1/r * H_D, \tag{3}$$

i.e., the total area occupied by the optimal algorithm is not smaller than the total area of all the rectangles packed in the density shelves H_D.

Since the height of each sparse shelf differs at least r times, we have

$$H_S \leq h * \sum_{i=0} r^{-i} \leq \frac{h}{(1-1/r)}.$$

By Lemma 2.1,

$$\begin{aligned} Alg &\leq \frac{H_S + H_D - h}{m} + h && (4)\\ &\leq H_D/m + H_S/m + h * (m-1)/m && (5)\\ &\leq 2r * Opt + \Big(\frac{r}{m*(r-1)} + \frac{(m-1)}{m}\Big) * h && (6)\\ &\leq 2r * Opt + \Big(\frac{r}{m*(r-1)} + \frac{(m-1)}{m}\Big) * h/\mu * Opt && (7) \end{aligned}$$

Then we get that

$$R_{RLS-NF} = \frac{E[Alg]}{Opt} \leq 2r + \Big(\frac{r}{m*(r-1)} + \frac{(m-1)}{m}\Big) * E[h/\mu].$$

By Lemma 2.2, the competitive ratio is

$$\frac{E[Alg]}{Opt} \leq 2r + \frac{1 + m(r-1)}{m \ln r}.$$

Let $c(m,r) = 2r + \frac{1+m(r-1)}{m \ln r}$. For each specific value of m, we need to calculate r so that $c(m,r)$ reaches its minimum, which can be computed by its derivative.

$$\frac{d(c(m,r))}{dr} = 2 + \frac{m^2 \ln r - (m^2 r + m - m^2)/r}{m^2 \ln^2 r} \tag{8}$$

It is not easy to get a explicit relationship between variables m and r since the derivative itself is not a simple function. However, by aids of computer programming, we get the following table.

$m =$	1	2	3	4	5	6	7	8
$r =$	1.6487	1.4550	1.3702	1.3199	1.2857	1.2605	1.2410	1.2254
$R_{NLS-NF} \leq$	6.5949	5.4567	4.9742	4.6931	4.5042	4.3662	4.2598	4.1747

□

2.2 Randomized LS-FF (RLS-FF)

In this section, we consider the randomized algorithm RLS-FF, which is gotten by replacing the NF algorithm by the FF algorithm in RLS-NF, i.e., we use first fit algorithm to packing rectangles instead of next fit algorithm.

To analyze the algorithm RLS-FF, we use the online algorithm of packing rectangles into one strip as a bridge. The details are the followings. Let FF and OPT be the costs of algorithm first fit shelf and an optimal off-line algorithm to pack all the items into one strip, respectively. Then we have the following lemma from [2].

Lemma 2.4. $FF/r - h/(r-1) \leq 1.7OPT$.

Theorem 2.5. *For any $r > 1$, the competitive ratio of RLS-NF algorithm is*

$$R_{RLS-FF} \leq \min_{r>1} \{1.7r + \frac{1 + m(r-1)}{m \ln r}\}. \tag{9}$$

Proof. Let FF^* and OPT^* be the costs of algorithm RLS-FF and an optimal off-line algorithm to pack rectangles into m strips, respectively. By the definitions, we have $OPT \leq m \cdot OPT^*$. Again similar with the proof in Lemma 2.1, we have $FF \geq m * FF^* - (m-1)h$, where h is the maximum height over all the shelves. Hence, we have

$$1.7m \cdot OPT^* \geq m \cdot FF^*/r - (m-1)h/r - h/(r-1),$$

and then

$$FF^* \leq 1.7r \cdot OPT^* + (m-1)h/m + r \cdot h/(m(r-1)). \tag{10}$$

Since $h \leq r \cdot \mu \leq r \cdot OPT^*$ and by Lemma 2.2, we have

$$E[FF^*]/OPT^* \leq 1.7r + (\frac{m-1}{m} + \frac{r}{m(r-1)}) \cdot E[h/\mu] \tag{11}$$

$$\leq 1.7r + \frac{1 + m(r-1)}{m \ln r}. \tag{12}$$

Using the same technique in Theorem 2.3, we get the following table.

$m =$	1	2	3	4	5	6	7	8
$r =$	1.6942	1.4863	1.3957	1.3418	1.3051	1.2783	1.2574	1.2407
$R_{NLS-FF} \leq$	6.0937	5.0156	4.5594	4.2930	4.1156	3.9854	3.8851	3.8048

□

2.3 Randomized LS-RFF Algorithm (RLS-RFF)

In this section, we give a randomized version of the revised shelf algorithm in [14]. The detail of the RLS-RFF algorithm is given as follows.

Algorithm RLS-RFF

1: **if** the incoming item is a big item, i.e., its width is larger than $1/2$ **then**
2: this item is packed to a new shelf with the same height as the item. This shelf is then assigned to a strip with the minimum load.
3: **end if**
4: **if** the incoming item is a small item, i.e., its width is at most $1/2$ **then**
5: call the algorithm Randomized LS-FF (RLS-FF).
6: **end if**

Denote by H_F the height of shelves that only contain big items. Again, let H_S and H_D be the height of sparse shelves and dense shelves.

Lemma 2.6. *[14] The total area of all the rectangles is at least* $\frac{H_F}{2} + \frac{2}{3} \cdot \frac{1}{r} \cdot H_D$, *i.e., the used efficiencies of shelves in* H_F *and* H_D *are 1/2 and* $2/3r$, *respectively.*

Theorem 2.7. *The competitive ratio of algorithm RLS-RFF is*

$$E[Alg]/Opt \leq \begin{cases} 2 + \frac{1+r(m-1)}{m \ln r}, & 1 < r \leq 4/3; \\ \frac{3}{2}r + \frac{1+r(m-1)}{m \ln r}, & r > 4/3. \end{cases}$$

Proof. The proof can be done by adopting Lemma 2.6 and similar idea as Theorem 2.3. However the detailed analysis is omitted due to space limited. Also by the same technical used in Theorem 2.3, we get the following table.

$m =$	1	2	3	4	5	6	7	8
$r =$	1.7305	1.5114	1.4159	1.3592	1.3334	1.3334	1.3333	1.3333
$R_{NLS-FF} \leq$	5.7513	4.7158	4.2783	4.0239	3.8540	3.7381	3.6553	3.5933

□

3 Deterministic Online Algorithms

In this section, we derandomize the algorithms in the last section and get the corresponding deterministic algorithms, called LS-NF, LS-FF, LS-RFF respectively. The approach to derandomize is quite simple: we let $\varepsilon = 0$ instead of a random variable in the first step of above random algorithms, then all the algorithms become the deterministic algorithms. Next we analyze the deterministic algorithms and show that the corresponding competitive ratio is worse than the ratio of the randomized algorithm.

Theorem 3.1. *For any m strips, the competitive ratio of LS-NF algorithm is* $3r + r/(m(r-1))$, *which reaches to the minimum* $(3m + 2\sqrt{3m} + 1)/m$ *when* $r = 1 + 1/\sqrt{3m}$.

Proof. The proof is very similar as Theorem 2.3, since only the variable h/μ is a random variable. To obtain a deterministic analysis, we replace with the item h/μ by the parameter r in the equation (7), since $h/\mu \leq r$. Therefore, the competitive ratio of LS-NF is

$$R_{LS-NF} = \frac{Alg}{Opt} \leq 2r + \left(\frac{r}{m*(r-1)} + \frac{(m-1)}{m}\right) * r \qquad (13)$$
$$= 3r + \frac{r}{m(r-1)}$$

From the above inequality, the minimum value of R_{LS-NF} is $(3m + 2\sqrt{3m} + 1)/m$ by setting $r = 1 + 1/\sqrt{3m}$. □

By an analogy analysis, we can get the deterministic version of algorithms LS-FF and LS-RFF.

Corollary 3.2. *The competitive ratio of deterministic algorithm LS-FF is* $2.7r + r/(m(r-1))$*, which reaches to the minimum* $(2.7m + 2\sqrt{2.7m} + 1)/m$ *when* $r = 1 + 1/\sqrt{2.7m}$.

Corollary 3.3. *The competitive ratio of deterministic algorithm LS-RFF is*

$$R_{LS-RFF} \leq \begin{cases} 2 + r + \frac{r}{m(r-1)}, & 1 < r \leq \frac{4}{3}; \\ \frac{5r}{2} + \frac{r}{m(r-1)}, & r > \frac{4}{3}. \end{cases}$$

Then we have

$$R_{LS-RFF} = \begin{cases} \frac{1}{m} + 2.5 + \frac{10}{\sqrt{10m}}, & m \leq 3, \quad r = 1 + \frac{2}{\sqrt{10m}}; \\ \frac{10}{3} + \frac{4}{m}, & 4 \leq m \leq 8, \quad r = \frac{4}{3}; \\ 3 + \frac{2}{\sqrt{m}} + \frac{1}{m}, & m \geq 9, \quad r = 1 + \frac{1}{\sqrt{m}}. \end{cases}$$

Lemma 3.4. *Let A be one of algorithms LS-NF, LS-FF and LS-RFF and* R_A *be one of algorithms RLS-NF, RLS-FF and RLS-RFF. Then we have* $R_{RA} < R_A$.

Proof. To compare the ratios in Theorems 2.3, 2.5 and 2.7 with the ratios in Theorem 3.1, Corollaries 3.2 and 3.3 respectively, we find if $\frac{r-1}{\ln r} < r$ for all $r > 1$, then this lemma holds.

It is not difficult to get this

$$r \ln r - r + 1 > 0,$$

by setting $r = e^x$, then applying the inequality $e^x > x + 1$ for all $x > 0$. □

4 Off-Line Version

In [16], it was shown that there is no hope to get an approximation algorithm within a factor of 2 even with two strips. In this section, we present $(2 + \varepsilon)$-approximation algorithm for the case with equal width of strips.

We deal with off-line problems into two steps, first we solve a scheduling problem to deploy the rectangles to strips, then apply Steinberg's algorithm [13] to pack the rectangles in each strip. The detailed algorithm is given as follows.

Algorithm WPS

1: For every item T_j, denote $S_j = w(T_j) * h(T_j)$ to be the work of the item. A new instance I of classical scheduling problem is established, where S_j is regarded as the processing time of the j-th job and m is the number of identical machines.
2: Apply the PTAS algorithm [8] to the instance I. Denote B_i to be the set of jobs assigned to the machine i and l_i to be the completion time of machine i after this procedure, where $1 \leq i \leq m$.
3: Apply Steinberg's algorithm [13] for the set B_i by setting the height of a bin to be $2l_i$ and width to be w_i.

Theorem 4.1. *For any $\varepsilon > 0$, the algorithm WPS is $(2+\varepsilon)$-approximation. Its running time is $O((n/\varepsilon)^{1/\varepsilon^2})$.*

Proof. Roughly, the proof consists of a lemma in [13] that ensures at most double space used for a specific set of rectangles, and the fact that the PTAS algorithm [8] evenly distributes the items to the strips. The detailed proof will be given in the full version. □

References

1. Baker, B.S., Coffman, E.G., Rivest, R.L.: Orthogonal Packings in Two Dimensions. SIAM J. Comput. 9, 846–855 (1980)
2. Baker, B.S., Schwartz, J.S.: Shelf Algorithms for Two-Dimensional Packing Problems. SIAM J. Comput. 12, 508–525 (1983)
3. Chrobak, M., Kenyon, C., Noga, J., Young, N.: Incremental Medians via Online Bidding. Algorithmica 50(4), 455–478 (2008)
4. Csirik, J., Woeginger, G.J.: Shelf Algorithms for On-Line Strip Packing. Inform. Process. Lett. 63, 171–175 (1997)
5. Du, J., Joseph, Y.T.L.: Complexity of Scheduling Parallel Task Systems. SIAM J. Discrete Math. 2, 473–487 (1989)
6. Foster, I., Kesselman, C.: The Grid: Blueprint for a Future Computing Infrastructure (1999)
7. Graham, R.L.: Bounds for Certain Multiprocessing Anomalies. Bell System Technical J. 45, 1563–1581 (1966)
8. Hochbaum, D.S., Shmoys, D.B.: Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results. J. ACM 34, 144–162 (1987)
9. Hurink, J.L., Paulus, J.J.: Online Algorithm for Parallel Job Scheduling and Strip Packing. In: Proc. 5th International Workshop in Approximation and Online Algorithms, pp. 67–74 (2007)
10. Hurink, J.L., Paulus, J.J.: Online Scheduling of Parallel Jobs on Two Machines is 2-Competitive. Oper. Res. Lett. 36(1), 51–56 (2008)
11. Marty, M.R., Hill, M.D.: Virtual Hierarchies to Support Server Consolidation. In: Proceedings of the 34th Annual International Conference on Computer Architecture, pp. 46–56 (2007)
12. Schwiegelshohn, U., Tchernykh, A., Yahyapour, R.: Online Scheduling in Grids. In: IEEE International Symposium on Parallel and Distributed Processing (IPDPS), pp. 1–10 (2008)

13. Steinberg, A.: A Strip-Packing Algorithm with Absolute Performance Bound 2. SIAM J. Comput. 26, 401–409 (1997)
14. Ye, D., Han, X., Zhang, G.: A Note on Online Strip Packing. J. Comb. Optim. 17(4), 417–423 (2009)
15. Ye, D., Zhang, G.: On-Line Scheduling of Parallel Jobs in a List. J. Sched. 10(6), 407–413 (2007)
16. Zhuk, S.: Approximate Algorithms to Pack Rectangles into Several Strips. Discrete Mathematics and Applications 16(1), 73–85 (2006)

A Cost-Sharing Method for the Soft-Capacitated Economic Lot-Sizing Game*

Ruichun Yang[1], Zhen Wang[2,**], and Dachuan Xu[2]

[1] College of Science, Hebei North University
Zhangjiakou 075000, China
yangruichun@emails.bjut.edu.cn
[2] Department of Applied Mathematics, Beijing University of Technology
Beijing 100124, China
{zw,xudc}@bjut.edu.cn

Abstract. We present a cross-monotonic competitive cost-sharing method with provable cost-recovery ratio for the soft-capacitated economic lot-sizing game, an extension of the basic economic lot-sizing game, under the weak triangle inequality assumption.

Keywords: Economic lot-sizing game; cost-sharing method; cross-monotonic; approximate cost recovery.

1 Introduction

The basic economic lot-sizing model was proposed by [9,12], which does not allow backlogging. In the general economic lot-sizing model [16], a retailer is facing a demand for a single product that occurs during each period of a consecutive time periods numbered through 1 to T. The demand of a given time period can be satisfied by orders at either that, previous, or later periods; that is, inventory holding and back-logging are allowed. The objective is to decide the order quantity at each period such that all demands are fulfilled and the total cost of ordering, inventory holding, and backlogging is minimized. As an extension of the above problem, the capacitated version consider the situation where the amount of production is limited by a given capacity for each time period [1].

In this paper, we consider the *soft-capacitated economic lot-sizing problem* (SCELSP). Let us consider T time periods. For each $t : 1 \leq t \leq T$, we define the following notations:

d_t: the amount of demand at period t;
x_t: the order quantity at period t, which is called an ordering period if $x_t > 0$;
s_t: the amount of (non-negative) inventory at the end of period t;
r_t: the amount of backlogged demand at period t;
h_t: the unit cost of holding inventory at the end of period t;

* Research supported by NSF of China (Grant No. 10871014 and 60773185) and Program for Beijing Excellent Talents.
** Corresponding author.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 166–173, 2009.

g_t: the unit cost of having backlogged demand at period t;
u_t: the capacity associated with period t;
$p_t(x_t)$: the cost of ordering x_t units at period t.

We consider the SCELSP with setup cost. In this problem, the ordering cost function has a fixed cost component and a variable cost component, i.e., $p_t(x_t) = f_t \lceil x_t/u_t \rceil + c_t x_t$ where f_t is the setup cost and c_t is a given constant at period t.

It is well-known that the SCELSP can be formulated as follows

$$\begin{aligned} P(d) := \min & \sum_{t=1}^{T} \{p_t(x_t) + h_t s_t + g_t r_t\} \\ \text{s.t.} \quad & x_1 = d_1 + s_1 - r_1, \\ & x_t + s_{t-1} - r_{t-1} = d_t + s_t - r_t, \forall t = 2, ..., T, \\ & x_t \geq 0, s_t \geq 0, r_t \geq 0, \qquad \forall t = 1, 2, ..., T, \end{aligned} \tag{1}$$

where $d = (d_1, d_2, ..., d_T)$ is a vector in R^T.

Consider the following soft-capacitated economic lot-sizing situation where multiple retailers sell the same product, they order the product from a single manufacturer. In a decentralized system, each retailer would solve an SCELSP. However, by exploiting economies of scale, the retailers may find that it is beneficial to form coalitions and place joint orders. A prime question is how to allocate the cost or profit in such a way that is advantageous for all the retailers, i.e., no retailer(s) gain more by deviating from the cooperation. This naturally gives rise to a cooperative game with the players being the retailers, which is referred to as the *soft-capacitated economic lot-sizing game* (SCELSG).

Formally, consider a set of retailers $N = \{1, 2, ..., n\}$ that sell the same product, all retailers buy the product from the same manufacturer. For each $l \in N$, let $d^l = (d_1^l, d_2^l, ..., d_T^l)$ be the demand vector of player (retailer) l, where d_t^l is the known demand of retailer l in time period $t : 1 \leq t \leq T$. For any given subset of players $S \subseteq N$, let d^S be the demand vector of S, i.e., $d^S = (d_1^S, d_2^S, ..., d_T^S)$. We assume that the holding and backlogging costs of each retailer are fixed in a single period.

The retailers can place orders individually. They can also cooperate by placing joint orders and keeping inventory at one warehouse which will lead to costs reduction. In this setting, we define the SCELSG (N, V), where the grand coalition is the set of retailers N and the characteristic cost function $V(S)$ $(S \subseteq N)$ is defined by $V(S) = P(d^S)$. The objective is to design a cost-sharing method that allocates the total cost to the different players, that is, computes the cost share η_l for each $l \in S$.

A cost-sharing method is cross-monotonic, if $\eta(S, l) \geq \eta(S', l)$ for all $S \subseteq S'$. A cost-sharing method is competitive, if $\sum_{l \in S} \eta(S, l) \leq V(S)$. A cost-sharing method is α-approximate cost recovering, if $\sum_{l \in S} \eta(S, l) \geq V(S)/\alpha$, where $\alpha \geq 1$.

We are interested in cost-sharing methods that would motivate the players to cooperate by revealing their true value. To obtain fair and group-strategyproof mechanisms, it is sufficient to focus on cross-monotonicity, competitiveness, cost recovery (cf. [10]).

The SCELSP can be reformulated as a soft-capacitated facility location problem. For the facility location problem and the corresponding game, we refer to [4,5,8,10,13,15,17,20] and references therein.

Xu and Yang [14] give the first cost-sharing method for the *economic lot-sizing game* (ELSG). Following the approaches of [6,7,10,14], we first transform the SCELSP to a special *economic lot-sizing problem* (ELSP), then call the cost-sharing method of the ELSG to construct the cost shares, resulting in cross-monotonicity and competitiveness naturally. In order to get the overall approximation cost recovery, one needs to balance carefully the transformation between the SCELSP and the ELSP. We construct an integral solution of the SCELSP based on the ELSP. This integral solution will reveal the approximate cost recovery.

Finally, the ELSG has also been investigated under the concept of *core* which is an allocation that ensures that no subset of the players is charged more than the true cost of the subset. When backlogging is allowed and the ordering cost is a general concave function, Chen and Zhang [2] showed that the core of the ELSG is always nonempty. For the other inventory games, we refer to [3,18,19] and references therein.

The organization of our paper is as follows. In the next section, we present a cost-sharing method for the SCELSG. The analysis will be given in Section 3. Some discussions are presented in Section 4.

2 Cost-Sharing Method

Let q_{kt} be the cost of satisfying one unit demand at period t by ordering at period k, i.e.,

$$q_{kt} = c_k + \sum_{i=k}^{t-1} h_i \text{ if } k \le t, \text{ and } q_{kt} = c_k + \sum_{i=t}^{k-1} g_i \text{ if } k > t.$$

It is well-known that the ELSP can be formulated as a facility location problem (cf. [11]). So we can give the following mixed integer program of the SCELSP

$$\begin{aligned} OPT \equiv \min \ & \sum_{k=1}^{T} f_k y_k + \sum_{k=1}^{T}\sum_{t=1}^{T} d_t^S q_{kt} x_{kt} \\ \text{s.t. } & d_t^S \sum_{k=1}^{T} x_{kt} \ge d_t^S, && \forall t, \\ & x_{kt} \le y_k, && \forall k, t, \\ & \sum_{t=1}^{T} d_t^S x_{kt} \le u_k y_k, && \forall k, \\ & x_{kt} \ge 0, && \forall k, t, \\ & y_k \in Z^+, && \forall k. \end{aligned} \tag{2}$$

In the above program, y_k is an integral variable denoting the ordering times of period k, and x_{kt} is the fraction of the demand at period t that is satisfied

by inventory ordered at period k. The first constraint states that the demand of each period must be satisfied. The second constraint ensures that if the fraction of the demand at period t is satisfied by a possible ordering period k then k must be an ordering period, and the third constraint says that if period k orders y_k times, it can serve at most $u_k y_k$ amount of demand.

The linear programming relaxation of (2) is:

$$\begin{aligned} P_1 \equiv \min \quad & \sum_{k=1}^{T} f_k y_k + \sum_{k=1}^{T}\sum_{t=1}^{T} d_t^S q_{kt} x_{kt} \\ \text{s.t. } & d_t^S \sum_{k=1}^{T} x_{kt} \geq d_t^S, && \forall t, \\ & x_{kt} \leq y_k, && \forall k, t, \\ & \sum_{t=1}^{T} d_t^S x_{kt} \leq u_k y_k, && \forall k, \\ & x_{kt}, y_k \geq 0, && \forall k, t. \end{aligned} \tag{3}$$

The dual program of (3) is:

$$\begin{aligned} D_1 \equiv \max \quad & \sum_{t=1}^{T} d_t^S \alpha_t \\ \text{s.t. } & d_t^S \alpha_t - \beta_{kt} - d_t^S \gamma_k \leq d_t^S p_{kt}, && \forall k, t, \\ & u_k \gamma_k + \sum_{t=1}^{T} \beta_{kt} \leq f_k, && \forall k, \\ & \alpha_t, \gamma_k, \beta_{kt} \geq 0, && \forall k, t. \end{aligned} \tag{4}$$

For each possible ordering period k, let us fix

$$\gamma_k = \frac{f_k}{2u_k}.$$

Then we eliminate variables γ_k from (4):

$$\begin{aligned} D_2 \equiv \max \quad & \sum_{t=1}^{T} d_t^S \alpha_t \\ \text{s.t. } & d_t^S \alpha_t - \beta_{kt} \leq d_t^S (p_{kt} + \frac{f_k}{2u_k}), && \forall k, t, \\ & \sum_{t=1}^{T} \beta_{kt} \leq \frac{f_k}{2}, && \forall k, \\ & \alpha_t, \beta_{kt} \geq 0, && \forall k, t. \end{aligned} \tag{5}$$

One can verify that (5) is exactly the dual of an ELSP. The primal program of (5) is:

$$\begin{aligned} P_2 \equiv \min \quad & \sum_{k=1}^{T} \frac{f_k}{2} Y_k + \sum_{k=1}^{T}\sum_{t=1}^{T} d_t^S (p_{kt} + \frac{f_k}{2u_k}) X_{kt} \\ \text{s.t. } & d_t^S \sum_{t=1}^{T} X_{kt} \geq d_t^S, && \forall t, \\ & X_{kt} \leq Y_k, && \forall k, t, \\ & X_{kt}, Y_k \geq 0, && \forall k, t. \end{aligned} \tag{6}$$

Denote $Q_{kt} \equiv q_{kt} + \dfrac{f_k}{2u_k}$ and $F_k \equiv \dfrac{f_k}{2}$. Then (6) can be transformed into the following linear program

$$\begin{aligned} P_2 = \min \ & \sum_{k=1}^{T} F_k Y_k + \sum_{k=1}^{T} \sum_{t=1}^{T} d_t^S Q_{kt} X_{kt}, \\ \text{s.t. } & d_t^S \sum_{k=1}^{T} X_{kt} \geq d_t^S, && \forall t, \\ & X_{kt} \leq Y_k, && \forall k, t, \\ & X_{kt}, Y_k \geq 0, && \forall k, t. \end{aligned} \tag{7}$$

Let $\bar{T} = \{1, 2, ..., T\}$. For any $k, t \in \bar{T}$, we always use k as the possible ordering period and t as the time period. For any $k, k' \in \bar{T}$, we define

$$Q_{kk'} \equiv \min_{t:t\in\bar{T}} \{Q_{kt} + Q_{k't}\}.$$

Now we are ready to describe the algorithm as follows.

Algorithm 2.1

Step 1 (the ghost process). We introduce the notation of *time* $\tilde{t}$, advancing from 0 to ∞. For every period $t \in \bar{T}$, the ghost of t is a ball centered at t with radius $d_t^S \tilde{t}$.

The period of $t \in \bar{T}$ *touches* a possible ordering period at time $\tilde{t}$ if $Q_{kt} \leq \tilde{t}$. All ghost that touch a possible ordering period start filling the setup cost of the possible ordering period. The contribution of the ghost t at time $\tilde{t}$ towards filling the setup cost of k is $\xi_{kt} = \max\{0, d_t^S \tilde{t} - d_t^S Q_{kt}\}$. Denote $\theta_{kt} = \max\{0, \tilde{t} - Q_{kt}\}$. The possible ordering period $k \in \bar{T}$ is said to be *fully paid* at some time $\tilde{t}(k)$ if $\sum_{t\in\bar{T}} \xi_{kt} = F_k$. Let R_k be the set of periods that contributed towards filling the setup cost of k, i.e., $R_k = \{t : \xi_{kt} > 0\}$, hence $F_k = \sum_{t\in R_k} (d_t^S \tilde{t}(k) - d_t^S Q_{kt})$.

Step 2 (the cost shares). Let $\alpha_t = \min\{\min_{k:t\in R_k} \tilde{t}(k), \min_{k:t\notin R_k} Q_{kt}\}$. The cost share of each player $l \in S$ is $\eta_l = \sum_{t=1}^{T} d_t^l \alpha_t$.

Step 3 (constructing ordering periods of (7)). Sort all the possible ordering periods in a nondecreasing order of the fully paid time $\tilde{t}(k)(k \in \bar{T})$. According to this order, the period k is the ordering period if and only if there is no already ordering period $k \in \bar{T}$ such that $Q_{kk'} \leq 2\tilde{t}(k)$. Suppose that $k \in \bar{T}$ is an ordering period, we assign the demand of periods in R_k is satisfied by k. The demand of remaining periods will be satisfied by the closest ordering period. Denote the corresponding integral solution of (7) as $(\bar{X}, \bar{Y})$.

Step 4 (constructing an integral solution of (2)). Set $\bar{x}_{kt} := \bar{X}_{kt}$ and $\bar{y}_k := \left\lceil \sum_{t=1}^{T} \bar{d}_t^S X_{kt}/u_k \right\rceil$.

3 Analysis

By adding more players, the setup cost of each possible ordering period $k \in \bar{T}$ will be fully paid more quickly, and each $\theta_{kt}, (k \in \bar{T}, t \in \bar{T})$ will not increase. Combining the definition of $\eta_l = \sum_{t=1}^{T} d_t^l \alpha_t$, we have the following lemmas.

Lemma 1. *The cost share generated by Algorithm 2.1 is cross-monotonic.*

Lemma 2. (**Competitiveness**) *Every solution to the SCELSP has cost at least* $\sum_{t=1}^{T} d_t^S \alpha_t$.

Proof. It is easy to see that the sum of $d_t^S \alpha_t$ can not exceed the cost of any solution of (7). Noting that

$$OPT \geq P_1 = D_1 \geq D_2 = P_2 \geq \sum_{t=1}^{T} d_t^S \alpha_t,$$

the lemma is concluded.

Assumption 3.1. *For any $t \in \bar{T}$, we assume that $a_1 h_t \leq g_t \leq a_2 h_t$, where a_1, a_2 are constants, $0 < a_1 \leq a_2$.*

Let $b := \max\{1/a_1, a_2\}$. Suppose that Assumption 3.1 holds throughout the remainder of our paper. For any possible ordering period k and k', any time period t, it is easy to verify that Q_{kt} satisfies the so-called *weak triangle inequality* (cf. [14]): $Q_{kt} \leq b(Q_{k't} + Q_{kk'})$. From Step 3 of Algorithm 2.1 and the definition of $Q_{kk'}$, we have

Lemma 3. *If k, k' are both ordering periods, then $R_k \cap R_{k'} = \emptyset$.*

Lemma 4. *We have*

$$\sum_{k=1}^{T} F_k \bar{Y}_k + \sum_{k=1}^{T} \sum_{t=1}^{T} d_t^S Q_{kt} \bar{X}_{kt} \leq 3b \sum_{t=1}^{T} d_t^S \alpha_t. \tag{8}$$

Proof. Let us consider the following two possibilities.

Case 1. Let $k \in \bar{T}$ be the ordering period, and let $t \in R_k$. Then $3b\alpha_t \geq \tilde{t}(k)$. Otherwise, for contradiction we assume that $3b\alpha_t < \tilde{t}(k)$. Let k' is the first fully paid period that t touched, i.e., $Q_{k't} \leq \alpha_t$ and $\tilde{t}(k') \leq \alpha_t$. We consider two subcases.

Case 1.1 If k' is an ordering period, then

$$Q_{kk'} \leq Q_{kt} + Q_{k't} \leq \tilde{t}(k) + \alpha_t < 2\tilde{t}(k).$$

Case 1.2 If k' is not an ordering period, there exists an ordering period $k'' \in \bar{T}$, such that $Q_{k'k''} \leq 2\tilde{t}(k') \leq 2\alpha_t$, note that $k \neq k''$, since $\tilde{t}(k'') \leq \tilde{t}(k') \leq \alpha_t < \tilde{t}(k)$, we have

$$\begin{aligned} Q_{kk''} &\leq Q_{kt} + Q_{k''t} \\ &\leq Q_{kt} + b(Q_{k't} + Q_{k''k'}) \\ &\leq \tilde{t}(k) + b(\alpha_t + 2\tilde{t}(k')) \\ &< 2\tilde{t}(k). \end{aligned} \tag{9}$$

Both of the above two cases contradict to Step 3 of Algorithm 2.1.

Case 2. Suppose that $t \in \bar{T}$ does not belong to any neighborhood R_k for an ordering period $k \in \bar{T}$, and t is assigned to the closest ordering period k. In the similar way, we can show that $3b\alpha_t \geq Q_{kt}$.

Summarizing all the above two cases, we get the desired conclusion (8). □

Theorem 1. *The cost of the solution constructed is at most* $6b\sum_{t=1}^{T}\alpha_t d_t^S$, *and hence Algorithm 2.1 is a* $6b$*-approximate cross-monotonic cost-sharing method for the SCELSG.*

Proof. Recall the constructions of $(\bar{X}, \bar{Y})$ and $(\bar{x}, \bar{y})$ at Algorithm 2.1. It is easy to see that

$$\bar{y}_k \leq \frac{\sum_{t=1}^{T} d_t^S \bar{X}_{kt}}{u_k} + \bar{Y}_k. \tag{10}$$

It follows from (10) and Lemma 4 that

$$\begin{aligned}\sum_{k=1}^{T}\frac{f_k}{2}\bar{y}_k + \sum_{k=1}^{T}\sum_{t=1}^{T} d_t^S q_{kt}\bar{x}_{kt} &\leq \sum_{k=1}^{T}\frac{f_k}{2}\Big(\frac{\sum_{t=1}^{T} d_t^S \bar{X}_{kt}}{u_k} + \bar{Y}_k\Big) + \sum_{k=1}^{T}\sum_{t=1}^{T} d_t^S q_{kt}\bar{X}_{kt} \\ &= \sum_{k=1}^{T} F_k\bar{Y}_k + \sum_{k=1}^{T}\sum_{t=1}^{T} d_t^S Q_{kt}\bar{X}_{kt} \\ &\leq 3b\sum_{t=1}^{T} d_t^S\alpha_t,\end{aligned}$$

which implies

$$\sum_{t=1}^{T}\sum_{k=1}^{T} q_{kt}\bar{x}_{kt} + \sum_{k=1}^{T} f_k\bar{y}_k \leq 6b\sum_{t=1}^{T} d_t^S\alpha_t. \tag{11}$$

Combining Lemmas 1-2 and (11), we conclude the proof. □

4 Discussions

In this paper, we present a $6b$-approximate cross-monotonic competitive cost-sharing method for the SCELSG. There are two possible directions for future research. It would be interesting to design efficient cost-sharing method for the economic lot-sizing game with general concave ordering cost. Another research direction is to design efficient cost-sharing method for the multilevel economic lot-sizing game.

References

1. Brahimi, N., Dauzere-Peres, S., Najid, N.M., Nordli, A.: Single Item Lot Sizing Problems. Eur. J. Oper. Res. 168, 1–16 (2006)
2. Chen, X., Zhang, J.: Duality Approaches to Economic Lot-sizing Games, Working Paper, Stern School of Business, New York University (2007)
3. Chen, X., Zhang, J.: A Stochastic Programming Duality Approach to Inventory Centralization Games. Oper. Res. (to appear)

4. Du, D., Wang, X., Xu, D.: An Approximation Algorithm for the k-Level Capacitated Facility Location Problem. J. Combinatorial Optimization (to appear)
5. Goemans, M.X., Skutella, M.: Cooperative Facility Location Games. J. Algorithms 50, 194–214 (2004)
6. Jain, K., Vazirani, V.V.: Primal-Dual Approximation Algorithms for Metric Facility Location and k-Median Probelms. J. ACM 48, 274–296 (2001)
7. Li, Y., Xu, D.: Soft-Capacity Facility Location Game. Acta Mathematicae Applicatae Sinica, English Series (to appear)
8. Mahdian, M., Ye, Y., Zhang, J.: Approximation Algorithms for Metric Facility Location Problems. SIAM J. Comput. 36, 411–432 (2006)
9. Manne, A.S.: Programming of Dynamic Lot Sizes. Management Sci 4, 115–135 (1958)
10. Pál, M., Tardös, E.: Group Strategyproof Mechanisms via Primal-Dual Algorithms. In: Proceedings of the 44th Annual IEEE Symp. on Foundations of Computer Science, pp. 584–593 (2003)
11. Pochet, Y., Wolsey, L.A.: Lot-Size Models with Backlogging: Strong Reformulations and Cutting planes. Math. Programming 40, 317–335 (1988)
12. Wagner, H.M., Whitin, T.M.: Dynamic Version of the Economic Lot Size Model. Management Sci. 5, 89–96 (1958)
13. Xu, D., Du, D.: The k-Level Facility Location Game. Oper. Res. Letters 34, 421–426 (2006)
14. Xu, D., Yang, R.: A Cost-Sharing Method for an Economic Lot-Sizing Game. Oper. Res. Letters 37, 107–110 (2009)
15. Xu, D., Zhang, S.: Approximation Algorithm for Facility Location with Service Installation Costs. Oper. Res. Letters 36, 46–50 (2008)
16. Zangwill, W.I.: A Backlogging Model and a Multi-Echelon Model of a Dynamic Economic Lot Size Production System-a Network Approach. Management Sci. 15, 506–527 (1969)
17. Zhang, J.: Approximating the Two-Level Facility Location Problem via a Quasi-Greedy Approach. Math. Programming 108, 159–176 (2006)
18. Zhang, J.: Joint Replenishment Game and Maximizing an H-Schur Concave Function Over a Polymatorid, Working paper, Stern School of Business, New York University (2008)
19. Zhang, J.: Cost Allocation for Joint Replenishment Models. Oper. Res. 57, 146–156 (2009)
20. Zhang, J., Chen, B., Ye, Y.: A Multiexchange Local Search Algorithm for the Capacitated Facility Location Problem. Math. Oper. Res. 30, 389–403 (2005)

Improved Bounds for Facility Location Games with Fair Cost Allocation

Thomas Dueholm Hansen and Orestis A. Telelis

Computer Science Department, Aarhus University, Denmark
{tdh,telelis}@cs.au.dk

Abstract. We study Facility Location games played by n agents situated on the nodes of a graph. Each agent orders installation of a facility at a node of the graph and pays connection cost to the chosen node, and shares fairly facility installation cost with other agents having chosen the same location. This game has pure strategy Nash equilibria, that can be found by simple improvements performed by the agents iteratively. We show that this algorithm may need super-polynomial $\Omega(2^{n^{\frac{1}{2}}})$ steps to converge. For metric graphs we show that approximate pure equilibria can be found in polynomial time. On metric graphs we consider additionally *strong* equilibria; previous work had shown that they do not always exist. We upper bound the overall (social) cost of α-approximate strong equilibria within a factor $O(\alpha \ln \alpha)$ of the optimum, for every $\alpha \geq 1$.

1 Introduction

We study Facility Location games in which, self-interested agents situated on the nodes of a network request installation of facilities and connection to them. Each agent i may be associated with non-negative demand weight w_i and may order facility installation on any node v of the network. The agent tries to minimize his individual cost; it consists of the (weighted) distance of i from v and a *fair* share of the facility installation cost at v. Facility installation cost at v is shared evenly among all agents having chosen v, or proportionally to their demand weight in case of weighted agents. In this paper we study pure Nash and *strong* equilibria (PNE and SE) of the described facility location game, and improve or extend previous results [2,8]. We prove a super-polynomial lower bound on the complexity of the *Iterative Best Response* algorithm for finding PNE, and analyze a polynomial-time algorithm for finding approximate PNE on metric networks. Also, for metric networks, we upper bound the *social cost* of approximate SE, that are resilient to coalitional deviations; the social cost is the sum of agents' individual costs. The upper bounding is with reference to the socially optimum cost; this ratio is known as the *Price of Anarchy* [9]. A previous result [8] provided an upper bound only in case of existence of exact SE.

The *fair* cost-sharing rule is of particular interest, as an instance of the *Shapley* Value that is stable for coalitional games [13]. The facility location game is a

[1] The Center for Algorithmic Game Theory, funded by the Carlsberg Foundation.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 174–185, 2009.

special case of the general network design game model with fair cost allocation proposed in [2], which has attracted a significant amount of recent research [1,6,4,7]. In this model each out of n selfish agents wishes to interconnect a given subset of network nodes, using network links. Each link is associated with a *cost* function and a *delay* function, both non-decreasing in the number of agents using the link. Agents try to minimize their individual cost, i.e. the sum of cost shares and delays of the links that an agent uses. For unweighted agents these games have PNE; they belong to the class of *potential* games, introduced by Monderer and Shapley [11]. They are associated with a *potential function* which, given any initial configuration of agents' strategies, can be optimized locally by a sequence of iterative improvements performed by the agents. When no agent can improve his cost any more, the resulting strategy configuration is a PNE. The authors in [2] studied the *Price of Stability* (PoS) of PNE, i.e. the worst-case ratio of the social cost of the least expensive equilibrium, relative to the socially optimum cost. They showed that for constant delay and cost functions $PoS = O(\ln n)$.

Recent work on network design with fair cost allocation has focused on identifying the Price of Stability, particularly when delays are zero. A lower bound of $\frac{4}{3}$ was given in [2], and improved to $\frac{12}{7}$ in [7]. The $O(\log n)$ upper bound for the general single-sink case was improved recently to $O(\frac{\log n}{\log \log n})$ [10]. Albers recently considered general network design with weighted players [1]; she proved an almost tight poly-logarithmic lower bound on the PoS for PNE and studied the Price of Anarchy of approximate SE with weighted and unweighted agents. SE of network design games were first studied by Epstein et al. [6]. The notion of SE is due to Aumann [3]. Metric Facility Location with unweighted players is the only case of the model of [2], in which tight constant bounds are known for the Price of Stability [8]. Also, existence of e-approximate SE ($e = 2.718\ldots$) was shown in [8], even for the non-metric weighted case. Nguyen Kim [12] showed that PNE do not always exist for metric facility location games with weighted agents. Facility Location games have been used for modeling caching systems [5].

Contribution: The complexity of finding PNE for network design games with fair cost allocation is an open problem. It was shown in [2] that $\Omega(2^{\frac{n}{3}})$ Iterative Best Responses may be needed before equilibrium is reached. This was shown though for the most general case of the game model. We prove that, even in the special case of Facility Location the algorithm needs super-polynomial $\Omega(2^{n^{\frac{1}{2}}})$ number of steps. On the positive side, we analyze a polynomial time algorithm for finding 2.258-*approximate* PNE on metric networks with uniform facility costs and weighted agents. This factor improves over the $e = 2.718\ldots$ factor shown in [8] for general Facility Location games. Finally, we develop an analysis for the Price of Anarchy of α-approximate SE for metric Facility Location games with unweighted agents, and show that it is $SPoA = O(\alpha \ln \alpha)$. The $SPoA$ of exact strong equilibria was bound in [8] by a constant, only when they exist.

Definitions: An instance of the facility location game is defined as a tuple (V, d, β, A, u, w). (V, d) defines a graph with vertex set V and distance function $d : V \times V \to \mathbb{R}_0^+ \cup \{\infty\}$. $\beta : V \to \mathbb{R}_0^+ \cup \{\infty\}$ associates every vertex with a

facility opening cost. $A = \{1, \ldots, n\}$ is the set of agents residing on the graph, with $u : A \to V$ mapping agents to vertices, and $w : A \to \mathbb{R}^+$ associating agents with positive demand weight. We use β_v, u_i and w_i instead of $\beta(v)$, $u(i)$ and $w(i)$, respectively. Each agent is a player with strategy space V. Let $s_i \in V$ be the strategy of $i \in A$. A *strategy profile* is denoted by $s = (s_1, \ldots, s_n)$. We use s_{-i} for $(s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$. $W_s(v) = \sum_{i:s_i=v} w_i$ is the sum of weights of agents playing v in s. The individual cost $c_i(s)$ of i under s is $c_i(s) = w_i d(u_i, s_i) + \frac{\beta_{s_i} w_i}{W_s(s_i)}$, i.e. i pays (weighted) connection cost and a fair share of the facility opening cost. Let $F_s = \{s_i | i \in A\}$. The *social cost* of s is defined as:

$$c(s) = \sum_{i \in A} c_i(s) = \sum_{i \in A} w_i d(u_i, s_i) + \sum_{v \in F_s} \beta_v$$

and coincides with the objective function of the facility location problem [14]. s is *socially optimum* if it is an optimum solution to the facility location problem.

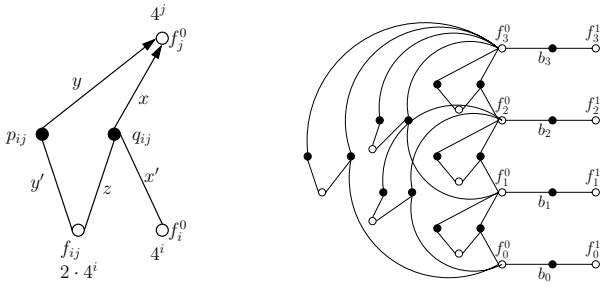
The game is metric when d is metric, unweighted when $w_i = 1$ for all $i \in A$, and has uniform facility costs when $\beta_v = 1$ for all $v \in V$. For $\alpha \geq 1$ a strategy profile s is an α-approximate pure Nash equilibrium (PNE) if for every player $i \in A$ and every strategy $s'_i \in V$ it is $c_i(s) \leq \alpha c_i(s_{-i}, s'_i)$. It is an α-approximate *strong* equilibrium (SE) if for every non-empty subset $I \subseteq A$, and pure strategies $s'_i \neq s_i$ for every $i \in I$, there is a $j \in I$ with $c_j(s) \leq \alpha c_j(s_{-I}, s'_I)$ [1,6,3]. When $\alpha = 1$, s is a PNE or a SE respectively.

For a strategy profile s, a *best response* of player i is any strategy s'_i that minimizes $c_i(s_{-i}, s'_i)$. The *Iterative Best Response* algorithm is initialized at an arbitrary profile s. It performs iteratively the following: while there is any player $i \in A$ and a best response s'_i of i with $c_i(s_{-i}, s'_i) > c_i(s)$, set s equal to (s_{-i}, s'_i). The algorithm terminates at a profile s which is a PNE [2,11].

2 Complexity of Iterative Best Response

We derive a super-polynomial lower bound on the convergence complexity of the Iterative Best Response algorithm for non-metric facility location, later showing that it can be modified to work for metric facility location as well. We exhibit a family of instances of the game and a *schedule* of iterative best responses on this family, that simulates a *bit counter*. We assume n *bit* agents b_i, $i = 0 \ldots n-1$, each choosing among two facility nodes f_i^0 and f_i^1, referred to as the 0-strategy and 1-strategy of b_i. We use auxiliary agents, whose choices will enable the *bit* agents to switch between 0- and 1-strategies. For any other location in the graph we make the distance of b_i big enough, to discourage b_i from connecting there.

Every pair of bits i and $j > i$ will be grouped by a *gadget* G_{ij}, see fig. 1(a) for an illustration. G_{ij} has two auxiliary agent nodes q_{ij} (referred to as *inner* agent) and p_{ij} (referred to as outer agent) and one extra facility node f_{ij}. Agent q_{ij} is attached to facility nodes f_i^0, f_j^0, and f_{ij}. Agent p_{ij} is attached only to the facility nodes f_j^0 and f_{ij}. The total number of used gadgets is $O(n^2)$. Fig. 1(b) illustrates an instance with 4 bit agents. We introduce a schedule of Iterative



(a) A gadget G_{ij}. **(b)** Construction with 4 bits.

Fig. 1. A gadget (initial state) and a construction of 4 bits, pairwise grouped by gadgets

Best Response, formally described as algorithm 1, which we call COUNT. Initially every bit agent b_i is connected to f_i^0, and in every gadget G_{ij} agents p_{ij} and q_{ij} are connected to f_j^0. During execution of COUNT the changes occurring to p_{ij}, q_{ij} are depicted in order in fig. 2. Let us explain how COUNT(k) proceeds recursively. Initially, for every $j \in \{0, \ldots, k\}$, agents b_j, p_{ij} and q_{ij} for all $i < j$ are connected to f_j^0. The following high level steps occur:

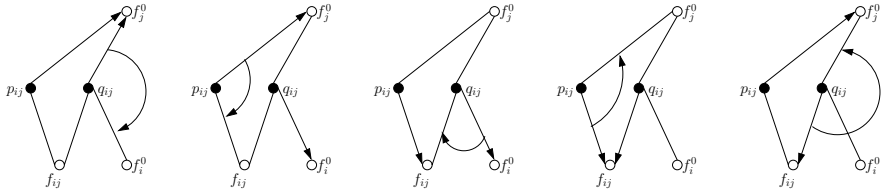
1. At first COUNT$(k-1)$ is called (line 1), that results in all agents b_i, $i < k$ being connected to f_i^1; also, agents p_{ji}, q_{ji} with $j < i$ are connected to f_{ji}.
2. Agent b_k switches from f_k^0 to f_k^1 (line 2). Lines 3-10 cause bit agents b_i, $i < k$ to switch back to f_i^0, so that number 2^{k-1} is formed by the strategy profile.
3. Finally, a call to COUNT$(k-1)$ is performed once more (line 11).

We explain lines 3-10 of the schedule. For any value of k, agent b_k is the first to leave f_k^0 and deviate to f_k^1. Agents q_{ik} follow, in order $i = k-1 \ldots 0$ and they deviate to f_i^0 (*Inner Fall* - fig. 2(a)). Each of them helps attracting sequentially *back to* f_i^0 all agents p_{ji}, q_{ji} in orders $j = i-1 \ldots 0$ and $j = 0 \ldots i-1$ respectively (*Outer* and *Inner Reset* - fig. 2(d) & (e)); recall that they were connected to f_{ji} after the high level step 1. Then, agents b_i, $i = k-1 \ldots 0$, are also attracted to f_i^0; at this time there is one agent more - q_{ij} - connected to f_i^0, than when b_i abandoned f_i^0 for f_i^1. Finally, this additional agent q_{ik} is attracted away from f_i^0, to create incentive for deviation to the bit agents b_i, $i < k$. This is achieved first by deviation of agents p_{ik} to f_{ik} in order $i = 0 \ldots k-1$ (*Outer Fall* - fig. 2(c)); this creates incentive for q_{ik} to leave f_i^0 for f_{ik} (*Inner Join* - fig. 2(d)).

In order for the moves specified by the schedule (algorithm 1) to be *best responses* of agents, we assign facility installation costs and distances so that certain inequalities hold. We assign f_i^0 cost 4^i, and facility nodes f_{ij} cost $2 \cdot 4^i$. Let distances be named as in fig. 1(a). The following inequalities should hold for the movements of a gadget's G_{ij} agents to be best responses. We consider movements of bit agents later. We use f for $2 \cdot 4^i$ - the cost of f_{ij}:

$$\textbf{Inner Fall of } q_{ij}: \quad x + \frac{4^j}{i+j+1} > x' + 4^i \quad , \; i = j-1 \ldots 0 \tag{1}$$

$$\textbf{Inner Fall of } q_{ij}: \quad z + f > x' + 4^i \quad , \; i = j-1 \ldots 0 \tag{2}$$



(a) Inner Fall **(b)** Outer Fall **(c)** Inner Join **(d)** Outer Reset **(e)** Inner Reset

Fig. 2. Movements performed by the "inner" and "outer" agents of a gadget G_{ij}

Algorithm 1. COUNT(k): counts up to $2^k - 1$

1: **if** $k > 1$ **then** COUNT$(k-1)$ {*Count up to* $2^{k-1}-1$}
2: $s_{b_k} \leftarrow f_k^1$ {*Set* b_k *to* 1}
3: **for** $i = k-1 \dots 0$ **do**
4: $\quad s_{q_{ik}} \leftarrow f_i^0$ {*Perform an Inner Fall for* q_{ik}}
5: $\quad$ **for** $j = i-1 \dots 0$ **do** $s_{p_{ji}} \leftarrow f_i^0$ {*Perform an Outer Reset for* p_{ji}}
6: $\quad$ **for** $j = 0 \dots i-1$ **do** $s_{q_{ji}} \leftarrow f_i^0$ {*Perform an Inner Reset for* q_{ji}}
7: $\quad s_{b_i} \leftarrow f_i^0$ {Set b_i to 0}
8: **end for**
9: **for** $i = 0 \dots k-1$ **do** $s_{p_{ik}} \leftarrow f_{ik}$ {*Perform an Outer Fall for* p_{ik}}
10: **for** $i = 0 \dots k-1$ **do** $s_{q_{ik}} \leftarrow f_{ik}$ {*Perform an Inner Join for* q_{ik}}
11: **if** $k > 1$ **then** COUNT$(k-1)$ {*Count up to* $2^{k-1}-1$}

$$\textbf{Outer Fall of } p_{ij}: \quad y + \frac{4^j}{j-i} > y' + f \quad , \; i = 0 \dots j-1 \tag{3}$$

$$\textbf{Inner Join of } q_{ij}: \quad x' + \frac{4^i}{2i+2} > z + \frac{f}{2} \quad , \; i = 0 \dots j-1 \tag{4}$$

$$\textbf{Outer Reset of } p_{ij}: \quad y' + \frac{f}{2} > y + \frac{4^j}{j-i+1} \quad , \; i = j-1 \dots 0 \tag{5}$$

$$\textbf{Inner Reset of } q_{ij}: \quad z + f > x + \frac{4^j}{i+j+2} \quad , \; i = 0 \dots j-1 \tag{6}$$

$$\textbf{Inner Reset of } q_{ij}: \quad x' + 4^i > x + \frac{4^j}{i+j+2} \quad , \; i = 0 \dots j-1 \tag{7}$$

Correctness of inequalities (1)-(7): For (1) and (2) note that *Inner Falls* of q_{ij} agents start when b_j has already left f_j^0. Initially there are $2j+1$ agents connected to each f_j^0. Thus each q_{ij} for $i = j-1 \dots 0$ shares the cost of f_j^0 with $i+j+1$ agents exactly before deviating. Furthermore, it is the first to connect to f_i^0, and no agent is connected to f_{ij}. For (3) we note that the number of p_{ij} agents connected to f_j^0 is exactly $j-i$, for $i = 0 \dots j-1$. Agent b_j and all q_{ij} agents have already deviated before at the time when *Outer Falls* begin. Since the order by which they occur is $i = 0 \dots j-1$ (3) is justified; each p_{ij} is the first to connect to f_{ij}. For (4), note that q_{ij} has attracted back to f_i^0 the $2i+1$ agents connected to f_i^0 in the initial state. It is the second agent (after p_{ij}) to deviate to f_{ij}. The rest of the inequalities are justified if we consider that agents

are reset in reverse order of their deviation, attracted to f_j^0 by an additional agent connected there, than when they deviated. To solve the system (1)-(7) at first we simplify as follows:

$$(3),\ (5) \quad \Rightarrow \quad \frac{4^j}{j-i+1} - 4^i < y' - y < \frac{4^j}{j-i} - 2 \cdot 4^i \tag{8}$$

$$(1),\ (7) \quad \Rightarrow \quad \frac{4^j}{i+j+2} - 4^i < x' - x < \frac{4^j}{i+j+1} - 4^i \tag{9}$$

$$(2),\ (4) \quad \Rightarrow \quad \frac{(2i+1)4^i}{2i+2} < x' - z < 4^i \tag{10}$$

The ranges given by (8)-(10) are non-empty. Several values for y,y' satisfy (8). For x',x and z we equalize $x'-x$, $x'-z$ to the average of bounds given by (9), (10) respectively. Then, we solve a system of 2 equations and 3 variables to express x' and z as functions of x. Sufficiently large $x > 0$, no larger than $O(4^n)$, ensures $x', z > 0$. This settles best responses of agents of G_{ij}. Let us decide distances of b_i from f_i^0 and f_i^1, and cost of f_i^1; b_i always leaves f_i^0 for f_i^1 when $2i$ other agents are connected to f_i^0, and returns to f_i^0 from f_i^1 when $2i+1$ other agents are connected to f_i^0. This yields 2 inequalities with 3 variables which can be easily satisfied. Facility nodes and distances other than the ones considered (fig. 1(b)) have a very big cost. The graph can be made metric by addition of a large constant to all described distances and usage of triangle inequality for the rest. Since we use $O(n^2)$ players we get:

Theorem 1. *Iterative Best Response for the Facility Location game with fair cost allocation may need superpolynomial $\Omega(2^{n^{\frac{1}{2}}})$ steps in the number of players.*

3 Approximation of Equilibria for Uniform Facility Costs

It is not known whether pure equilibria of facility location games can be computed efficiently. In light of Theorem 1 it seems reasonable to consider computation of approximate equilibria as a first step towards this goal. This was also suggested in [2] for the general network design model. In this section we restrict our attention to metric facility location games with uniform facility opening costs and describe an algorithm for computing approximate equilibria of such games. The existence of such an algorithm furthermore improves previous bounds for existence of approximate equilibria [8] in this setting when agents are weighted.

Let (V, d, β, A, u, w) be a metric weighted facility location game with uniform facility costs. Let $A(U) = \cup_{i:u_i \in U}\{i\}$ be the set of agents residing on nodes $v \in U$. We claim that Algorithm 2 constructs an approximate equilibrium s.

The parameter δ in the input of Algorithm 2 facilitates analysis of the algorithm. $S = \cup_{i \in A}\{u_i\}$ is the set of vertices where agents are residing. The algorithm incrementally constructs a set $F_s \subseteq S$ by adding more and more facilities. F_δ is maintained as the set of agents with a distance of at most $\frac{\delta}{w}$ to a facility $f \in F_s$, where w is the weight of the agent in question. While $S \setminus F_\delta \neq \emptyset$, the vertex of the agent in $S \setminus F_\delta$ with highest weight is added to F_s. Intuitively,

Algorithm 2. Compute an approximate equilibrium

Input: (V, d, β, A, u, w), δ.
$F_s \leftarrow \emptyset$, $F_\delta \leftarrow \emptyset$, $S \leftarrow \cup_{i \in A}\{u_i\}$.
while $S \setminus F_\delta \neq \emptyset$ **do**
  Pick $i \in A(S \setminus F_\delta)$, such that $\forall j \in A(S \setminus F_\delta) : w_i \geq w_j$.
  $F_s \leftarrow F_s \cup \{u_i\}$.
  for $j \in A(S \setminus F_\delta)$ **do**
    if $w_j d(u_i, u_j) \leq \delta$ **then** $F_\delta \leftarrow F_\delta \cup \{u_j\}$.
  end for
end while
For all $i \in A : s_i \leftarrow \arg\min_{f \in F_s} d(u_i, f)$.
return $s = (s_1, \ldots, s_n)$.

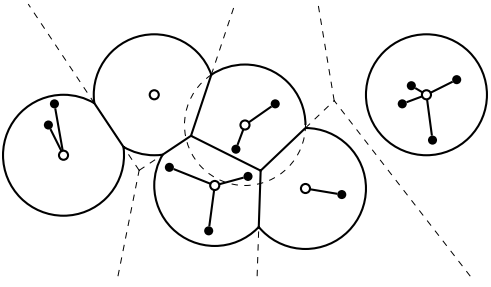


Fig. 3. Approximate equilibrium computed for an unweighted facility location game. The Voronoi diagram and the balls are included for illustrative purposes.

F_δ is the union of balls centered at the facilities of F_s. See Figure 3 for an illustration of the outcome for a facility location game with unweighted agents.

Theorem 2. *Algorithm 2 computes a* 2.258*-approximate equilibrium (*1.781*-approximate equilibrium) in* $O(n^2)$ *time given a metric weighted (unweighted) facility location game with uniform facility opening costs and* n *agents.*

Proof. First, let us observe that the running time of Algorithm 2 is at most quadratic in the number of agents. Indeed, at most $|S| \leq n$ facilities will be added to F_s, and finding the next facility and updating $S \setminus F_\delta$ can be done in $O(n)$ time. Connecting the agents to the nearest facility of F_s can be done in $O(n \cdot |F_s|)$ time, so overall the running time is $O(n^2)$.

Next, let us show that we have indeed computed a strategy profile s that is an approximate equilibrium. Let $i \in A$ be any agent. Let $f \in F_s$ be the facility that i is connected to in s, and let $f' \in F_s \cup \{u_i\}$ be a best response of i resulting in a strategy profile s'. Note that when i opens a facility on his own, i.e. $f' \in V \setminus F_s$, it is always a best response to open it at u_i. We need to show that $c_i(s)/c_i(s') \leq \alpha$ for some constant α. We can split this into the following two cases. We will use w_f, $f \in F_s$, to denote the weight of the agent that caused f to be picked for F_s.

Case 1: $f' = u_i \notin F_s$, i.e. agent i opens a facility on his own.

$$\frac{c_i(s)}{c_i(s')} = w_i d(u_i, f) + \frac{w_i}{W_s(f)} \leq w_i d(u_i, f) + \frac{w_i}{w_i + w_f} < \delta + \frac{2}{3} \tag{11}$$

In (11) we use $w_i d(u_i, f) \leq \delta$. Also, because addition of some facility g to F_s caused u_i to be covered by F_δ, we take 2 cases. Either $f = g$ and $w_i \leq w_f$, or $f \neq g$ and $w_i < 2w_f$ because $w_i d(u_i, g) \leq \delta < w_f d(f, g) \leq w_f(d(f, u_i) + d(u_i, g)) \leq w_f 2d(u_i, g)$. In both cases $w_i < 2w_f$, hence the last inequality.

Case 2: $f' \in F_s$, i.e. agent i connects to another facility that is already open.

$$\frac{c_i(s)}{c_i(s')} = \frac{w_i d(u_i, f) + w_i / W_s(f)}{w_i d(u_i, f') + w_i / W_{s'}(f')} < 1 + \frac{1}{w_f d(u_i, f')} \tag{12}$$

$$< 1 + 2/[w_f d(f, f')] < 1 + 2/\delta \tag{13}$$

In (12) we use that $d(u_i, f) \leq d(u_i, f')$ and that $w_f \leq W_s(f)$. In (13) we use the triangle inequality saying that $d(f, f') \leq d(f, u_i) + d(u_i, f') \leq 2d(u_i, f')$.

Hence, if $\alpha = \max\left(\delta + \frac{2}{3}, 1 + \frac{2}{\delta}\right)$, no agent will gain more than a factor of α by deviating. Then α is minimized for $\delta = \frac{1}{6}(1+\sqrt{73})$, with $\alpha = \frac{1}{6}(5+\sqrt{73}) < 2.258$.

If agents are unweighted, we would get $c_i(s)/c_i(s') \leq \delta + 1/2$ instead of (11) and $c_i(s)/c_i(s') < 1+1/\delta$ instead of (12). The second inequality follows from two subcases. Either i is the only agent going to f in s, in which case $c_i(s)/c_i(s') < 1 + 1/(w_f d(f, f'))$, or there is someone else sharing the cost, in which case $c_i(s)/c_i(s') < 1 + 1/(2w_f d(u_i, f'))$. Hence, we could let $\alpha = \max\left(\delta + \frac{1}{2}, 1 + \frac{1}{\delta}\right)$, which is minimized for $\delta = (1 + \sqrt{17})/4$, with $\alpha = (3 + \sqrt{17})/4 < 1.781$. □

Approximate strong equilibria are also approximate pure equilibria, by definition. The following corollary improves over the best known approximation factor of $e = 2.718\ldots$, for approximate pure equilibria with weighted agents [8].

Corollary 1. 2.258-*approximate equilibria are guaranteed to exist in weighted metric facility location games with uniform facility opening costs.*

4 Strong Price of Anarchy: The Unweighted Metric Case

The metric unweighted Facility Location game does not always have exact SE, but α-approximate SE always exist, for $\alpha \geq e = 2.718\ldots$ [8]. We prove an upper bound for the Price of Anarchy of α-approximate SE (Strong Price of Anarchy - $SPoA_\alpha$). Exact SE were shown to have $SPoA = O(1)$, when they exist [8]. Let s be any α-approximate SE, s^* the socially optimum profile, and $A_{s^*}(v)$ the subset of agents that are connected to $v \in F_{s^*}$ under s^*. For $v \in F_{s^*}$ define $c_v(s^*) = \beta_v + \sum_{i: s_i^* = v} d(u_i, v)$. We upper bound the $SPoA_\alpha$ as:

$$SPoA_\alpha = \frac{c(s)}{c(s^*)} = \frac{\sum_{v \in F_{s^*}} \sum_{i \in A_{s^*}(v)} c_i(s)}{\sum_{v \in F_{s^*}} c_v(s^*)} \leq \max_{v \in F_{s^*}} \frac{\sum_{i \in A_{s^*}(v)} c_i(s)}{c_v(s^*)}$$

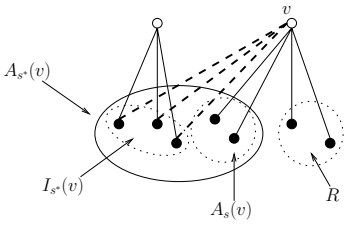


Fig. 4. The situation examined in the proof of theorem 3

We will upper bound the latter ratio for any $v \in F_{s^*}$, so as to prove:

Theorem 3. *For any $\alpha \geq 1$, the Price of Anarchy of α-approximate strong equilibria in the unweighted metric Facility Location game is $O(\alpha \ln \alpha)$.*

For any facility node $v \in F_{s^*}$, define $A_s(v) \subseteq A_{s^*}(v)$ to be the subset of those agents that are connected to v both in s^* and s. Define $I_{s^*}(v) = A_{s^*}(v) \setminus A_s(v)$ to be the subset of agents that are connected to v in s^*, but not in s. Let R denote agents connected to v under s, with $R \subseteq A \setminus A_{s^*}(v)$. See fig. 4 for an illustration. To simplify notation we use $d(u_i, v) = x_i^*$, for $i \in A_{s^*}(v)$. With respect to R we will consider two cases. At first assume that, under s_{-R}, no agent of $I_{s^*}(v)$ has incentive to deviate to v in with coalition $I_{s^*}(v)$. Then for every $i \in I_{s^*}(v)$ it is $c_i(s) \leq \alpha c_i(s^*)$, because s is an α-approximate SE. Then:

$$\sum_{i \in A_{s^*}(v)} c_i(s) = \sum_{i \in A_s(v)} c_i(s) + \sum_{i \in I_{s^*}(v)} c_i(s) \leq \beta_v + \sum_{i \in A_s(v)} x_i^* + \alpha \left(\beta_v + \sum_{i \in I_{s^*}(v)} x_i^* \right)$$

Thus $\sum_{i \in A_{s^*}(v)} c_i(s) \leq (1+\alpha) \left(\beta_v + \sum_{i \in A_{s^*}(v)} x_i^* \right) = (1+\alpha) c_v(s^*)$.

For the rest of the analysis we assume there exists at least one agent $i \in I_{s^*}(v)$ willing to deviate to v in coordination with the coalition $I_{s^*}(v)$ under s_{-R}. Define a *minimal disagreeing subset* $I^0_{s^*}(v) \subseteq I_{s^*}(v)$ as a *minimal subset of misconnected agents* containing an agent i that would deviate to v with $I^0_{s^*}(v)$, under s_{-R}. Also define $J_{s^*}(v) = I_{s^*}(v) \setminus I^0_{s^*}(v)$. Fix $i \in I^0_{s^*}(v)$ to be from now on the agent (or one of them if there are many) that would deviate to v in coordination with $I^0_{s^*}(v)$. We call i the *unstable* agent of the minimal disagreeing subset $I^0_{s^*}(v)$. By definition, the following holds for the *unstable* agent i:

$$c_i(s) > \alpha \left(x_i^* + \frac{\beta_v}{|I^0_{s^*}(v)| + |A_s(v)|} \right) \tag{14}$$

The rest of the analysis consists of bounds for agents in $I^0_{s^*}$ and $J_{s^*}(v)$ separately. In particular, lemmas 1 and 2 describe upper bounds for these sets respectively.

Lemma 1. *Let $I_{s^*}(v)$ be a subset of misconnected agents under an α-approximate strong equilibrium profile s. Let $I^0_{s^*}(v)$ be a minimal disagreeing subset of $I_{s^*}(v)$ and $i \in I^0_{s^*}(v)$ an unstable agent. Then:*

$$\sum_{l \in I^0_{s^*}(v)} c_l(s) \leq 2\alpha \left(\beta_v + \sum_{l \in I^0_{s^*}(v)} x_l^* \right) \tag{15}$$

Proof. By minimality of $I^0_{s^*}(v)$, every agent $l \in I^0_{s^*}(v)$ is *not willing* to deviate to v under s_{-R} with a coalition of size $|I^0_{s^*}(v)| - 1$. Thus for every $l \in I^0_{s^*}(v)$:

$$c_l(s) \leq \alpha \left(x^*_l + \frac{\beta_v}{|I^0_{s^*}(v)| + |A_s(v)| - 1} \right) \leq \alpha \left(x^*_l + \frac{\beta_v}{|I^0_{s^*}(v)| - 1} \right)$$

Summing over $I^0_{s^*}(v)$ yields the upper bound of (15). □

Lemma 2. *Let $I_{s^*}(v)$ be a subset of misconnected agents under an α-approximate strong equilibrium profile s. Let $I^0_{s^*}(v)$ be a minimal disagreeing subset of $I_{s^*}(v)$ and $J_{s^*}(v) = I_{s^*}(v) \setminus I^0_{s^*}(v)$. Then:*

$$\sum_{j \in J_{s^*}(v)} c_j(s) \leq \alpha \sum_{j \in J_{s^*}(v)} x^*_j + \alpha \beta_v \Big(H(|A_{s^*}(v)|) - H(|I^0_{s^*}(v)| + |A_s(v)|) \Big) \quad (16)$$

Proof. We use an argument by Albers [1]. W.l.o.g. name agents $j \in J_{s^*}(v)$ by distinct indices $1, \ldots, |J_{s^*}(v)|$ and define a series of supersets of $I^0_{s^*}(v)$, as follows: $I^j_{s^*}(v) = I^{j-1}_{s^*}(v) \cup \{j\}$. Since s is an α-approximate SE, every set $I^j_{s^*}(v)$ contains an agent not willing to deviate to v in coordination with $I^j_{s^*}(v)$; it is found either in $I^0_{s^*}(v) \setminus \{i\}$ or in $I^j_{s^*}(v) \setminus I^0_{s^*}(v)$. We can assume w.l.o.g. that for $I^j_{s^*}(v)$ this agent is j; otherwise we exchange j with an agent from $I^0_{s^*}(v) \setminus \{i\}$. By definition of a minimal disagreeing subset, this will not harm our previous results. Then:

$$c_j(s) \leq \alpha \Big(x^*_j + \frac{\beta_v}{|I^j_{s^*}(v)| + |A_s(v)| + |R|} \Big), \quad j = 1, \ldots, |J_{s^*}(v)| \in J_{s^*}(v) \quad (17)$$

We omit $|R|$ and sum the inequality over $j \in J_{s^*}(v)$. The result follows. □

The following lemma will provide a lower bound for the socially optimum connection cost $\sum_{j \in J_{s^*}(v)} x^*_j$ appearing in the upper bounding expression (16).

Lemma 3. *Let $I_{s^*}(v)$ be a subset of misconnected agents under α-approximate strong equilibrium profile s. Let $I^0_{s^*}(v)$ be a minimal disagreeing subset of $I_{s^*}(v)$ and $J_{s^*}(v) = I_{s^*}(v) \setminus I^0_{s^*}(v)$. Then for $r = |I^0_{s^*}(v)| + |A_s(v)|$:*

$$\sum_{j \in J_{s^*}(v)} x^*_j \geq \frac{\beta_v}{1+\alpha} \Big(\frac{|A_{s^*}(v)| - \lceil \alpha r \rceil}{r} - \alpha \Big(H(|A_{s^*}(v)|) - H(\lceil \alpha r \rceil) \Big) \Big) \quad (18)$$

Proof. Let i be the fixed *unstable* agent of $I^0_{s^*}(v)$. Note that, under strategy profile s, i does not have an incentive to join facility node s_j for any $j \in J_{s^*}(v)$. Thus if j pays for s_j a share of $\frac{\beta_{s_j}}{\lambda_j}$ (that is, s_j serves λ_j agents in total in s):

$$c_i(s) \leq \alpha \left(d(u_i, s_j) + \frac{\beta_{s_j}}{1 + \lambda_j} \right) \leq \alpha \left(d(u_i, v) + d(u_j, v) + d(u_j, s_j) + \frac{\beta_{s_j}}{\lambda_j} \right) \Rightarrow$$

$$c_i(s) \leq \alpha \left(x^*_i + x^*_j + c_j(s) \right) \leq \alpha \left(x^*_i + x^*_j + \alpha \left(x^*_j + \frac{\beta_v}{|I^j_{s^*}(v)| + |A_s(v)|} \right) \right) \quad (19)$$

The last inequality derives by (17) for $c_j(s)$ and by safely omitting $|R|$. Using (19) and the lower bound for $c_i(s)$ by (14), we solve for x_j^*. By definition of $I_{s^*}^j(v)$ in lemma 2, $|I_{s^*}^j(v)| = |I_{s^*}^0(v)| + j$, $j = 1, \ldots, |J_{s^*}(v)|$. Then for $j = 1 \ldots |J_{s^*}(v)|$:

$$x_j^* \geq \max\Big\{0, \frac{\beta_v}{1+\alpha}\Big(\frac{1}{|I_{s^*}^0(v)| + |A_s(v)|} - \frac{\alpha}{j + |I_{s^*}^0(v)| + |A_s(v)|}\Big)\Big\}$$

Finally we sum up the latter bound over all j. Notice that x_j^* becomes non-zero only when $j + |I_{s^*}^0(v)| + |A_s(v)| \geq \alpha(|I_{s^*}^0(v)| + |A_s(v)|)$. Since $j + |I_{s^*}^0(v)| + |A_s(v)|$ is an integral value, it turns out that x_j^* becomes non-negative for those values of j for which it is $j + |I_{s^*}^0(v)| + |A_s(v)| \geq \lceil \alpha(|I_{s^*}^0(v)| + |A_s(v)|)\rceil$. Then, by setting $r = |I_{s^*}^0(v)| + |A_s(v)|$, and by summing up over all j we obtain (18). □

Proof of Theorem 3. We put everything together. A *lower bound* on $c_v(s^*)$ is:

$$c_v(s^*) \geq \beta_v + \sum_{j \in J_{s^*}(v)} x_j^* + \sum_{l \in I_{s^*}^0(v)} x_l^* + \sum_{i \in A_s(v)} x_i^* \tag{20}$$

Accordingly, we obtain the following *upper bound* on $\sum_{i \in A_{s^*}(v)} c_i(s)$ by (15):

$$\begin{aligned}\sum_{i \in A_{s^*}(v)} c_i(s) &\leq \sum_{l \in I_{s^*}^0(v)} c_l(s) + \sum_{j \in J_{s^*}(v)} c_j(s) + \sum_{i \in A_s(v)} c_i(s) \\ &\leq 2\alpha\Big(\beta_v + \sum_{l \in I_{s^*}^0(v)} x_l^*\Big) + \sum_{j \in J_{s^*}(v)} c_j(s) + \sum_{i \in A_s(v)} c_i(s)\end{aligned} \tag{21}$$

We use (20), (21) for bounding $\frac{1}{c_v(s^*)} \sum_{i \in A_{s^*}(v)} c_i(s)$, and substitute by (16), (18). Using bounds $\gamma + \ln m \leq H(m) \leq 1 + \ln m$ for the harmonic numbers ($\gamma > 0.5$ is Euler' constant), and $\lceil \alpha r \rceil \leq (1+\alpha)r$, we obtain:

$$SPoA \leq 1 + 2\alpha + 2\alpha \frac{1 - \gamma + \ln \frac{|A_{s^*}(v)|}{r}}{\frac{1}{1+\alpha}\Big(\frac{|A_{s^*}(v)|}{r} - \alpha \ln \frac{|A_{s^*}(v)|}{r} + \alpha(\gamma + \ln \alpha - 1)\Big)}$$

We substitute γ with 0.5 and, by setting $y = \frac{|A_{s^*}(v)|}{r}$, we simplify to:

$$SPoA_\alpha < 1 + 2\alpha + 2\sqrt{e}(1+\alpha)\frac{\ln \alpha + \ln(y\sqrt{e}/\alpha)}{y\sqrt{e}/\alpha - \sqrt{e}\ln(y\sqrt{e}/\alpha)}$$

For $x > 0$ it is $\frac{1}{x - \sqrt{e}\ln x} = O(1)$ and $\frac{\ln x}{x - \sqrt{e}\ln x} = O(1)$, thus $SPoA_\alpha = O(\alpha \ln \alpha)$.

Corollary 2. *The Price of Anarchy of e-approximate strong equilibria for the metric unweighted Facility Location game is* $O(1)$.

5 Future Work

Computing PNE and approximate strong equilibria remains wide open. Hardness results in this respect would be of major interest as they would not only apply

to facility location, but also to more general models. Besides being of independent interest, further improvements of approximation algorithms might lead to insights in this area. In particular, one might produce a better way of assigning agents to facilities in Algorithm 2. Studying the quality of computed (approximate) equilibria would be of importance as well. A wide range of approximation algorithms are known for the classical facility location problem, some of which might be modified to produce approximate equilibria of low cost. Finally, the existence of PNE of weighted facility location games with uniform facility costs remains unresolved.

References

1. Albers, S.: On the value of coordination in network design. In: Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 294–303 (2008)
2. Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, E., Wexler, T., Roughgarden, T.: The Price of Stability for Network Design with Fair Cost Allocation. SIAM J. Comput. 38(4), 1602–1623 (2008)
3. Aumann, Y.: Acceptable Points in General Cooperative n-Person Games. In: Tucker, A.W., Luce, R.D. (eds.) Contributions to the Theory of Games IV, Annals of Mathematics Study 40, pp. 287–324 (1959)
4. Chekuri, C., Chuzhoy, J., Lewin-Eytan, L., Naor, J., Orda, A.: Non-cooperative multicast and facility location games. In: Proc. ACM Conf. on Electronic Commerce (EC), pp. 72–81 (2006)
5. Chun, B., Chaudhuri, K., Wee, H., Barreno, M., Papadimitriou, C.H., Kubiatowicz, J.: Selfish caching in distributed systems: a game-theoretic analysis. In: Proc. ACM Symp. on Principles of Distributed Computing, pp. 21–30 (2004)
6. Epstein, A., Feldman, M., Mansour, Y.: Strong Equilibrium in Cost Sharing Connection Games. In: Proc. ACM Conf. on Electronic Commerce (EC), pp. 84–92 (2007)
7. Fiat, A., Kaplan, H., Levy, M., Olonetsky, S., Shabo, R.: On the Price of Stability for Designing Undirected Networks with Fair Cost Allocations. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 608–618. Springer, Heidelberg (2006)
8. Hansen, T.D., Telelis, O.A.: On Pure and (approximate) Strong Equilibria of Facility Location Games. In: Papadimitriou, C., Zhang, S. (eds.) WINE 2008. LNCS, vol. 5385, pp. 490–497. Springer, Heidelberg (2008)
9. Koutsoupias, E., Papadimitriou, C.H.: Worst-case Equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
10. Li, J.: An $o(\frac{\log n}{\log \log n})$ Upper Bound on the Price of Stability for Undirected Shapley Network Design Games. arXiv:0812.2567v1 [cs.GT] (2008)
11. Monderer, D., Shapley, L.S.: Potential games. Games and Economic Behavior 14, 124–143 (1996)
12. Nguyen, K.T.: Private communication (2009), thang@lix.polytechique.fr
13. Osborne, M.J., Rubinstein, A.: A Course in Game Theory. MIT Press, Cambridge (1994)
14. Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2001)

Two-Level Heaps: A New Priority Queue Structure with Applications to the Single Source Shortest Path Problem

K. Subramani[1,⋆] and Kamesh Madduri[2,⋆⋆]

[1] LDCSEE, West Virginia University
Morgantown, WV 26506, USA
ksmani@csee.wvu.edu
[2] Computational Research Division, Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA
KMadduri@lbl.gov

Abstract. The Single Source Shortest Paths problem with positive edge weights (SSSPP) is one of the more widely studied problems in Operations Research and Theoretical Computer Science [1,2] on account of its wide applicability to practical situations. This problem was first solved in polynomial time by Dijkstra [3], who showed that by extracting vertices with the smallest distance from the source and relaxing its outgoing edges, the shortest path to each vertex is obtained. Variations of this general theme have led to a number of algorithms, which work well in practice [4,5,6]. At the heart of a Dijkstra implementation is the technique used to implement a priority queue. It is well known that using Dijkstra's approach requires $\Omega(n \log n)$ steps on a graph having n vertices, since it essentially sorts vertices based on their distances from the source. Accordingly, the fastest implementation of Dijkstra's algorithm on a graph with n vertices and m edges should take $\Omega(m + n \cdot \log n)$ time and consequently the Dijkstra procedure for SSSPP using Fibonacci Heaps is optimal, in the comparison-based model. In this paper, we introduce a new data structure to implement priority queues called Two-Level Heap (TLH) and a new variant of Dijkstra's algorithm called *Phased Dijkstra*. We contrast the performance of Dijkstra's algorithm (both the simple and the phased variants) using a number of data structures to implement the priority queue and empirically establish that Two-Level heaps are far superior to Fibonacci heaps on every graph family considered.

1 Introduction

This paper is concerned with the design of fast empirical strategies for the Single Source Shortest Path problem with positive weights (SSSPP). SSSPP is one of

⋆ This research was supported in part by the Air-Force Office of Scientific Research under contract FA9550-06-1-0050 and in part by the National Science Foundation through Award CCF-0827397.

⋆⋆ This work was supported in part by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 186–196, 2009.

the more widely studied problems within the Operations Research and Theoretical Computer Science communities on account of its wide applicability. This problem was first solved efficiently in [3] using a variant of Breadth-First Search (BFS). Since then, a number of variants of the general relaxation-based theme have been proposed, each claiming success on a select class of inputs. At the heart of the Dijkstra approach is a priority queue structure which implements the EXTRACT-MIN() and DECREASE-KEY() operations efficiently [1]; indeed the technique used for implementing the priority queue, more or less determines the complexity of the Dijkstra implementation. The three common techniques of priority queue implementation are arrays, binary heaps and Fibonacci heaps. A Dijkstra implementation using Fibonacci heaps runs in $O(m + n \cdot \log n)$ time, on a graph with n vertices and m edges and this is the best that one can hope for in a comparison-based model using the Dijkstra approach, since Dijkstra's algorithm sorts the vertices in terms of their actual distances from the source. However, Fibonacci heaps are notoriously difficult to implement and analyze; consequently, there is sufficient interest in investigating whether simpler alternatives providing comparable performance exist. This paper answers that question in the affirmative by detailing a modification of binary heaps called Two-Level Heaps (TLH) that are simple to visualize, analyze and implement and yet outperform Fibonacci heaps on all graph families that we considered. Our results are particularly surprising, since the asymptotic complexity of Dijkstra's algorithm using TLHs is worse than the complexity using Fibonacci heaps.

The principal contributions of this paper are as follows:

(a) The introduction of a new data structure called Two-Level Heaps (TLHs); although this structure was invented primarily for Dijkstra's algorithm, it can be used in any situation where a priority queue is required.
(b) The introduction of the Phased-Dijkstra algorithm; although asymptotically no better than simple Dijkstra, it performs better empirically.

2 Preliminaries

We assume that we are given a graph $\mathbf{G} =< V, E, s >$ with V denoting the vertex set, E denoting the edge set and s denoting the source vertex. Associated with each edge is it The basic Dijkstra approach is described by Algorithm 2.1.

The RELAX() operation is described by Algorithm 2.2.

There are two important operations that determine the complexity of Algorithm 2.1, viz., the EXTRACT-MIN() operation, which is performed once per vertex, and the RELAX() operation, which is performed once per edge. The latter operation is performed through a DECREASE-KEY() operation on the appropriately defined priority queue. Accordingly, the running time of Algorithm 2.1 is $n * T_E + m * T_D$, where T_E denotes the time required for an EXTRACT-MIN() operation and T_D denotes the time required for a DECREASE-KEY() operation. When Algorithm 2.1 completes execution, the shortest path distances are stored in the $d[]$ array. Typical implementations include a storage structure to represent the Shortest Path Tree (SPT), but we will not be concerned with the SPT.

Function SINGLE-SOURCE-SHORTEST-PATH ($\mathbf{G} =< V, E, s >$)
1: $S \leftarrow \phi$; $Q \leftarrow V$.
2: {S contains the vertices whose shortest paths from the source have been determined, while Q is the queue containing the remaining vertices. The actual distance of a node $v \in V$ from the source s is stored in $d[v]$, while its parent in the current Shortest Path Tree is stored in $\pi[v]$. The graph $\mathbf{G}$ itself is stored in adjacency list form, with the nodes that are adjacent to node u being stored in a linked list $Adj[u]$. If vertex v is on vertex u's adjacency list, then $c(u, v)$ denotes the cost of the edge from u to v. }
3: **for** (each vertex $v \in \mathbf{V}$) **do**
4: $\pi[v] =$ NIL
5: $d[v] = \infty$
6: **end for**
7: $\pi[s] = s$
8: $d[s] = 0$
9: **while** ($Q \neq \phi$) **do**
10: $r \leftarrow$ EXTRACT-MIN(Q)
11: {It is the structure of the priority-queue Q that determines the efficiency of the update operations in Dijkstra's algorithm.}
12: $S \leftarrow S \cup \{r\}$
13: **for** (each vertex $v \in Adj[r]$) **do**
14: RELAX(r, v)
15: **end for**
16: **end while**

Algorithm 2.1. Dijkstra's Algorithm

3 Phased Dijkstra

We build on the ideas of the last section to develop a phase-based implementation of Dijkstra's algorithm. This implementation requires the following data structures:

(i) A heap structure H, and
(ii) A current array A.

We first run a Breadth-First search (BFS) on G from the source s and update the $d[\ ]$ values of vertices using this search. Note that the BFS updates the

Function RELAX (u, v)
1: **if** ($d[v] > d[u] + c(u, v)$) **then**
2: $d[v] = d[u] + c(u, v)$
3: $\pi[v] = u$
4: **end if**

Algorithm 2.2. The RELAX Procedure

distance labels of vertices in the order that vertices are seen. Accordingly, the distance label of a vertex is set exactly once. The $d[\ ]$ values returned by the BFS serve as a first approximation to the true shortest path values. The vertices are then organized as a priority queue H. This is followed by extracting the $\log n$ smallest elements in H and placing them in A. The idea behind the BFS is that in many representative families (especially sparse graphs), we get an approximation which is very close to the actual shortest path distances to many of the vertices in the graph. Thus, the number of queue operations is reduced.

An EXTRACT-MIN() operation is performed by searching through the elements of A; this can be done in $\log n$ time. If the array A is emptied as a consequence of the EXTRACT-MIN(), then an additional $\log n$ elements are moved from H into A.

Let us now study the DECREASE-KEY() operation, which involves the following steps:

(a) If the element whose key is decreased is in A, then decreasing the key is an $O(1)$ operation.
(b) If the element whose key is decreased is in H, then the corresponding element could become the smallest element in H. In this case, the element is compared with the largest element in A and an appropriate exchange is made.

Clearly the time taken for a DECREASE-KEY() operation is proportional to the time required for an EXTRACT-MIN() operation on H. We thus see that from an asymptotic perspective, the phased Dijkstra approach is no better than the regular Dijkstra algorithm described in Section 2.

4 Two-Level Binary Heaps

As described in Section 2, there are n EXTRACT-MIN() calls and m DECREASE-KEY() calls in the basic flow of Dijkstra's algorithm. We now propose the following Two-Level binary heap, which permits implementations of these operations which are more efficient than the traditional Binary Heap.

Assume that the n vertices are divided into $\log^k n$ min-heaps, each containing $\frac{n}{\log^k n}$ elements[1]. The minimum elements of these heaps are then further organized as a linked list D.

Note that there are at most $n/\log^k n$ elements in each of the min-heaps and at most $\log^k n$ elements in D.

To perform an EXTRACT-MIN() operation on this structure, we first determine the smallest element in D; this takes $\log^k n$ time. This element is removed from D as is its mirror in the corresponding min-heap, say H. An element from H is then bubbled to the top and into D. Clearly the total number of steps is $\log^k n + \log \frac{n}{\log^k n}$, which simplifies to $\leq 2 \cdot \log^k n$. Note that for an EXTRACT-MIN() operation, $\log^k n$ comparisons *must* be carried out.

[1] We use $\log^k n$ to represent $(\log n)^k$, as described in [1].

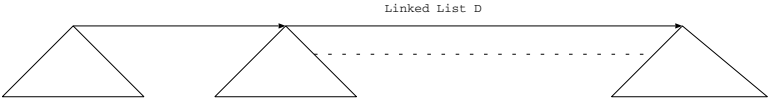


Fig. 1. Two-Level heap

Consider a DECREASE-KEY() operation; in the worst-case, an element in one of the min-heaps could bubble all the way to the top replacing its current representative in D; the time taken for this operation is $\log \frac{n}{\log^k n}$.

Thus the total number of operations taken by Algorithm 2.1 is:

$$f(k) = 2n \cdot \log^k n + m \cdot [\log \frac{n}{\log^k n}] \tag{1}$$

Optimizing for k, we get

$$k = \frac{\log[\frac{m \log\log n}{2n \ln \log n}]}{\log \log n} \tag{2}$$

Note that even at $k = 1$, our data structure beats simple min-heaps, so that for the value of k calculated above, we should do much better. It is very important to note that for each input graph, the value of k has to be recalculated and the data structure has to be built accordingly. Secondly, for very sparse graphs, it is possible that the optimal value of k is negative; in this case, we choose $k = 1$.

5 Performance Analysis

5.1 Experimental Setup

Our test platform for performance results is a 2.8 GHz 32-bit Intel Xeon machine with 4GB memory, 512KB cache and running RedHat Enterprise Linux 4. We build our codes with the Intel C compiler (icc) Version 9.0 and the optimization flag `-O3`.

We report the execution time of a baseline Breadth-First Search implementation for comparison with our shortest path code running times. The BFS running time is a natural lower bound for our SSSPP codes and is a good indicator of how close to optimal the shortest path running times are. It also removes dependencies of low-level implementation details and cache effects. Further, it gives us an estimate of the initialization time in the phased Dijkstra algorithm.

We represent the graph (vertex degree and adjacencies) using cache-friendly adjacency arrays [7], which takes optimal $O(m+n)$ space. Note that the ordering of vertex IDs determines how the graph is traversed. For the synthetic graph instances we experiment with, we randomly permute the vertex IDs so that there is no locality in the input graph. For each test instance, we choose five shortest path source vertices randomly. For each source vertex, we determine the shortest path tree and distances ten times, and compute the average execution time, removing the best and worst cases.

Our two-level binary heap implementation works for both directed and undirected instances. We use eight bytes to represent the edge weight, which can be either an positive integer or a positive real number.

5.2 Problem Families

We study performance on graph instances from several different families. However, we do not include dense graphs where $m = \Omega(n^2)$ in our experiments for two reasons: (a) Dijkstra's algorithm with an array for a priority queue is actually a linear time algorithm in this case (b) dense graphs for the graph sizes that we studied do not fit into the DRAM memory of our computer.

The synthetic graph families we experiment with are listed below. Some of the generators and graph instances are part of the 9th DIMACS Shortest Path Implementation Challenge benchmark package [8].

Random graphs: We generate graphs according to the Erdos-Renyi random graph model, and ensure that the graph is connected. The generator may produce parallel edges as well as self-loops. The ratio of the number of edges to the number of vertices, and the weight distribution, can be varied. We experiment with the following variants of random graphs:

- *iRandom-n*: Integer weight edges, the maximum weight is set to n, m is set to $4n$. n increases by a factor of two for one set of parameter values to the next. 2^{21}. Maximum weight is varied from 2^8 to 2^{20}, in multiples of sixteen.
- *rRandom-n*: Real weighted edges uniformly chosen from [0, 1], m is set to $4n$. n increases by a factor of two for one set of parameter values to the next.
- *iRandom-m*: Integer weight edges, the maximum weight is set to n, m is varied from n to $n\sqrt{n}$.

Mesh graphs: These are synthetic two-dimensional meshes with grid dimensions x and y. We generate *Long* grids where $x = \frac{n}{16}$ and $y = 16$ for this study. The diameter of these graphs is significantly higher than random graphs, and the degree distribution is uniform. We define the graph families (*iLong-n* and *rLong-n* similar to the Random graph family.

Small-world networks: We use the Recursive MATrix (R-MAT) [9] random graph generation algorithm to generate input data sampled from a Kronecker product that are representative of real-world networks with a small-world topology. As in the above random graph classes, we set $m = 4n$, and experiment with both integer (iSW) and real-weighted (rSW) graphs. Small-world networks have a low diameter and an unbalanced degree distribution.

The graph generators we use write the graphs to disk in plain text format, which the shortest path implementations parse and load to memory. The graph generation and loading steps are not timed.

5.3 Shortest Path Implementations

The worst case complexity of Dijkstra's algorithm depends on the priority queue implementation used to store the visited vertices. Cherkassky et al. [10] conducted an extensive experimental evaluation of data structures for solving the

shortest paths problem with non-negative integral edge weights. We use the code from the SPLIB library [11] (that implements the data structures discussed in [10]) for comparison with our two-level binary heap data structure. Note that both our implementation and SPLIB use the same internal representation for the graph (adjacency arrays), are coded in C, compiled with the same compiler (Intel C compiler) and optimization flags, and execution time results are obtained with an identical experimental test setup. We make straight-forward modifications to the Fibonacci Heap and Binary Heap implementations in SPLIB to handle real edge weights.

We compare the performance of Dijkstra's algorithm for graphs with both real and integer weights using the following priority queue representations:

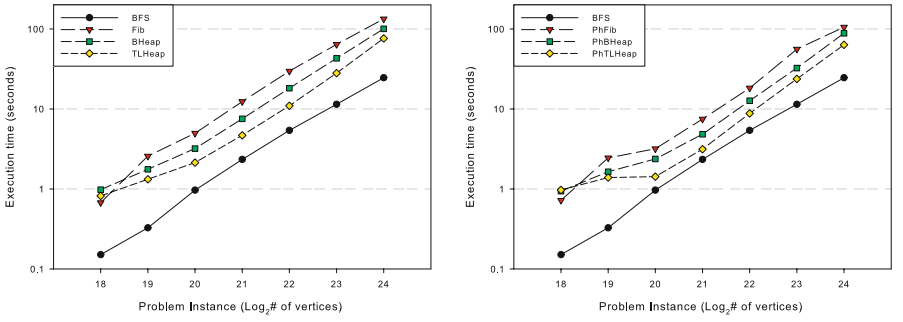
- *Queue*: storing the visited vertices in an array, which results in a worst case complexity of $O(n^2)$.
- *BHeap*: using a binary heap that runs in $O(m \log n)$ time.
- *Fib*: using Fibonacci heaps, that has a worst case $O(m + n \log n)$ complexity.
- *TLHeap*: our two-level heap data structure.

We also evaluate the performance of the Phased Dijkstra algorithm using all the priority queue representations. We denote the implementations *PhBHeap*, *PhFib* and *PhTLHeap*.

5.4 Experimental Results

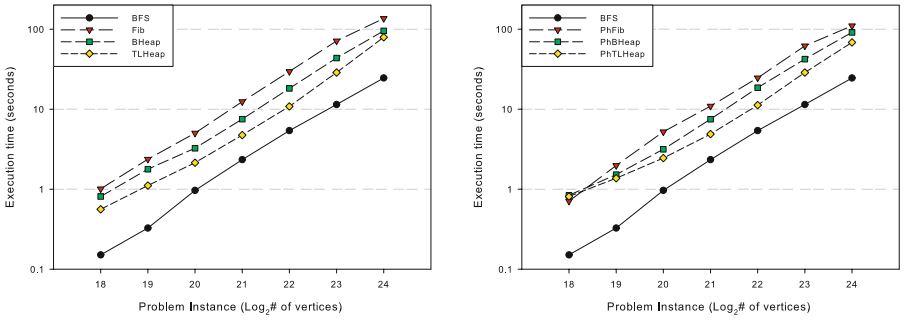
For each graph family, we present two plots, one corresponding to Dijkstra's algorithm implementations and the second for phased Dijkstra implementations with the same data structures. Figure 2 gives the performance of the various data structures for the *iRandom-n* family, where the number of vertices varied by two orders of magnitude from 2^{18} to 2^{24}. We find that *TLHeap* outperforms the other implementations for problem instances where n is greater than 2^{20}. On an average, the *TLHeap* implementation is 1.6× faster than *BHeap* and 2.64× faster than *Fib*. The performance of the phase-based Dijkstra implementations for the heap and two level-heap implementations are comparable to the regular Dijkstra implementations. However, the phase-based Dijkstra with a Fibonacci heap priority queue implementation performs better than *Fib* (a 20% improvement for the largest problem size). Note that in case of the *iRandom* graph family, the number of vertices in the priority queue is typically large. Our heuristic of maintaining a current array in the phased Dijkstra implementation improves the performance of the Fibonacci heap implementation. As the problem size increases, the performance of the Fibonacci heap implementation does not scale as well as the other data structures. This can be attributed to the pointer-intensive nature of the Fibonacci heap data structure, which leads to poor cache performance in comparison to the other implementations.

In Figure 3, we observe trends identical to integer-weighted case. For large networks, *TLHeap* performs better than the other implementations. Phased Dijkstra results in a minor 5% improvement in performance of *TLHeap* and *BHeap*



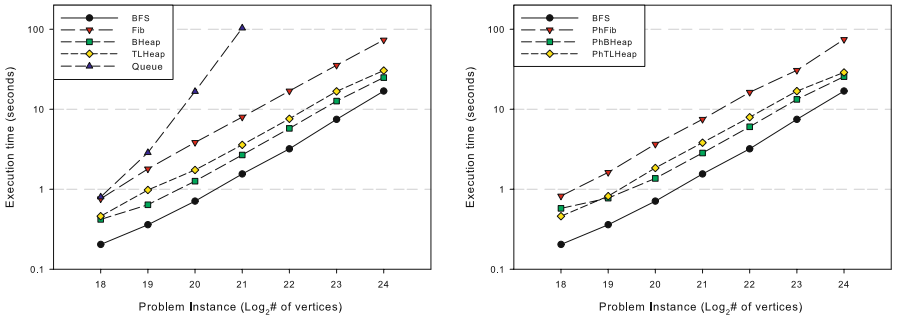
(a) (b)

Fig. 2. Execution time for graph instances from the *iRandom-n* family. n is varied from 2^{18} to 2^{24}, and m is set to $4n$.



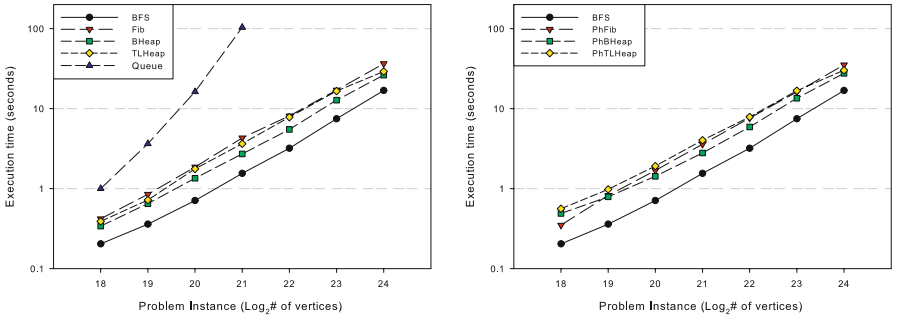
(a) (b)

Fig. 3. Execution time for graph instances from the *rRandom-n* family. n is varied from 2^{18} to 2^{24}, and m is set to $4n$.



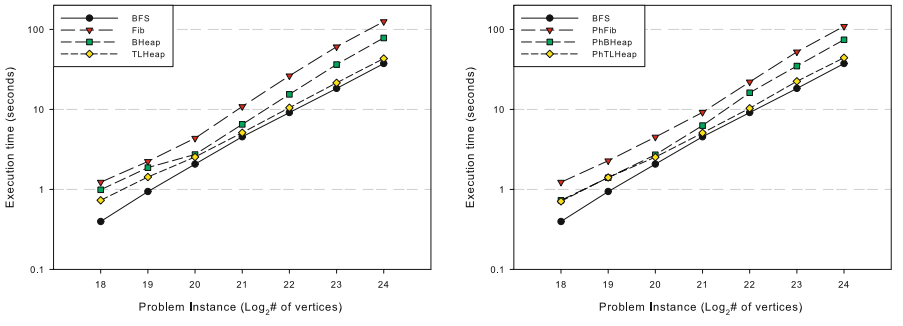
(a) (b)

Fig. 4. Execution time for graph instances from the *iLong-n* family. n is varied from 2^{18} to 2^{24}, and m is set to $4n$.



(a) (b)

Fig. 5. Execution time for graph instances from the *rLong-n* family. n is varied from 2^{18} to 2^{24}, and m is set to $4n$.



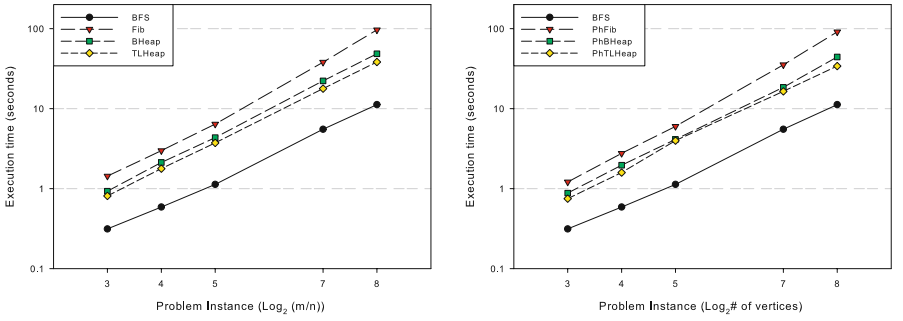
(a) (b)

Fig. 6. Execution time for graph instances from the *iSW-n* family. n is varied from 2^{18} to 2^{24}, and m is set to $4n$.

for this family. The results show that our implementations perform equally well on graphs classes with integer and real-weighted edges.

Figure 4 and 5 give the performance of our implementations on integer and real-weighted mesh networks respectively. These are high diameter graphs, and the number of visited vertices in the priority queue is comparatively smaller than the Random-n family of graphs. In this case, the running times of *BHeap* is nearly twice as slow as the reference BFS. *TLHeap* is slightly slower than *BHeap*, as the overhead for having two levels is not justified with a low number of vertices in the priority queue in practice. The performance of *Fib* is comparable to *TLHeap*. For smaller problem sizes, the running time of *Queue* may be comparable to the other implementations. There are also no significant gains in using the phase-based Dijkstra for any of the data structures.

Figure 6 gives the performance of the implementations for synthetic small-world networks with integer-weighted edges. These are typically low diameter



(a) (b)

Fig. 7. Execution time for graph instances from the *iRandom-m* family. n is set to 2^{17}, and m is varied from 2^{20} to 2^{25}.

graphs, and we expect the performance to be similar to the *Random-n* family. *TLHeap* and *PhTLHeap* outperform the other Dijkstra-based implementations, and the performance of *PhFib* is significantly better than *Fib*. The running time of *TLHeap* is on an average 1.23× the time taken for BFS, which is the best result across all graph families.

Next, we vary the network sparsity for the random graph family. Figure 7 plots the performance as m is varied across two orders of magnitude, keeping n fixed. We observe the same trends as in the previous case of the *iRandom-n* family. As m increases, the performance of *TLHeap* and *BHeap* is very similar, and they are faster than *Fib*.

6 Conclusion

Our work in this paper was motivated primarily by the need to find a simpler alternative to Fibonacci heaps. Towards this end, we designed the Two-Level Heap structure, which is easy to implement and analyze. When the data structure was designed, we expected that its performance over Dijkstra computations would be comparable (and perhaps somewhat inferior) to the performance of Fibonacci heaps. Our empirical results seem to indicate that not only are Two-Level heaps comparable to Fibonacci heaps, but that they are far superior to the same. This observation is very surprising, considering that Two-Level heaps are in fact asymptotically inferior to Fibonacci heaps over Dijkstra computations. We also note that Two-Level heaps are stand alone structures and could conceivably be used as priority queues in more general applications; consequently, a more detailed study of this structure is merited.

Acknowledgments

The research of the first author was conducted in part at the Discrete Algorithms and Mathematics Department, Sandia National Laboratories, Albuquerque, New

Mexico. Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
2. Goldberg, A.V.: Scaling algorithms for the shortest paths problem. SIAM Journal on Computing 24(3), 494–504 (1995)
3. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
4. Raman, R.: Recent results in single-source shortest paths problem. SIGACT news 28, 81–87 (1997)
5. Ahuja, R.K., Mehlhorn, K., Orlin, J.B., Tarjan, R.E.: Faster algorithms for the shortest path problem. Journal of the ACM 37(2), 213–223 (1990)
6. Pallottino, S.: Shortest path methods: Complexity, interrelations and new propositions. NETWORKS: Networks: An International Journal 14, 257–267 (1984)
7. Park, J., Penner, M., Prasanna, V.: Optimizing graph algorithms for improved cache performance. In: Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2002), Fort Lauderdale, FL (April 2002)
8. Demetrescu, C., Goldberg, A., Johnson, D.: 9th DIMACS implementation challenge – Shortest Paths, http://www.dis.uniroma1.it/~challenge9/
9. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A recursive model for graph mining. In: Proc. 4th SIAM Intl. Conf. on Data Mining, Florida, USA (April 2004)
10. Cherkassky, B., Goldberg, A., Radzik, T.: Shortest paths algorithms: theory and experimental evaluation. Mathematical Programming 73, 129–174 (1996)
11. Goldberg, A.: Network optimization library, http://www.avglab.com/andrew/soft.html

On Construction of Almost-Ramanujan Graphs*

He Sun[1] and Hong Zhu[2]

[1] Shanghai Key Laboratory of Intelligent Information Processing
School of Computer Science
Fudan University, Shanghai, China
sunhe@fudan.edu.cn
[2] Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, Shanghai, China
hzhu@fudan.edu.cn

Abstract. Reingold et al. introduced the notion zig-zag product on two different graphs, and presented a fully explicit construction of d-regular expanders with the second largest eigenvalue $O(d^{-1/3})$. In the same paper, they ask whether or not the similar technique can be used to construct expanders with the second largest eigenvalue $O(d^{-1/2})$. Such graphs are called Ramanujan graphs. Recently, zig-zag product has been generalized by Ben-Aroya and Ta-Shma. Using this technique, they present a family of expanders with the second largest eigenvalue $d^{-1/2+o(1)}$, what they call almost-Ramanujan graphs. However, their construction relies on local invertible functions and the dependence between the big graph and several small graphs, which makes the construction more complicated.

In this paper, we shall give a generalized theorem of zig-zag product. Specifically, the zig-zag product of one "big" graph and several "small" graphs with the same size will be formalized. By choosing the big graph and several small graphs individually, we shall present a family of fully explicitly almost-Ramanujan graphs with locally invertible function waived.

1 Introduction

Expanders are graphs of low-degree and high connectivity. There are several ways to measure the quantity of expansion. Geometrically, every subset of vertex set has "many" neighbors. Algebraically, we consider the adjacency (or Laplacian) matrix of the graph and expanders are those graphs whose second largest eigenvalue of adjacency matrix are small. From the probabilistic viewpoint, expanders can be considered as the time reversible Markov chain which has low mixing time.

Since the first construction of expanders [8], there has been a number of expander constructions using number theory [7,8,9]. On the other hand, expanders have been shown a powerful tool in Theoretical Computer Science and can be

* The research was supported by Shanghai Leading Academic Discipline Project with Number B412 and B114, Research and Development Project of High-Technology of China with Number 2007AA01Z189.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 197–207, 2009.

used to reduce the need of randomness, find good error-correcting codes, construct new proofs of PCP Theorem, derandomize some complexity classes, and so on. This line of research leads to a number of combinatorial construction for expanders. Ajtai [1] presented a combinatorial construction of cubic expanders. However, his construction leaked any simply described form as well as the explicitness level of the algebraic construction. Reingold et al. [10] proposed the notion zig-zag product. Using this product as well as the replacement product, which is widely used in Graph Theory, they constructed a family of expanders. Relying on revised replacement products, Alon et al. [2] constructed constant-degree expanders. In a very different setting, Reingold [11] used zig-zag product and graph powering to reduce any graph into expanders. Using this technique, he presented a logarithmic space algorithm for solving UNSTCON problem, which implies SL=L in Complexity Theory, and was open for many years.

In the following, we use the second largest eigenvalue $\lambda_2(G)$ associated with the normalized adjacency matrix of the graph G to characterize the algebraic expansion. Furthermore, any d-regular graph satisfying $\lambda_2(G) \leq 2\sqrt{d-1}/d$ is called *Ramanujan graphs*. Regarding this topic, Lubotzky et al. [7] firstly constructed a family of Ramanujan graphs. Following their work, there are several constructions on Ramanujan graphs, see [9] for example.

On the other hand, it has been shown that there exist expanders with better parameters [12], though the explicit construction is open. Friedman [5] showed that almost all the random graphs are Ramanujan graphs. Relying on zig-zag products and const-size Ramanujan graphs, Reingold et al. [10] obtained a family of d-regular expanders with the second largest eigenvalue $O(d^{-1/3})$. In the same paper, they ask whether or not the similar technique can be used to construct expanders with the optimal eigenvalue $O(d^{-1/2})$. Very recently, Ben-Aroya et al. [4] presented a fully constructible family of d-regular expanders $\{G_i\}$ satisfying $\lambda_2(G_i) \leq d^{-1/2+o(1)}$, what they call almost-Ramanujan graphs.

Here we call an N-vertex d-regular graph explicitly constructible if the construction algorithm satisfies the following three properties: (1) We can build the entire graph in poly(N) time; (2) From a vertex v, we can find the i-the neighbor within running time poly$(\log N, \log d)$; (3) Given vertices u and v, we can determine whether or not they are adjacent in poly$(\log N)$ time.

Our Results: In this paper, we shall generalize the standard definition of zig-zag product proposed in [10]. Compared with Ben-Aroya et al.'s definition [4], our generalized zig-zag product is more natural and allows us to choose the big graph as well as several small graphs independently. Employing this new zig-zag product, we shall present a family of fully explicitly constructible almost-Ramanujan graphs. Compared with the previously best construction, our method is simpler, and makes the iterative construction more efficient.

More specifically, our construction shall be divided into four steps: (1) Choose a set of small graphs $\overline{H} = (H_1, \cdots, H_k)$ from G_{N_2,D_2} uniformly and independently, where N_2 and D_2 are constant numbers. Verify that each graph satisfies the given spectral upper bound. (2) Choose constant-size graphs G_1 and G_2. (3) Let $G_{t+1} = \left(G_{\lfloor \frac{t-1}{2} \rfloor} \otimes G_{\lceil \frac{t-1}{2} \rceil}\right)^2 \textcircled{z} \overline{H}$. (4) Iteratively construct the graph using

(3) to obtain the family $\{G_i\}$. Our analysis shall show that each graph in $\{G_i\}$ is almost-Ramanujan.

2 Preliminaries

Notations: We use $[N]$ to express the set consisting of N elements. $G_{N,d}$ is defined as the set of d-regular graphs with N vertices. For any vector α, $|\alpha|_1 := \sum_{i=1}^{n} |\alpha_i|$. For vectors $\alpha \in \mathbb{R}^{N_1}$ and $\beta \in \mathbb{R}^{N_2}$, the tensor product of α and β is the vector $\alpha \otimes \beta \in \mathbb{R}^{N_1 \cdot N_2}$ whose (i,j)'s entry is $\alpha_i \cdot \beta_j$. If A is an $N_1 \times N_1$ matrix and B is an $N_2 \times N_2$ matrix, then $(A \otimes B)(\alpha \otimes \beta) = (A\alpha) \otimes (B\beta)$ for all α and β.

In addition, $\mathbb{S}_\Lambda$ represents the permutation group over Λ.

Spectral Expansion: For the given undirected d-regular graph G with normalized adjacency matrix M, denote λ_2 be the second largest eigenvalue of M. Furthermore, a graph is said to have spectral expansion λ if and only if $\lambda_2 \leq \lambda$. In the following we use (N, d, λ) to express a d-regular graph consisting of N vertices and the second largest eigenvalue is bounded by λ. On tensor product, it has been shown in [10] that if G_1 is an (N_1, D_1, λ_1)-graph and G_2 is an (N_2, D_2, λ_2)-graph, then $G_1 \otimes G_2$ is an $(N_1 \cdot N_2, D_1 \cdot D_2, \max(\lambda_1, \lambda_2))$-graph.

It is well known that λ is a key quantity to estimate the "randomness" of graph G, and also has close relationship with chromatic number, the size of independent set, and some other combinatorial quantities. For those who are not familiar with this topic, we refer to the excellent surveys [2,6].

It is easy to show that for any connected non-bipartite graph G, $\lambda(G) < 1$. On the other hand, N. Alon et al. proved that for any d-regular n-vertex graph $\lambda \geq 2\sqrt{d-1}/d - o_n(1)$, where $o_n(1)$ term is a quantity that tends to zero for every fixed d as $n \to \infty$. Thus, any d-regular graph G satisfying $\lambda(G) \leq \lambda_{\text{Ram}} := 2\sqrt{d-1}/d$ is called *Ramanujan Graphs*.

Theorem 1 ([5]). *For every $\delta > 0$ and for every even d, there exists a constant $c > 0$, independent of N, such that*

$$\Pr_{G \sim G_{N,d}}[\lambda(G) \geq \lambda_{\text{Ram}} + \delta] \leq c \cdot N^{-\lceil(\sqrt{d-1}+1)/2\rceil - 1}.$$

Graph Powering: Graph powering in our paper is used to decrease the spectral expansion, with the cost of increasing degree. By the basic Linear Algebra, we know that if G is an (N, d, λ)-graph, then G^k, the k-th powering of the adjacency matrix of G, represents an (N, d^k, λ^k)-graph. From the rotation map's perspective, if G is a d-regular graph with rotation map Rot_G, then the k-th powering of G is a d^k-regular graph whose rotation map is given by $\text{Rot}_{G^k}(v_0, (a_1, \cdots, a_k)) = (v_k, (b_k, \cdots, b_1))$, where the values $b_1, \cdots, b_k$ and v_k are computed via the rule $(v_i, b_i) = \text{Rot}_G(v_{i-1}, a_i)$.

Replacement Product: Let $G = (V, E)$ be a d-regular graph. Assume that for every $v \in V$, its d edges are labeled from 1 to d. Let $v[i]$ be the i-th neighbor of

v in G. The rotation map of G, $\mathrm{Rot}_G : V \times [d] \mapsto V \times [d]$, is defined as follows: $\mathrm{Rot}_G(v,i) = (u,j)$ if $v[i] = u[j]$.

For D-regular graph G with N vertices and d-regular graph H with D vertices, the replacement product, denoted as $G \textcircled{r} H$, is a $(d+1)$-regular graph with $N \cdot D$ vertices. Each vertex in G is replaced by graph H, called *a cloud*. Moreover, $\mathrm{Rot}_{G \textcircled{r} H}((u,k),i) = ((v,\ell),j)$ if and only if $u = v$ and $\mathrm{Rot}_H(k,i) = (\ell,j)$, or $i = j = d+1$ and $\mathrm{Rot}_G(u,k) = (v,\ell)$.

Zig-Zag Product: Relying on rotation maps, zig-zag product is defined in the following way.

Definition 1 ([10]). *If G is a D-regular graph on $[N]$ with rotation map Rot_G and H is a d-regular graph on $[D]$ with rotation map Rot_H, then their zig-zag product $G \textcircled{z} H$ is defined to be the d^2-regular graph on $[N] \times [D]$ whose rotation map $\mathrm{Rot}_{G \textcircled{z} H}$ is as follows: (1) Let $(a', i') = \mathrm{Rot}_H(a,i)$; (2) Let $(w,b') = \mathrm{Rot}_G(v,a')$; (3) Let $(b,j') = \mathrm{Rot}_H(b',j)$; (4) Output $((w,b),(j',i'))$ as the value of $\mathrm{Rot}_{G \textcircled{z} H}((v,a),(i,j))$.*

Zig-zag product corresponds to 3-step walks on the replacement product graph, where the first and the last steps are in the inner-cloud edges and the middle step is an inter-cloud edge, and each vertex in the cloud corresponds to an edge starting from the vertex which the cloud represents. O. Reingold et al. [10] showed that $\lambda(G \textcircled{z} H)$ was bounded by $\lambda(G)$ and $\lambda(H)$. Formally, if G is an (N, D, λ_1)-graph and H is a (D, d, λ_2)-graph, then $\lambda(G \textcircled{z} H)$ is an $(N \cdot D, d^2, f(\lambda_1, \lambda_2))$-graph where

$$f(\lambda_1, \lambda_2) = \frac{1}{2}(1-\lambda_2^2)\lambda_1 + \frac{1}{2}\sqrt{(1-\lambda_2^2)^2\lambda_1^2 + 4\lambda_2^2}.$$

It is easy to show that if $\lambda_1 < 1, \lambda_2 < 1$ then $f(\lambda_1, \lambda_2) < 1$. Thus zig-zag product can be used to reduce two expanders into a new expander. In that paper, two variations of zig-zag product have also been discussed. In 2008, this standard definition of zig-zag product, as shown in Definition 1, has been generalized by Ben-Aroya et al. [4] in the following way. Instead of taking a random walk on a big graph G and two steps on the same small graph H, A. Ben-Aroya et al. considered the zig-zag product on one big graph G and several small regular graphs $\overline{H} = (H_1, \cdots, H_k)$ with the same vertex number. However, the analysis of the zig-zag product described in their paper relies on the notion *local invertible function*, which makes their construction more complicated and lacks the generally well-understood description. Moreover, in order to compute this kind of zig-zag product, one have to deal with directed expanders and convert such expanders into undirected ones.

3 Generalized Zig-Zag Product

In this section, we shall present a generalized zig-zag product. Compared with the revised zig-zag product presented by A. Ben-Aroya et al., a more natural

generalization of standard zig-zag product [10] will be studied. We start our discussion with the notion *local inversion function*, which is used and plays a critical role in A. Ben-Aroya et al.'s construction.

Definition 2 ([4]). *A d-regular graph G is locally invertible if its rotation map is of the form $\mathrm{Rot}_G(v,i) = (v[i], \psi(i))$ for some permutation $\psi : [d] \mapsto [d]$. We say that ψ is the local inversion function.*

Lemma 1. *Every d-regular graph $G = (V, E)$ is locally invertible.*

Proof. First of all, we consider the case that the vertex number of $G = (V, E)$ is even. In such case, G can be considered as the union of d perfect matchings on V. For each perfect matching, we label the edges with respect to each vertex the same number. Thus the identity mapping $\phi : [d] \mapsto [d]$ is the local inversion function.

On the other hand, if $|V|$ is odd, then the degree d is even and G can be seen as the union of $d/2$ edge-disjoint cycles. For the i-th cycle, $1 \leq i \leq d/2$, the edges with respect to each vertex is labeled by $2i-1$ and $2i$ respectively. Thus the local inversion function ψ can be written as

$$\psi(i) = \begin{cases} i+1 & i \text{ is odd} \\ i-1 & i \text{ is even.} \end{cases}$$

Thus every regular graph is locally invertible. □

Without loss of generality, we assume that the number of vertices for each graph is even. However, as described in the following, this local inversion function of the lemma above will be used throughout this paper. Actually this function suffices to construct almost-Ramanujan graphs.

Now we turn to show the generalized definition of zig-zag product. The input is as follows:

- A "big" graph $G_1 = (V_1, E_1)$ with parameter (N_1, D_1, λ_1).
- k "small" graphs $\overline{H} = (H_1, \cdots, H_k)$, where each graph H_i is an (N_2, D_2, λ_2)-graph over the vertex set V_2 such that $|V_2| = N_2 = D_1^{4k}$.

Let the resulting graph be G, a D_2^k-regular graph with vertex number $|V_1| \times |V_2|$. Each edge in G is expressed by $\overline{i} = (i_1, \cdots, i_k) \in [D_2]^k$. For any vertex $v = (v^{(1)}, v^{(2)}) \in V_1 \times V_2$, $\mathrm{Rot}_G(v, \overline{i})$ is defined in the following way: We start from vertex $v = \left(v_0^{(1)}, v_0^{(2)}\right)$. For $j = 1, \cdots, 2k-1$, if j is odd, we set $t = (j+1)/2$ and take one step on $H_t(\cdot, i_t)$, *i.e.* $v_j^{(1)} = v_{j-1}^{(1)}$ and $(v_j^{(2)}, i_t) = \mathrm{Rot}_{H_t}(v_{j-1}^{(2)}, i_t)$. On the other hand, for even j, let $v_j^{(1)} = v_{j-1}^{(1)}\left[\pi(v_{j-1}^{(2)})\right]$ and $v_j^{(2)} = v_{j-1}^{(2)}$ where $\pi(v)$ is the first D_1 bits of v. Finally, $\mathrm{Rot}(v, \overline{i}) = \left((v_{2k-1}^{(1)}, v_{2k-1}^{(2)}), (i_k, \cdots, i_1)\right)$.

In summary, we begin with a D_1-regular graph over N_1 vertices (we think of D_1 as a constant and $N_1 = |V_1|$ as a growing parameter). We replace each D_1-degree vertex with a "cloud" of D_1^{4k} vertices, and map a cloud vertex to a D_1 instruction employing π. We then take a $(2k-1)$-step walk, with alternating

$\overline{H}$ and G_1 steps, over the resulting graph. Notice that, compared with zig-zag product [4], our definition only relies on one "big" graph as well as several "small" graphs. In this sense, we say our construction is a more natural generalization of standard zig-zag products.

4 Analysis

Instead of dealing with graphs G_1 and $\overline{H}$, we consider the vertex sequence induced by permutations. Intuitively, we want to prove that if every small graph is almost-Ramanujan, then such sequence of graphs as well as a big const-size graph can be used to construct the almost-Ramanujan graphs with increasing size. In order to achieve this goal, one has to deal with the vertex sequence induced by the resulting graph in the sense that almost-Ramanujan graphs, similarly with random graphs, have low mixing time, and the vertex sequence is like random ones after short steps over the resulting graph. To formulate this intuition, we use the following definitions as well as lemmas to analyze the spectral expansion of the resulting graph.

Definition 3 (Permutation sequence). *Let* $\gamma_1, \cdots, \gamma_{k-1} : V_2 \mapsto V_2$ *be permutations. Denote* $\overline{\gamma} = (\gamma_1, \cdots, \gamma_{k-1})$. *The permutation sequence* $\overline{q} = (q_0, \cdots, q_{k-1})$ *induced by* $\overline{\gamma}$ *is defined as follows:*

- $q_0(v^{(2)}) = v^{(2)}$.
- *For* $1 \leq i < k$, $q_i(v^{(2)}) = \gamma_i(q_{i-1}(v^{(2)}))$.

Notice that after graph G_1, the second component $v^{(2)}$ is not changed, whereas after a small graph H_i, corresponding to permutation γ_i, the second component $q_{i-1}(v^{(2)})$ becomes $\gamma_i(q_{i-1}(v^{(2)}))$. Thus $q_i(v)$ is the V_2 value after $(2i-1)$ steps, starting with $v^{(2)}$ and alternating between H_i and G_1.

We know that the behavior of Markov chain on a good expander is similar with randomly regular graphs, *i.e.* random walks on good expanders mix rapidly to the stable distributions. On the other hand, Theorem 1 states that almost all the random graphs are Ramanujan graphs. To characterize the randomness of the vertex sequence $\overline{q}$ induced by $\overline{H}$, we introduce the notion "ε-pseudorandomness" as well as "ε-good". Since G_1 can be considered as permutations, such operations do not change the randomness involving in $\overline{q}$.

Definition 4. *Let* $q_0, \cdots, q_{k-1} : V_2 \mapsto V_2$ *be the permutation sequence induced by* $\overline{\gamma} = (\gamma_1, \cdots, \gamma_{k-1})$. $\overline{\gamma}$ *is called* ε*-pseudorandom if*

$$\left| \pi(q_0(U)) \circ \cdots \circ \pi(q_{k-1}(U)) - U_{[D_1]^k} \right|_1 \leq \varepsilon,$$

where $\pi(q_0(U)) \circ \cdots \circ \pi(q_{k-1}(U))$ *is the distribution obtained by picking* $v^{(2)} \in V_2$ *uniformly at random and outputting* $\pi(q_0(U)) \circ \cdots \circ \pi(q_{k-1}(U))$ *and* $U_{[D_1]^k}$ *is the uniform distribution over* $[D_1]^k$.

Notice that every D_2-regular graph H_i can be considered as the union of D_2 permutations $\mathcal{H}_{i,1}, \cdots, \mathcal{H}_{i,D_2}$. For the i-th permutation, $1 \leq i \leq D_2$, we label all the edges i with respect to each vertex. From this way, it is easy to see that the local inversion function involving in the rotation map coincides with the one used in Lemma 1.

Definition 5 (ε-good). *Let $\overline{H} = (H_1, \cdots, H_k)$ be a k-tuple of D_2-regular graphs over V_2. We say $\overline{H}$ is ε-pseudorandom, if we can express each graph H_i as $H_i = \frac{1}{D_2}\sum_{j=1}^{D_2} \mathcal{H}_{i,j}$ such that*

- *$\mathcal{H}_{i,j}$ is the transition matrix of a permutation $\gamma_{i,j} \in \mathbb{S}_{V_2}$.*
- *For any $1 \leq \ell_1 \leq \ell_2 \leq k, j_{\ell_1}, \cdots, j_{\ell_2} \in [D_2]$, the sequence $\gamma_{\ell_1,j_{\ell_1}}, \cdots, \gamma_{\ell_2,j_{\ell_2}}$ is ε-pseudorandom.*

In addition, if for each $i = 1, \cdots, k$, we have $\lambda(H_i) \leq \lambda_{\mathrm{Ram}}(D_2) + \varepsilon$, we say that $\overline{H}$ is ε-good.

Now we follow the approach of [4] to show that with high probability the family of randomly picked graphs from G_{N_2,D_2} is ε-good.

Lemma 2 ([4]). *Let Ω be a universe and $S_1 \subseteq \Omega$ a fixed subset of size m. Let $S_2, \cdots, S_k \subseteq \Omega$ be uniformly random subsets of size m. Set $\mu_k = \mathbf{E}_{S_2,\cdots,S_k}\left[S_1 \cap \cdots \cap S_k\right] = m^k/|\Omega|^{k-1}$. Then for every $0 < \varepsilon \leq 1/4k$,*

$$\Pr_{S_2,\cdots,S_k}\left[\left||S_1 \cap \cdots \cap S_k| - \mu_k\right| \geq 2\varepsilon k\mu_k\right] \leq 2k\mathrm{e}^{-\varepsilon^2\mu_k/6}.$$

Lemma 3. *For every $\varepsilon > 0$ a sequence of uniformly random and independent permutations $\overline{\gamma} = (\gamma_1, \cdots, \gamma_{k-1})$ satisfies*

$$\Pr_{\gamma_1,\cdots,\gamma_{k-1}}\left[(\gamma_1, \cdots, \gamma_{k-1}) \text{ is not } \varepsilon\text{-pseudorandom}\right] \leq D_1^k \cdot 2ke^{-\Omega(\varepsilon D_1^{3k}k^{-2})}.$$

Proof. Let $q_0, \cdots, q_{k-1} : V_2 \mapsto V_2$ be the permutation sequence induced by $\overline{\gamma}$. Let A be the distribution $\pi(q_0(U)) \circ \cdots \circ \pi(q_{k-1}(U))$ and B the uniform distribution over $[D_1]^k$. Fix an arbitrary $\overline{r} = (r_1, \cdots, r_k) \in [D_1]^k$. For $1 \leq i \leq k$, define $S_i = \{x \in V_2 | \pi(q_i(x)) = r_i\}$. Since q_i is the permutation and π is regular, $|S_i| = |V_2|/D_1$, which implies that for each i, q_i is a random permutation distributed uniformly in $\mathbb{S}_{V_2}$. On the other hand, we know that $q_0, \cdots, q_{k-1}$ are independent. Define $\overline{A} = |S_1 \cap \cdots \cap S_k|/|V_2|$. Since

$$\mathbf{E}_{S_2,\cdots,S_k}\left[|S_1 \cap \cdots \cap S_k|\right] = \frac{(|V_2|/D_1)^k}{|V_2|^{k-1}} = \frac{|V_2|}{D_1^k} = D_1^{3k},$$

by Lemma 2 we have

$$\Pr_{r_1,\cdots,r_k}\left[|A(\overline{r}) - B(\overline{r})| \geq \varepsilon D_1^{-k}\right] \leq 2ke^{-\Omega(\varepsilon D_1^{3k}k^{-2})}.$$

Therefore the probability of event $\exists\overline{r}|A(\overline{r}) - B(\overline{r})| > \varepsilon D_1^{-k}$ is bounded by $D_1^k 2ke^{-\Omega(\varepsilon D_1^{3k}k^{-2})}$. On the other hand we have $|A-B|_1 \leq D_1^k \cdot \max_{\overline{r}}\{A(\overline{r}) - B(\overline{r})\}$ and with probability $1 - D_1^k 2ke^{-\Omega(\varepsilon D_1^{3k}k^{-2})}$ we have $|A - B|_1 \leq \varepsilon$. □

Theorem 2. *For every even $D_2 \geq 4$, there exists a constant B, such that for every $D_1 \geq B$ and every $k \geq 3$ the following holds: Set $N_2 = D_1^{4k}$ and $\varepsilon = D_2^{-k}$. Pick $\overline{H} = (H_1, \cdots, H_k)$ with each H_i sampled independently and uniformly from G_{N_2,D_2}. Then with high probability, $\overline{H}$ is ε-good.*

Proof. Pick $\overline{H} = (H_1, \cdots, H_k)$ with each H_i sampled independently and uniformly from G_{N_2,D_2}, *i.e.* let $\{\gamma_{i,j}\}_{i\in[k],j\in[D_2/2]}$ be a set of random permutations chosen uniformly and independently from $\mathbb{S}_{V_2}$. For $1 \leq i \leq k$, let H_i be the undirected graph over V_2 formed from the permutations $\{\gamma_{i,j}\}_{j\in[D_2]}$. Therefore for every $j_1, \cdots, j_k \in [D_2]$ the k-tuple $\overline{\gamma} = (\gamma_{1,j_1}, \cdots, \gamma_{k,j_k})$ is uniform in $(\mathbb{S}_{V_2})^k$. By Lemma 3,

$$\Pr\left[\overline{H} \text{ is not } \varepsilon - \text{pseudorandom}\right] \leq 2kD_1^k D_2^k e^{-\Omega(\varepsilon D_1^{3k} k^{-2})}.$$

Taking $\varepsilon = D_2^{-k} \geq D_1^{-k}$, the error term is at most $D_1^{3k} e^{-\Omega(\varepsilon D_1^{3k} k^{-2})}$.

On the other hand, by Theorem 1, the probability of choosing a graph H_i with $\lambda(H_i) \geq \lambda_{\mathrm{Ram}}(D_2) + \varepsilon$ is at most

$$c \cdot |V_2|^{-\lceil(\sqrt{D_2-1}+1)/2\rceil-1} \leq kcD_1^{-4k}.$$

Taking D_1 large enough and applying union bound, we obtain that with high probability $\overline{H}$ is ε-good. □

The rest of this section is devoted to prove the spectral expansion of the resulting graph $G = G_1 \circledz \overline{H}$.

Theorem 3. *If $G_1 = (V_1, E_1)$ be an (N_1, D_1, λ_1) graph, and $\overline{H} = (H_1, \cdots, H_k)$ be a ε-good sequence of $(N_2 = D_1^{4k}, D_2, \lambda_2)$ graphs where each graph H_i satisfies $\lambda(H_i) \leq 1/2$. Then $G = G_1 \circledz \overline{H}$ is an $\left(N_1 \cdot N_2, D_2^k, f(\lambda_1, \lambda_2, k)\right)$ graph for $f(\lambda_1, \lambda_2, k) = \lambda_2^{k-1} + 2(\varepsilon + \lambda_1) + \lambda_2^k$.*

In the resulting graph, each vertex in G_1 is replaced by H_i, thus the number of vertices in G is $N_1 \cdot N_2$. Notice that each edge in G is indexed by $i_1, \cdots, i_k$, where each $i_j \in [D_2], 1 \leq j \leq k$, thus G is D_2^k-regular. So it suffices to analyze the spectral gap of the resulting graph.

Following the approach of [4], we consider G_1 as a linear operator on a dim-N_1 vector space, and $\dot{G}$ an operator on a vector space $\mathcal{V}$ of dimension $N_1 \cdot N_2$ that is the adjacency matrix of the inter-clouds edges. Secondly, we let $\tilde{H} = I \otimes H$. In such case, the standard zig-zag product corresponds to $\tilde{H}\dot{G}\tilde{H}$ and our goal is to bound the second largest eigenvalue of $\tilde{H}\dot{G}\tilde{H}$. For the generalized case, we need to bound the spectral expansion of $\tilde{H}\dot{G}\cdots\dot{G}\tilde{H}$.

We decompose $\mathcal{V} = \mathcal{V}_1 \otimes \mathcal{V}_2$ to its parallel and perpendicular parts. $\mathcal{V}^{\|}$ is defined by $\mathcal{V}^{\|} = \mathrm{Span}\{v^{(1)} \otimes \mathbf{1} : v^{(1)} \in V_1\}$ and $\mathcal{V}^{\perp}$ is its orthogonal complement. Intuitively, each vector in $\mathcal{V}^{\|}$ corresponds to a probability distribution on the vertices of $G = G_1 \circledz \overline{H}$ such that the conditional distribution on the clouds are uniform, whereas the element in $\mathcal{V}^{\perp}$ corresponds to a distribution so that the conditional distribution on the clouds are all far from uniform. For any vector

$\tau \in \mathcal{V}$, we denote by $\tau^{\|}$ and $\tau^{\perp}$ the projections of τ on $\mathcal{V}^{\|}$ and $\mathcal{V}^{\perp}$ respectively. Define $x_i = \dot{G}_1 \tilde{H} x_{i-1}^{\perp}$ and $y_i = \dot{G}_1 \tilde{H}_{k-i+1} y_{i-1}^{\perp}$. Thus

$$y_0^{\mathrm{T}} \tilde{H}_k \dot{G}_1 \cdots \tilde{H}_2 \dot{G}_1 \tilde{H}_1 x_0^{\perp} = y_0^{\mathrm{T}} \tilde{H}_k \dot{G}_1 \cdots \tilde{H}_2 x_1,$$

which results in

$$\begin{aligned} & y_0^{\mathrm{T}} \tilde{H}_k \dot{G}_1 \cdots \tilde{H}_2 \dot{G}_1 \tilde{H}_1 x_0 \\ =& y_0^{\mathrm{T}} \tilde{H}_k x_{k-1}^{\perp} + \sum_{i=1}^{k} y^{\mathrm{T}} \tilde{H}_k \dot{G}_1 \cdots \tilde{H}_{i+1} \dot{G}_1 \tilde{H}_i x_{i-1}^{\|} \\ =& y_0^{\mathrm{T}} \tilde{H}_k x_{k-1}^{\|} + \sum_{i=1}^{k} \left(y_{k-i}^{\perp}\right)^{\mathrm{T}} x_{i-1}^{\|} + \sum_{i=1}^{k} \left(y_{k-i}^{\perp}\right)^{\mathrm{T}} x_{i-1}^{\|} + \\ & \sum_{1 \le i \le j \le k} \left(y_{k-i}^{\|}\right)^{\mathrm{T}} \dot{G}_1 \tilde{H}_{j-1} \cdots \tilde{H}_{i+1} \dot{G}_1 x_{i-1}^{\|} \end{aligned} \tag{1}$$

Lemma 4 ([4]). $\left|\left|y_0^{\mathrm{T}} \tilde{H}_k x_{k-1}^{\perp}\right|\right| \le \lambda_2^k$, $\sum_{i=1}^{k} \left(y_{k-i}^{\perp}\right)^{\mathrm{T}} x_{i-1}^{\|} = \lambda_2^{k-1}$.

Lemma 5 ([4]). $\sum_{i=1}^{k} \left(y_{k-i}^{\perp}\right)^{\mathrm{T}} x_{i-1}^{\|} = 0$.

So it suffices to give the upper bound of the last term in Eq. (1).

Lemma 6 ([4]). *Suppose $\overline{\gamma} = (\gamma_1, \cdots, \gamma_k)$ is ε-pseudorandom. Let $\tilde{\Gamma}_1, \cdots, \tilde{\Gamma}_k$ be the operators corresponding to $\gamma_1, \cdots, \gamma_k$. Any $\Gamma, \xi \in \mathcal{V}^{\|}$ can be written as $\tau = \tau^{(1)} \otimes \mathbf{1}$ and $\xi = \xi^{(1)} \otimes \mathbf{1}$. For any such τ, ξ,*

$$\left|\left\langle \dot{G}_1 \tilde{\Gamma}_k \dot{G}_1 \cdots \tilde{\Gamma}_1 \tau, \xi \right\rangle - \left\langle G^{k+1} \Gamma^{(1)}, \xi^{(1)} \right\rangle\right| \le \varepsilon ||\tau|| \cdot ||\xi||.$$

Lemma 7. *For every $\ell \ge 1$ and $\tau, \xi \in \mathcal{V}^{\|}$, $\tau, \xi \in \mathbf{1}_V$,*

$$\left|\left\langle \dot{G}_1 \tilde{\Gamma}_k \dot{G}_1 \cdots \tilde{\Gamma}_1 \tau, \xi \right\rangle\right| \le (\lambda_1^{\ell+1} + \varepsilon) ||\tau|| \cdot ||\xi||. \tag{2}$$

Proof. Since $\overline{H}$ is ε-good, we can express each H_i as $H_i = \frac{1}{D_2} \sum_{j=1}^{D_2} \mathcal{H}_{i,j}$ so that $\mathcal{H}_{i,j}$ is the transition matrix of a permutation $\gamma_{i,j} \in \mathbb{S}_{V_2}$ and each of the sequences $\gamma_{1,j_1}, \cdots, \gamma_{k,j_k}$ is ε-pseudorandom. Define $\Gamma_{i,j}$ be the operator on $\mathcal{V}_2$ corresponding to the permutation $\gamma_{i,j}$ and $\tilde{\Gamma}_{i,j} = I \otimes \Gamma_{i,j}$ be the corresponding operator on $\mathcal{V}_1 \otimes \mathcal{V}_2$. As shown in [4], we have

$$\left\langle \dot{G}_1 \tilde{H}_{i+\ell} \dot{G}_1 \cdots \tilde{H}_{i+1} \dot{G}_1 \tau, \xi \right\rangle = \mathbf{E}_{j_1, \cdots, j_\ell \in [D_2]} \left[\left\langle \dot{G}_1 \tilde{\Gamma}_{i+\ell, j_\ell} \dot{G}_1 \cdots \tilde{\Gamma}_{i+1, j_1} \dot{G}_1 \tau, \xi \right\rangle \right].$$

Since $\overline{H}$ is ε-good, every subsequence of $\overline{H}$ is ε-pseudorandom. Therefore

$$\left|\left\langle \dot{G}_1 \tilde{H}_{i+\ell} \dot{G}_1 \cdots \tilde{H}_{i+1} \dot{G}_1 \tau, \xi \right\rangle - \left\langle G^{\ell+1} \tau^{(1)}, \xi^{(1)} \right\rangle\right| \le \varepsilon \cdot ||\tau|| \cdot ||\xi||. \tag{3}$$

On the other hand, since $\tau, \xi \perp \mathbf{1}$, so does $\tau^{(1)}, \xi^{(1)}$, and

$$\left|\left\langle G^{\ell} \tau^{(1)}, \xi^{(1)} \right\rangle\right| \le \lambda_1^{\ell+1} ||\tau^{(1)}|| ||\xi^{(1)}||. \tag{4}$$

Combing Eq. (3) and (4), we obtain Eq. (2). □

Combing the equations above, we obtain $|y^{\mathrm{T}}Gx| \leq \lambda_2^{k-1} + 2(\varepsilon + \lambda_1) + \lambda_2^k$, which completes the proof of Theorem 3.

5 Iterative Construction

For the given even D of the form $D = D_2^k$, let $\varepsilon = D_2^{-k}$ and $\lambda_2 = \lambda_{\mathrm{Ram}}(D_2) + \varepsilon$. We use brute search to find a sequence of graphs $\overline{H} = (H_1, \cdots, H_k)$ that is ε-good, where $H_i, 1 \leq i \leq k$, is a (D^{16}, D_2, λ_2)-graph. Since D, D_2 and λ_2 are independent with the final graph size, $\overline{H}$ can be found within constant time. Alternatively, we can choose each H_i from G_{D^{16},D_2} uniformly and independently and verify whether or not $\lambda(H_i) \leq \lambda_2$. Theorem 2 guarantees that with high probability $\overline{H}$ is good.

Starting with a constant-size graph G_1 of the form (N_0, D, λ), G_2 of the form (N_0^2, D, λ), where $N_0 = D^{16}, \lambda = 2\lambda_2^{k-1}$, and $\overline{H} = (H_1, \cdots, H_k)$, the family of almost-Ramanujan graphs can be iteratively constructed by definition

$$G_{t+1} = \left(G_{\lfloor \frac{t-1}{2} \rfloor} \otimes G_{\lceil \frac{t-1}{2} \rceil}\right)^2 \textcircled{z} \overline{H}.$$

We have the following theorem.

Theorem 4. *Each graph G_{t+1} is an (N_0^{t+1}, D, λ)-graph.*

Proof. From the description of generalized zig-zag products, it is easy to see that each vertex in G_t is replaced by a "cloud" H_i with size N_0. Combining this fact with the property of tensor product, it is not hard to show that the number of vertices in G_{t+1} is N_0^{t+1}. Since each edge in G_t corresponds to a walk with length $2k-1$ indexed by $\overline{i} = (i_1, \cdots, i_k)$, thus the degree of G_{t+1} is D.

The spectral analysis of G_{t+1} is directly from [4]. □

Corollary 1. *Each graph G_{t+1} is fully explicitly constructible.*

Acknowledgements. This work came about during a delightful visit to The University of Hong Kong. He Sun thanks Professor Francis Y. L. Chin for creating the enjoyable research environment for the author. The authors of this paper thank Keping Huang for many hours of simulating discussion on closely related topics.

References

1. Ajtai, M.: Recursive Construction for 3-Regular Expanders. Combinatorica 14(4), 379–416 (1994)
2. Alon, N.: Spectral Techniques in Graph Algorithms. In: Proceedings of the 3rd International Conference on Latin American Theoretical Informatics, pp. 206–215 (1998)
3. Alon, N., Schwartz, O., Shapira, A.: An Elementary Construction of Constant-Degree Expanders. Combinatorics, Probability and Computing 17, 319–328 (2008)

4. Ben-Aroya, A., Ta-Shma, A.: A Combinatorial Construction of Almost-Ramanujan Graphs using the Zig-Zag Product. In: Proceedings of the 40th Symposium on Theory of Computation, pp. 325–334 (2008)
5. Friedman, J.: A Proof of Alon's Second Eigenvalue Conjecture. In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing, pp. 720–724 (2003)
6. Hoory, S., Linial, N., Wigderson, A.: Expander Graphs and their Applications. Bulletin of the American Mathematical Society 43(4), 439–561 (2006)
7. Lubotzky, A., Phillips, R., Sarnak, P.: Ramanujan Graphs. Combinatorica 8(3), 261–277 (1988)
8. Margulis, G.A.: Explicit Construction of Expanders. Problemy Peredaci Informacii 9(4), 71–80 (1973)
9. Morgenstern, M.: Existence and Explicit Construction of $q+1$ Regular Ramanujan Graphs for Every Prime Power q. Journal of Combinatorial Theory, Series B 62(1), 44–62 (1994)
10. Reingold, O., Vadhan, S., Wigderson, A.: Entropy Waves, the Zig-Zag Graph Product, and New Constant-Degree Expanders. Annuals of Mathematics 155(1), 157–187 (2002)
11. Reingold, O.: Undirected st-Connectivity in Log-Space. In: Proceedings of the 37th Symposium on Theory of Computation, pp. 376–385 (2005)
12. Schöning, U.: Construction of Expanders and Superconcentrators Using Kolmogorov Complexity. Random Structures and Algorithms 17(1), 64–77 (2000)

A $2\log_2(n)$-Approximation Algorithm for Directed Tour Cover

Viet Hung Nguyen

LIP6, Université Pierre et Marie Curie Paris 6, 4 place Jussieu, Paris, France
LIF, Université de la Mediterrannée, 163 avenue de Luminy, Marseille, France

Abstract. Given a directed graph G with non-negative cost on the arcs, a directed tour cover T of G is a cycle (not necessary simple) in G such that either head or tail (or both of them) of every arc in G is touched by T. The minimum directed tour cover problem (DToCP) which is to find a directed tour cover of minimum cost, is NP-hard. It is thus interesting to design approximation algorithms with performance guarantee to solve this problem. Although its undirected counterpart (ToCP) has been studied in recent years [1,6], in our knowledge, the DTCP remains widely open. In this paper, we give a $2\log_2(n)$-approximation algorithm for the DTCP.

1 Introduction

Let $G = (V, A)$ be a directed graph with a (non-negative) cost function $c : A \Rightarrow \mathbb{Q}_+$ defined on the arcs. A *directed tour*(respectively *tree*) cover T is a subgraph $T = (U, F)$ such that

1. for every $e \in A$, F contains an arc f intersecting e, i.e. f and e have at least one end-node in common.
2. T is a closed directed walk (respectively branching).

We consider in this paper *the minimum directed tour cover (DToCP)* problem which is to find a directed tour cover of minimum cost. As a directed tour cover can visit a node more than one time, we can assume without loss of generality that the vector cost c is a metric. A closely related problem of DToCP is its undirected counterpart, the minimum tour cover (ToCP) problem which has been studied in recent years. The ToCP is introduced in a paper by Arkin et al. [1]. The authors prove that ToCP is NP-hard and give a purely combinatorial 5.5-approximation algorithm. Later, Konemann et al. [6] propose an integer formulation for ToCP and using it to improve the approximation ratio to 3.

We can prove that DToCP is NP-hard by a reduction from the metric Asymmetric Traveling Salesman Problem (ATSP). The proof is very similar to the one for ToCP described in [1]. Given any instance $G = (V, A)$ of metric ATSP, for every node $v \in V$, we create a new node v' and an arc (v, v') of cost $+\infty$ to obtain a new graph G'. Let us consider the DToCP on G', we can see that any directed tour cover on G' must cover V. As the costs are metric, given any

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 208–218, 2009.

directed tour cover, we can find a traveling salesman tour on G of no greater cost. Hence, there is an one-one correspondance between the optimal solutions of DToCP on G' and the optimal traveling salesman tours on G. Moreover, this reduction is approximation-preserving, i.e. if we have an α-approximation algorithm for DToCP then we can approximate the metric ATSP with the same ratio α. In our knowledge, the best approximation ratio for ATSP is so far $0.824\log_2(n)$ achieved by Kaplan et al. [5]. Therefore, it is rather difficult to design an algorithm of a constant approximation ratio for DToCP. In fact in this paper, we will present a $2\log_2(n)$-approximation algorithm for the DTCP. This ratio is more than 2 times the ratio for ATSP that can be explained by the fact that our algorithm is inspired from the algorithm given by Konemann et al. [6] for the undirected case which is based on the bound of the Held-Karp relaxation for metric TSP. It is proved that the metric TSP costs no more than $\frac{3}{2}$ the Held-Karp bound for the undirected case and the metric ATSP costs no more than $\log_2(n)$ the Held-Karp bound for the directed case [2,8]. Hence, our ratio $2\log_2(n)$ for DToCP follows logically the best ratio 3 for ToCP.

The paper is organized as follows: In Section 2, we present an integer formulation for DToCP. We discuss in Section 3 about the parsimonious property of Eulerian directed graphs, in particular in 3.1. we prove the equalivalent directed version of Goemans and Bertsimas's theorem on splitting operations. In 3.2., we derive an equivalent linear program to the Hald-Karp relaxation for metric ATSP. This linear program will be useful in the analysis of our algorithm's performance guaranntee in Section 5. Before that, we state our approximation algorithm for DToCP in Section 4.

Let us introduce the notation that will be used in the paper. Let $G = (V, A)$ be a digraph with vertex set V and arc set A. If $x \in \mathbb{Q}^{|A|}$ is a vector indexed by the arc set A and $F \subseteq E$ is a subset of arcs, we use $x(F)$ to denote the sum of values of x on the arcs in F, $x(F) = \sum_{e\in F} x_e$. Similarly, for a vector $y \in \mathbb{Q}^{|V|}$ indexed by the vertices and $S \subseteq V$ is a subset of vertices, $y(S)$ denotes the sum of values of y on the vertices in the set S. For a subset of vertices $S \subseteq V$, let $A(S)$ denote the set of the arcs having both end-nodes in S. Let $\delta^+(S)$(respectively $\delta^-(S)$) denote the set of the arcs having only the tail (respectively head) in S. We will call $\delta^+(S)$ the *outgoing cut* associated to S, $\delta^-(S)$ the *ingoing cut* associated to S. For two subset $U, W \subset V$ such that $U \cap W = \emptyset$, let $(U : W)$ be the set of the arcs having the tail in U and the head in W. For $u \in V$, we say v an *outneighbor* (respectively *inneighbor*) of u if $(u, v) \in A$ (respectively $(v, u) \in A$). For the sake of simplicity, in clear contexts, the singleton $\{u\}$ will be denoted simply by u. When we work on more than one graph, we precise the graph in the index of the notation, e.g. $\delta^+_G(S)$ will denote $\delta^+(S)$ in the graph G.

2 Integer Formulation

Let

$$\mathcal{F} = \{S \subseteq V \mid A(S) \neq \emptyset, A(V \setminus S) \neq \emptyset\}.$$

Given any tour cover TC of G, by definition, for any $S \in \mathcal{F}$ we have $TC \cap \delta^+(S) \geq 1$ and $TC \cap \delta^-(S) \geq 1$. Note that the condition $TC \cap \delta^-(S) \geq 1$ is equivalent to $TC \cap \delta^+(V \setminus S) \geq 1$, and by definition of $\mathcal{F}$, a vertex subset S belongs to $\mathcal{F}$ if and only if $V \setminus S$ belongs to $\mathcal{F}$. This observation motivates our integer formulation for DToCP. For any $e \in A$, let x_e indicate the number of copies of e included in the tour cover. We minimize the total weight of arcs included, under the condition that every outgoing cut associated to some $S \in \mathcal{F}$ be crossed at least one. In order to ensure that our solution is a tour we also need to specify that for any node $v \in V$, the number of arcs entering v is equal to the number of arcs leaving v in the tour. This integer formulation can be stated as follows:

$$\min \sum_{e \in A} c_e x_e$$

$$\sum_{e \in \delta^+(v)} x_e = \sum_{e \in \delta^-(v)} x_e \qquad \text{for all } v \in V;$$

$$\sum_{e \in \delta^+(S)} x_e \geq 1 \qquad \text{for all } S \in \mathcal{F};$$

$$x_e \text{ integer} \qquad \text{for all } e \in A.$$

Replacing the integrality constraints by

$$x_e \geq 0 \text{ for all } e \in A$$

we obtain the linear programming relaxation. We use DToC(G) to denote the convex hull of all vectors x satisfying the constraints of the constraints above (those of the linear programming relaxation).

Clearly minimizing the linear cost function $\sum_{e \in A} c_e x_e$ over DToC(G) can be done in polynomial time since the separation problem of the cut constraint can be solved in polynomial time. Indeed, given a candidate solution x and for every $e \in A$, let us consider x_e as the capacity on arc e. For each pair of arcs $e_1,\ e_2 \in E$, we compute the a minimum capacity cuts $\delta^+(U)$ separating them.

3 Parsimonious Property for Directed Eulerian Graphs

3.1 Connectivity and the Splitting Operation

Consider the following linear programs:

$$\min \sum_{e \in A} c_e x_e \tag{1}$$

subject to

$$x(\delta^+(u)) = k \qquad \text{for all } u \in V;$$

$$x(\delta^-(u)) = k \qquad \text{for all } u \in V;$$

$$x(\delta^+(S)) \geq k \qquad \text{for all } S \subset V \text{ such that } |S| \geq 2;$$

$$x_e \geq 0 \qquad \text{for all } e \in A,$$

and

$$\min \sum_{e \in A} c_e x_e \tag{2}$$

subject to

$$\begin{aligned} x(\delta^+(u)) &\geq k && \text{for all } u \in V \\ x(\delta^+(u)) &= x(\delta^-(u)) && \text{for all } u \in V \\ x(\delta^+(S)) &\geq k && \text{for all } S \subset V \text{ such that } |S| \geq 2 \\ x_e &\geq 0 && \text{for all } e \in A \end{aligned}$$

The following theorem states the parsimonious property for Eulerian directed graphs which expresses the relation between (1) and (2).

Theorem 1. *If the costs c satisfy the triangle inequality then the optimum of (2) is equal to the optimum of (1).*

The parsimonious property can be also formulated for undirected Eulerian graphs. We simply replace the outgoing and ingoing cuts in (2) and (1) by the corresponding cut to obtain the two linear programs for the undirected case. The undirected version of Theorem 1 (in a little more general form) has been proved by Goemans and Bertsimas [3]. Their proof consists mainly in showing the following lemma.

Lemma 1. [3] *Let $G = (V, E)$ be an Eulerian multigraph. Let $c_G(i, j)$ $(i, j \in V)$ denote the maximum number of edge-disjoint paths between i and j. Let s be any vertex of G and let u be any neighbor of s. Then there exists another neighbor of s, say v, such that, by splitting su and sv i.e., removing the edges su and sv and adding the edge uv, we obtain a multigraph G' satisfying the following conditions:*

1. *$c_{G'}(i, j) = c_G(i, j)$ for all $i, j \in V \setminus \{s\}$ and*
2. *$c_{G'}(s, j) = \min\{c_G(s, j), d_G(s)-2\}$ for all $j \in V \setminus \{s\}$, where $d_G(s)$ represents the degree of the vertex s in G.*

Condition 1 of Lemma 1 is a result due to Lovasz [7] on the connectivity properties of Eulerian multigraphs. Condition 2, added by Goemans et Bertsimas, states that splitting operation can be performed while maintaining most connectivity requirements involving vertex s.

For the connectivity of Eulerian multi-digraphs, Jackson [4] shows a similar result as Lovasz's one (i.e. Condition 1) for Eulerian multigraphs. This result can be stated in the following lemma.

Lemma 2. [4] *Let s be any node of G and let u be any outneighbor of s. Then there exists another inneighbor of s, say v, such that, by splitting (s, u) and (v, s) i.e., removing the arcs (s, u) and (v, s) and adding the arc (v, u), we obtain a multigraph G' satisfying*

1. *$p_{G'}(i, j) = p_G(i, j)$ for all $i, j \in V \setminus \{s\}$.*

Like the work of Goemans and Bertsimas, we formulate Condition 2 to Lemma 2 about the connectivities involving s.

Lemma 3. *Let s be any node of G and let u be any outneighbor of s. Then there exists another inneighbor of s, say v, such that, by splitting (s,u) and (v,s) i.e., removing the arcs (s,u) and (v,s) and adding the arc (v,u), we obtain a multigraph G' satisfying the following conditions:*

1. $p_{G'}(i,j) = p_G(i,j)$ *for all* $i,j \in V \setminus \{s\}$ *and*
2. $p_{G'}(s,j) = \min\{p_G(s,j), d^+_G(s) - 1\}$.

Proof. Our proof for Lemma 3 is very similar to the Goemans and Bertsimas' proof for Lemma 1. This proof also is in major part inspired from the one of Lovasz for Lemma 1 (Condition 1) which proceeds along the following lines.

1. There exists at most one set S satisfying:
   (a) $s \in S$, $u \notin S$,
   (b) $|\delta_G(S)| = c_G(i,j)$ for some $i,j \in V$, $i \in S$, $j \notin S$ and $i \neq s$ and
   (c) S is minimal with respect to the above two conditions.
2. If there is no such S, then any neighbor v of s can be used for the splitting operation.
3. If such a S exists then there exists at least one neighbor of s in S. Moreover, any neighbor of s in S can be used for the splitting operation.

We will show that this proof procedure can be used for the directed case. Let us remove the orientation on the arcs of G, then G becomes a undirected Eulerian multigraph. Given any $s \in V$, applying the Lovasz's procedure, we have

– either there exists a S as described above. Reconsider S in the original directed G, we prove that S satisfies
  (a) $s \in S$, $u \notin S$,
  (b) $|\delta^+_G(S)| = p_G(i,j)$ for some $i,j \in V$, $i \in S$, $j \notin S$ and $i \neq s$ and
  (c) S is minimal with respect to the above two conditions.

  Indeed, since G is Eulerian then $|\delta^+_G(S)| = |\delta^-_G(S)|$ and consequently $|\delta_G(S)| = 2|\delta^+_G(S)| = 2|\delta^-_G(S)|$ where $\delta_G(S)$ is the cut associated to S in the undirected graph G without the orientation on the arcs. Thus if $\delta_G(S) = c_G(i,j) = \min\{|\delta(T)| : i \in T, j \in V \setminus T\}$ then $\delta^+_G(S) = p_G(i,j) = \min\{|\delta^+(T)| : i \in T, j \in V \setminus T\}$. The conditions **(a)** and **(c)** are independent from the fact that G is directed or not.
– otherwise, i.e. such a S does not exist. Then for all $S \subset V$ such that $s \in S$ and $u \notin S$, we have $|\delta_G(S)| > c_G(i,j)$ for all $i,j \in V$, $i \in S$, $j \notin S$ and $i \neq s$. By $|\delta_G(S)| = 2|\delta^+_G(S)| = 2|\delta^-_G(S)|$, we have also in the original directed graph G, $|\delta^+_G(S)| > c_G(i,j)$ for all $i,j \in V$, $i \in S$, $j \notin S$ and $i \neq s$. By $|\delta_G(S)| = 2|\delta^+_G(S)| = 2|\delta^-_G(S)|$, we have $|\delta^+_G(S)| > p_G(i,j)$ for all $i,j \in V$, $i \in S$, $j \notin S$ and $i \neq s$ Hence, we can select any inneighbor v of s we can do the splitting operation on (s,u) and (v,s) without affecting the value of $p_G(i,j)$ for all $i,j \in V \setminus \{s\}$.

Hence we can summarize below the directed version of Lovasz's proof procedure.

1. There exists at most one set S satisfying:
   (a) $s \in S$, $u \notin S$,
   (b) $|\delta^+_G(S)| = p_G(i,j)$ for some $i, j \in V$, $i \in S$, $j \notin S$ and $i \neq s$ and
   (c) S is minimal with respect to the above two conditions.
2. If there is no such S, then any inneighbor v of s can be used for the splitting operation.
3. If such a S exists then there exists at least one inneighbor of s in S. Moreover, any inneighbor of s in S can be used for the splitting operation.

To show Lemma 3, we add a new node $\hat{s}$ to G and $d^+_G(s) - 1$ arcs from $\hat{s}$ to s and $d^-_G(s) - 1$ arcs from s to $\hat{s}$. Let $\hat{G}$ be this new graph. Clearly, $\hat{G}$ is Eulerian. Moreover,

$$p_{\hat{G}}(i,j) = p_G(i,j) \text{ for all } i, j \in V \setminus \{s\},$$

$$p_{\hat{G}}(\hat{s},j) = \min\{p_G(s,j), d^+_G(s) - 1\} \text{ for all } j \in V \setminus \{s\}.$$

Along the below proof, when we use $d^+(s)$ or $d^-(s)$, it can be understood respectively as $d^+_G(s)$ or $d^+_{\hat{G}}(s)$ and $d^-_G(s)$ or $d^-_{\hat{G}}(s)$. Since $d^+_G(s) = d^+_{\hat{G}}(s)$ and $d^-_G(s) = d^-_{\hat{G}}(s)$, we do not distinguish them.

Now applying the proof procedure to $\hat{G}$, we can see that there are two possible cases:

1. There is no $S \subset V \cup \{\hat{s}\}$ that satisfies the condition described in the proof procedure. Then we can select any inneighbor $v \neq \hat{s}$ of s and do the splitting operation on the arcs (v,s) and (s,u). Let G' be the resulted after this splitting operation. Since the splitting operation does not affect the connectivity of all the nodes in $V \cup \{\hat{s}\} \setminus \{s\}$, we have $p_{G'}(\hat{s},j) = p_{\hat{G}}(\hat{s},j)$ for all $j \in V \setminus \{s\}$. But as $d^+_{G'}(s) = d^+_G(s) - 1$,

   $$p_{\hat{G}}(\hat{s},j) = \min\{p_G(s,j), d^+_G(s) - 1\} \text{ for all } j \in V \setminus \{s\} \text{ and}$$

   $$p_{G'}(\hat{s},j) = \min\{p_{G'}(s,j), d^+_{G'}(s)\} \text{ for all } j \in V \setminus \{s\}.$$

   As $p_{\hat{G}}(\hat{s},j) = p_{G'}(\hat{s},j)$ then $p_{G'}(s,j)$ can not strictly smaller than $\min\{p_G(s,j), d^+_G(s)-1\}$ and as $p_{G'}(s,j) \leq p_G(s,j)$, by construction, $p_{G'}(s,j)$ can not strictly greater than $\min\{p_G(s,j), d^+_G(s) - 1\}$. Hence

   $$p_{G'}(s,j) = \min\{p_G(s,j), d^+_G(s) - 1\}.$$

2. Otherwise, i.e. there exists exactly one set $S \subset V \cup \{\hat{s}\}$ satisfying
   (a) $s \in S$, $u \notin S$,
   (b) $|\delta^+_{\hat{G}}(S)| = p_{\hat{G}}(i,j)$ for some $i, j \in V \cap \{\hat{s}\}$, $i \in S$, $j \notin S$ and $i \neq s$ and
   (c) S is minimal with respect to the above two conditions.

Proposition 1. *If such a S exists then $\hat{s} \in S$.*

Proof. Indeed if $\hat{s} \notin S$ then $|\delta^+_{\hat{G}}(S)| \geq d^+(s)$ since both u and $\hat{s} \notin S$ and $\hat{G}$ is Eulerian. Moreover $|\delta^+_{\hat{G}}(S)|$ would be equal to $p_{\hat{G}}(i, \hat{s})$ for some $i \in S \setminus \{s\}$. This follows by **(b)** and the fact that $|\delta^+(S \cup \{\hat{s}\})_{\hat{G}}| < |\delta^+_{\hat{G}}(S)|$ (since $s \in S$) which implies that there is no i and j in $V \setminus \{s\}$ with $i \in S$ and $j \notin S$ such that $|\delta^+_{\hat{G}}(S)| = p_{\hat{G}}(i, j)$. This leads to a contradiction since $d^+(s) \leq |\delta^+_{\hat{G}}(S)| = p_{\hat{G}}(i, \hat{s}) \leq d^-_{\hat{G}}(\hat{s}) < d^-(s) = d^+(s)$. Proposition 1 is proved. □

Proposition 2. *If a S satisfying **(a)-(c)** exists there must exist some $v \in S$ with $v \neq \hat{s}$ such that v is an inneighbor of s.*

Proof. Indeed, if S does not contain any inneighbor of s other than $\hat{s}$ then

$$|\delta^+_{\hat{G}}(S)| = |\delta^-_{\hat{G}}(S)| > d^-(s) > d^+(\hat{s}) \geq p_{\hat{G}}(\hat{s}, j).$$

Hence, there would exist some $i \in S$, $s \neq i \neq \hat{s}$ and $j \notin S$ such that $\delta^+_{\hat{G}}(S) = p_{\hat{G}}(i, j)$. This leads to a contradiction to the fact that S is minimal since $S \setminus \{s, \hat{s}\}$ would separate i from j and as

$$|\delta^+(S \setminus \{s, \hat{s}\})_{\hat{G}}| = |\delta^+_{\hat{G}}(S)| + |(S : \{s\})| - |(\{s\} : V \setminus S)|$$

and $|(S : \{s\})| = 0$ and $|(\{s\} : V \setminus S)| > 0$, we have $|\delta^+(S \setminus \{s, \hat{s}\}_{\hat{G}})| < |\delta^+_{\hat{G}}(S)|$. Proposition 2 is then proved. □

Therefore, there exists some $v \neq \hat{s}$ such that by splitting of (s, u) and (v, s) we obtain a graph $\hat{G}'$ with $p_{\hat{G}}(i, j) = p_{\hat{G}'}(i, j)$ for all $i, j \in V \cup \{\hat{s}\} \setminus \{s\}$. After removing $\hat{s}$, we obtain a graph G' which can be also obtained from G by splitting off (s, u) and (s, v). G' satisfies

$$p_{G'}(i, j) = p_{\hat{G}'}(i, j) = p_{\hat{G}}(i, j) = p_G(i, j) \text{ for all } i, j \in V \setminus \{s\}$$

and

$$\begin{aligned} p_{G'}(s, j) = p_{\hat{G}'}(s, j) \geq p_{\hat{G}'}(\hat{s}, j) = p_{\hat{G}}(\hat{s}, j) \\ = \min\{p_G(s, j), d^+_G(s) - 1\} \text{ for all } j \in V \setminus \{s\}. \end{aligned}$$

Moreover, due to splitting operation

$$p_{G'}(s, j) \leq p_G(s, j) \text{ and } p_{G'}(s, j) \leq d^+_G(s) - 1.$$

We finish here the proof of Lemma 3. □

Proof of Theorem 1: Let C be the optimal value of (2) and C' be the optimal value of (1). As (2) is a relaxation of (1), we have $C \leq C'$. To show $C \geq C'$ let x be an optimal solution of (2), i.e. $C = \sum_{e \in A} c_e x_e$. We can assume without loss of generality that x is integer since there exists some integer p such that px is integer and we can replace k by $p.k$ in (2) and (1). Let $G = (V, A)$ be the multigraph which has x_e copies of each arc e such that $x_e > 0$. Applying Lemma 3 repeatedly with s chosen among the nodes of V such that $x(\delta^+(v)) > k$, we will obtain a multigraph G' such that

- $p_{G'}(i, j) \geq k$ for all $i, j \in V$
- $d^+_{G'}(i) = k$ for all $i \in V$.

Let x' the point such that for each $e \in A$, x'_e is equal to the number of copies of e in G'. We can see that x' is a solution of (1). Since the costs satisfy the triangle inequality, each time we perform a splitting operation the cost of the solution does not increase which implies that

$$C = \sum_{e \in A} c_e x_e \geq \sum_{e \in A} c_e x'_e \geq C'.$$

Theorem 1 is then proved. □

3.2 Held-Karp Relaxation for ATSP and the Parsimonious Property

Let us consider the Asymmetric Travelling Salesman Problem (ATSP) on G. The Held-Karp bound for this problem can be computed by solving the following linear program:

$$\min \sum_{e \in A} c_e x_e$$

subject to

$$\begin{aligned}
x(\delta^+(S)) &\geq 1 && \text{for all } S \subset V \text{ such that } |S| \geq 2;\\
x(\delta^+(v)) &= 1 && \text{for all } v \in V;\\
x(\delta^-(v)) &= 1 && \text{for all } v \in V;\\
0 \leq x_e &\leq 1 && \text{for all } e \in A.
\end{aligned}$$

This linear program is called *Held-Karp relaxation* for ASTSP. In fact the upper bound constraint $x_e \leq 1$ for all $e \in A$ is redundant due to the degree constraints $x(\delta^+(v)) = 1$ and $x(\delta^-(v)) = 1$ for all $v \in V$. Thus, we can rewrite the Held-Karp relaxation as follows:

$$\min \sum_{e \in A} c_e x_e \tag{3}$$

subject to

$$\begin{aligned}
x(\delta^+(S)) &\geq 1 && \text{for all } S \subset V \text{ such that } |S| \geq 2;\\
x(\delta^+(v)) &= 1 && \text{for all } v \in V\ ;\\
x(\delta^-(v)) &= 1 && \text{for all } v \in V;\\
x_e &\geq 0 && \text{for all } e \in A.
\end{aligned}$$

The following theorem is a direct application of Theorem 1.

Theorem 2. *If the costs c satisfy the triangle inequality then the optimum of (3) is equal to the optimum of:*

$$\min \sum_{e \in A} c_e x_e \tag{4}$$

$$
\begin{aligned}
&\textit{subject to}\\
&x(\delta^+(S)) \geq 1 && \textit{for all } \emptyset \neq S \subset V \textit{ (included } S \textit{ singleton)};\\
&x(\delta^+(v)) = x(\delta^-(v)) && \textit{for all } v \in V;\\
&x_e \geq 0 && \textit{for all } e \in A.
\end{aligned}
$$

4 Algorithm

We are now ready to state our algorithm for directed tour cover.

---

(1) Let x^* be the vector minimizing cx over DToC(G).
(2) Let $U \leftarrow \{v \in V | x^*(\delta^+(\{v\})) \geq \frac{1}{2}\}$.
(3) Let G_U be the subgraph of G induced by U with the same cost c_e for each arc e in G_U.
(4) Run the Frieze, Galbiati and Maffioli heuristic [2] to find an approximate minimum traveling salesman directed tour on G_U.

Note that the linear program in step **(1)** can be solved in polynomial time by using the ellipsoid method with a min-cut computation as separation oracle.

The algorithm outputs a directed tour which spans U. We can see that U is vertex cover of G. Since for any arc $e = (u, v) \in A$, $x^*(\delta^+(\{u, v\})) \geq 1$ and $x^*(\delta^+(\{u, v\})) = x^*(\delta^+(u)) + x^*(\delta^+(v)) - 2x^*_e$, at least $x^*(\delta^+(u))$ or $x^*(\delta^+(v))$ is greater or equal to $\frac{1}{2}$, i.e. at least u or v should belong to U. Therefore, the algorithm outputs a directed tour cover of G.

5 Performance Guarantee

Let x^* be the vector minimizing $\sum_{e \in A} c_e x_e$ over DToC(G) and

$$U = \big\{v \in V \mid x^*(\delta^+(\{v\})) \geq \frac{1}{2}\big\}.$$

Let $y^* = 2x^*$, and k be the minimum integer such that ky^* is integer. Let $\tilde{G} = (V, \tilde{A})$ be the multi-graph that has ky^*_e copies of arc e for each $e \in \tilde{A}$. Note that $\tilde{G}$ is Eulerian. For any $i \neq j \in V$, let $p_{\tilde{G}}(i, j)$ be number of disjoint paths from i to j in $\tilde{G}$.

Remark 1. $p_{\tilde{G}}(u, j) \geq k$ for all $u \in U$ and $j \in V \setminus \{u\}$.

Proof. To prove the remark, we will show that for any $S \subset V$ such that $u \in S$ and $j \in V \setminus S$, we have $|\delta^+_{\tilde{G}}(S)| \geq k$. Indeed, if $A(S) \neq \emptyset$ then by construction, $x^*(\delta^+_G(S)) \geq 1/2$ which implies $y^*(\delta^+_G(S)) \geq 1$ and consequently $|\delta^+_{\tilde{G}}(S)| \geq k$. Otherwise, if $A(S) = \emptyset$ then $x^*(\delta^+_G(S)) \geq x^*(\delta^+(u)) \geq \frac{1}{2}$. Equivalently, $y^*(\delta^+_G(S)) \geq y^*(\delta^+(u)) \geq 1$. Hence, we also have $|\delta^+_{\tilde{G}}(S)| \geq k$. □

In $\tilde{G}$, for each $i \in V \setminus U$, doing repeatedly the splitting operation on i until $d^+_{\tilde{G}}(i) = d^-_{\tilde{G}}(i) = 0$. Consequently, after these operations, all the arcs in $\tilde{G}$ having the end-nodes in U. Let $\tilde{G}_U = (U, \tilde{A}_U)$ be the subgraph of induced by U of $\tilde{G}$.

Remark 2. After these splitting operations, $p_{\tilde{G}_U}(u, v) \geq k$ for all $u \in U$ and $v \in U \setminus \{u\}$.

Let $G_U = (U, A_U)$ be the simple graph obtained from $\tilde{G}_U$ by taking only one copie for each $e \in \tilde{A}_U$. Let x^U be the vector in $\mathbb{R}^{|A_U|}$ such that for each $e \in A_U$, $x^U_e = \frac{p_e}{k}$ where p_e is the number of copies of e in $\tilde{A}_U$. Let $P_G(i, j)$ is the shortest path from i to j in G for all $i \neq j \in V$. We define $c^U \in \mathbb{R}_+^{|A_U|}$ as follows

$$c^U_{ij} = \begin{cases} c_{ij} & \text{if } (i, j) \in A; \\ c(P_G(i, j)) & \text{otherwise.} \end{cases}$$

for all arc $(i, j) \in \tilde{A}_U$.

Lemma 4. *We have* $2cx^* = cy^* \geq c^U x^U$.

Proof. We have seen that kx^U is obtained from ky^* by repeatedly splitting operations on the vertices in $V \setminus U$. Since c satisfies the triangle inequality, we have $c^U(kx^U) \leq c(ky^*)$ which implies $c^U x^U \leq cy^* = 2cx^*$. □

Let z^* be an optimal solution of Held-Karp relaxation over G_U.

Lemma 5. $c^U x^U \geq c^U z^*$.

Proof. We can see that x^U satisfies the following linear inequalities

$$x(\delta^+_{G_U}(u)) = x(\delta^-_{G_U}(u))$$

and

$$x(\delta^+_{G_U}(S)) \geq 1 \text{ for all } \emptyset \subset S \subset U,$$

which are the inequalities of (4) applied to G_U. By Theorem 2, z^* minimizes the cost function $c^U x$ over (4). Hence we have $c^U x^U \geq c^U z^*$. □

We are ready now to state the main theorem

Theorem 3. *The algorithm outputs a directed tour cover of cost no more than* $2\log_2(n)$ *times the cost of the minimum directed tour cover.*

Proof. The algorithm return as a solution of DToCP, T the travelling salesman directed tour output by Frieze et al's heuristic applied on G_U. Let x^T be its incidence vector. The cost of T is then equal to $c^U x^T$. Williamson [8] proved that the cost of T is not more than $\log_2(n)$ times the cost of the optimal solution of Held-Karp relaxation over G_U, i.e. $c^U x^T \leq \log_2(n) c^U z^*$. By Lemmas 5 and 4, we have $c^U x^T \leq 2\log_2(n) cx^*$. □

References

1. Arkin, E.M., Halldórsson, M.M., Hassin, R.: Approximating the Tree and Tour Covers of a Graph. Information Processing Letters 47, 275–282 (1993)
2. Frieze, A.M., Galbiati, G., Maffioli, F.: On the Worst Case Performance of some Algorithms for the Asymmetric Traveling Salesman problem. Networks 12, 23–39 (1982)
3. Goemans, M.X., Bertsimas, D.J.: On the Parsimonious Property of Connectivity Problems. In: Proc. of the 1st ACM-SIAM Symposimum on Disc. Alg (SODA 1990), pp. 388–396 (1990)
4. Jackson, B.: Some Remarks on Arc-Connectivity, Vertex Splitting, and Orientation in Graphs and Digraphs. Journal of Graph Theory 12(3), 429–436 (1988)
5. Kaplan, H., Lewenstein, M., Shafrir, N., Sviridenko, M.: Approximation Algorithms for Asymmetric TSP by Decomposing Directed Regular multigraphs. J. ACM 52(4), 602–626 (2005)
6. Könemann, J., Konjevod, G., Parekh, O., Sinha, A.: Improved Approximations for Tour and Tree Covers. Algorithmica 38(3), 441–449 (2003)
7. Lovasz, L.: On Some Connectivity Properties of Eulerian Graphs. Acta Mathematica Hungarica 28, 129–138 (1976)
8. Williamson, D.P.: Analysis of the Held-Karp Heuristic for the Traveling Salesman Problem. Master Thesis, MIT (1990)

Approximation Algorithms for Max 3-Section Using Complex Semidefinite Programming Relaxation*

Ai-fan Ling

School of Finance, Jiangxi University of Finance & Economics
Nanchang 330013, China
aifanling@yahoo.com.cn

Abstract. We present an approximation algorithm for the Max 3-section problem which divides a weighted graph into 3 parts of equal size so as to maximize the weight of the edges connecting different parts. The algorithm is based on a complex semidefinite programming and can in some sense be viewed as a generalization of the approximation algorithm proposed by Ye [17] for the Max Bisection problem. Our algorithm can hit the 2/3 bound and has approximate ratio 0.6733 for Max 3-section that sightly improves the 2/3 bound obtained by Andersson [1] and Gaur [8], respectively.

1 Introduction

Given a graph $G(V; E)$, with node set $V$and edge set E, the Max 3-cut problem is to find a partition $S_0 \subset V$, $S_1 \subset V$ and $S_2 \subset V$, of the set V, such that $S_0 \bigcup S_1 \bigcup S_2 = V$, $S_i \bigcap S_j = \emptyset$ $(i \neq j)$ and the sum of the weights on the edges connecting the different parts is maximized.

Similar to the Max cut problem, the max 3-cut problem has long been known to be NP-complete [14], even for any un-weighted graphs [7], and has also applications in circuit layout design, statistical physics and so on [2]. However, due to the complexity of this problem, its research progresses is much lower than that of Max cut. Goemans and Williamson [9] are the pioneers of using semidefinite programming relaxation to solve the Max cut problem and obtained the splendid 0.878-approximate algorithm. Frieze and Jerrum [6] extended Goemans and Williamson's 0.878-approximate algorithms [9] to general Max k-Cut. Particular, Frieze and Jerrum [6] obtained a 0.651-approximation algorithm for Max bisection and a 0.800217-approximation algorithm for Max 3-Cut. Recently, Goemans and Williamson [10] improved Frieze and Jerrum's 0.800217-approximation ratio to 0.836 using random hyperplane based on a complex semidefinite programming relaxation. Zhang and Huang [18] also obtained a 0.836-approximation algorithm

* This work is supported by National Natural Science Foundations of China, No. 10671152.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 219–230, 2009.

for Max 3-Cut using complex semidefinite programming relaxation and complex-valued multivariate normal distribution.

If we further require that $|S_0| = |S_1| = |S_2| = n/3$ when the number of nodes n is a multiple of 3, then Max 3-Cut becomes to the Max 3-section problem that will be mainly considered in this paper. Comparing with Max bisection, thought there exist several approximation algorithms for Max bisection, e.g. see [3,6,16,17,5], as addressed by Frieze and Jerrum [6], their 0.651-approximation algorithm is not directly generalized to Max 3-section such that their bisection heuristic beat the 2/3 lower bound of a simple random selection heuristic. Andersson [1] proposed a novel random algorithm that was also based on semidefinite programming and obtained the approximation ratio $(2/3) + O(\frac{1}{n^3})$. Andersson's approach may be viewed as a generation of Frieze and Jerrum's random rounding method. Recently, Gaur et al. [8] proposed a determinate approximation algorithm for the capacitated Max k-Cut problem. Their determinate approximation algorithm can beat the 2/3 lower bound for the Max 3-section problem.

In this paper, we also present an approximation algorithm for the Max 3-section problem. The algorithm is based on a complex semidefinite programming that is originally proposed by Goemans and Williamson [10] for Max 3-cut. Our rounding method can in some sense be viewed as a generalization of the approximation algorithm proposed by Ye [17] for the Max Bisection problem. Our algorithm can hit the 2/3 lower bound. In particular, the proposed algorithm can obtain the approximate ratio 0.6733 for Max 3-section that sightly improves the bound obtained by Andersson [1] and Gaur [8], respectively.

This paper is organized as follows. In Section 2, we state the model of Max 3-section, give its complex semidefinite programming relaxation and random rounding method. In Section 3, we present a fussy size adjusting procedure to obtain three parts of V with equal sizes. We further verify that, this size adjusting procedure can generate a good bound in this section. In Section 4, we analyze the performance ratio of the proposed rounding method.

Throughout out of this paper, we use the following standard notations from Goemans and Williamson [10] and Zhang and Huang [18]. For a complex number $y = a + ib \in \mathbb{C}$, we denote its real part a by $\mathrm{Re}(y)$, its image part b by $\mathrm{Im(y)}$ and its complex angle principal value by $\mathrm{Arg}(y) \in [0, 2\pi]$. The complex conjugate of $y = a + ib$ is denoted by $\overline{y} = a - ib$. For an n dimensional complex vector $\mathbf{y} \in \mathbb{C}^n$ written as bold letter and n dimensional complex matrix $Y \in \mathbb{C}^{n\times n}$, we write $\mathbf{y}^*$ and Y^* to denote their conjugate and transpose. That is $\mathbf{y}^* = \overline{\mathbf{y}}^T$ and $Y^* = \overline{Y}^T$. We write $\mathbf{e}_i$ to denote the vector with zeros everywhere except for an 1 in the i-th cmponent. Let $\mathcal{S}_n$ denote the set of symmetric $n \times n$ real matrices and $\mathcal{H}_n$ the set of complex $n \times n$ Hermitian matrices. For any two complex vector $\mathbf{u}, \mathbf{v} \in \mathbb{C}^n$, we use $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u} \cdot \mathbf{v} = \mathbf{u}^* \mathbf{v}$ as their inner product. For any two complex matrices $A, B \in \mathcal{H}_n$, we let $\langle A, B \rangle = A \cdot B$ be the inner product of the two matrices; i.e. $\langle A, B \rangle = A \cdot B = \mathrm{Tr}(B^* A) = \sum_{i,j} \overline{b}_{ij} a_{ij}$, where $A = (a_{ij})$ and $B = (b_{ij})$. We denote a positive semidefinite Hermitian matrix A by $A \succeq 0$.

2 Complex SDP Relaxation and Random Algorithm

We denote the third roots of unity by $1, \omega = e^{i\frac{2\pi}{3}}, \omega^2 = e^{i\frac{4\pi}{3}}$. Introduce a complex variable $y_i \in \{1, \omega, \omega^2\}, i = 1, \cdots, n$ and denote $S_k = \{i : y_i = \omega^k\}$, $k = 0, 1, 2$. Then the Max 3-section problem can be expressed as

$$\begin{aligned} \text{M3S}: \qquad \max \quad & \sum_{i<j} w_{ij}(\tfrac{2}{3} - \tfrac{1}{3} y_i \cdot y_j - \tfrac{1}{3} y_j \cdot y_i) \\ & = \tfrac{2}{3} \sum_{i<j} w_{ij}(1 - \mathrm{Re}(y_i \cdot y_j)) \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^n y_i = 0, \\ & y_i \in \{1, \omega, \omega^2\}, i = 1, 2, \cdots, n. \end{aligned} \tag{2.1}$$

We relax the complex variable y_i into an n dimensional complex vector $\mathbf{y}_i$, then we get a CSDP relaxation of M3S as follows.

$$\begin{aligned} \text{CSDP}: \qquad \max \quad & \sum_{i<j} w_{ij}(\tfrac{2}{3} - \tfrac{1}{3} \mathbf{y}_i \cdot \mathbf{y}_j - \tfrac{1}{3} \mathbf{y}_j \cdot \mathbf{y}_i) \\ & = \tfrac{2}{3} \sum_{i<j} w_{ij}(1 - \mathrm{Re}(\mathbf{y}_i \cdot \mathbf{y}_j)) \\ \text{s.t.} \quad & \textstyle\sum_{i,j} \mathbf{y}_i \cdot \mathbf{y}_j = 0, \\ & \|\mathbf{y}_i\| = 1, i = 1, 2, \cdots, n, \\ & A_{ij}^k \cdot Y \geq -1, i, j = 1, 2, \cdots, n, \quad k = 0, 1, 2 \\ & Y \succeq 0, \end{aligned} \tag{2.2}$$

where $Y_{ij} = \mathbf{y}_i \cdot \mathbf{y}_j$ and $A_{ij}^k = \omega^k \mathbf{e}_i \mathbf{e}_j^T + \omega^{-k} \mathbf{e}_j \mathbf{e}_i^T$. It is easily to verify that constraints $A_{ij}^k \cdot Y \geq -1$ can be expressed as

$$\mathrm{Re}(\omega^k Y_{ij}) \geq -\frac{1}{2}, \quad k = 0, 1, 2.$$

Let Y be an optimal solution of CSDP relaxation and $\theta \in [0, 1]$ be given. Define

$$\widehat{Y} = \theta Y + (1 - \theta) I.$$

We randomly generate a complex vector ξ, such that $\xi \sim N(0, \widehat{Y})$, and assign

$$y_i = \begin{cases} 1, & \text{If } \ \mathrm{Arg}(\xi_i) \in [0, \frac{2\pi}{3}); \\ \omega, & \text{If } \ \mathrm{Arg}(\xi_i) \in [\frac{2\pi}{3}, \frac{4\pi}{3}); \\ \omega^2, & \text{If } \ \mathrm{Arg}(\xi_i) \in [\frac{4\pi}{3}, 2\pi). \end{cases} \tag{2.3}$$

It has been verified by Zhang and Huang [18] that

$$\begin{aligned} \mathbb{E}[y_i \overline{y}_j] &= \tfrac{9}{8\pi^2} \begin{pmatrix} \arccos^2(-\mathrm{Re}(\widehat{Y}_{ij})) + \omega \arccos^2(-\mathrm{Re}(\omega^2 \widehat{Y}_{ij})) \\ +\omega^2 \arccos^2(-\mathrm{Re}(\omega \widehat{Y}_{ij})) \end{pmatrix} \\ &= \tfrac{9}{8\pi^2} \begin{pmatrix} \arccos^2(-\mathrm{Re}(\theta Y_{ij})) + \omega \arccos^2(-\mathrm{Re}(\omega^2 \theta Y_{ij})) \\ +\omega^2 \arccos^2(-\mathrm{Re}(\omega \theta Y_{ij})) \end{pmatrix}. \end{aligned} \tag{2.4}$$

Let the partition of V that is corresponding to the round y_i in (2.3) be $\mathcal{S} = \{S_0, S_1, S_2\}$ and the objective value be denoted by $W(\mathcal{S})$. Define a real function

$$f(x) = \frac{9}{8\pi^2}\arccos^2(-x) - \arccos^2(\frac{1}{2}x),$$

for $x \in [-1, 1]$. Then, we have the following result.

Lemma 1

$$\mathbb{E}[W(\mathcal{S})] \geq \alpha(\theta)W^*, \tag{2.5}$$

where

$$\alpha(\theta) = \min_{-\frac{1}{2}\leq x<1} \frac{1 - f(\theta x)}{1 - x}.$$

Proof. Since $\arccos^2(x)$ is a convex function (see Lemma 10 of [10]), it follows that

$$\begin{aligned}
\mathrm{Re}(\mathbb{E}[y_i\overline{y}_j]) &= \tfrac{9}{8\pi^2}\mathrm{Re}\begin{pmatrix}\arccos^2(-\mathrm{Re}(\theta Y_{ij})) + \omega\arccos^2(-\mathrm{Re}(\omega^2\theta Y_{ij}))\\ +\omega^2\arccos^2(-\mathrm{Re}(\omega\theta Y_{ij}))\end{pmatrix}\\
&= \tfrac{9}{8\pi^2}\begin{pmatrix}\arccos^2(-\mathrm{Re}(\theta Y_{ij})) - \frac{1}{2}\arccos^2(-\mathrm{Re}(\omega^2\theta Y_{ij}))\\ -\frac{1}{2}\arccos^2\arccos^2(-\mathrm{Re}(\omega\theta Y_{ij}))\end{pmatrix}\\
&\leq \tfrac{9}{8\pi^2}\left(\arccos^2(-\mathrm{Re}(\theta Y_{ij})) - \arccos^2(-\tfrac{1}{2}\mathrm{Re}(\omega\theta Y_{ij} + \omega^2\theta Y_{ij}))\right)\\
&= \tfrac{9}{8\pi^2}\left(\arccos^2(-\mathrm{Re}(\theta Y_{ij})) - \arccos^2(\tfrac{1}{2}\mathrm{Re}(\theta Y_{ij}))\right)\\
&= f(\mathrm{Re}(\theta Y_{ij}))
\end{aligned}$$

By the linearity of exception and $\mathrm{Re}(Y_{ij}) \geq -\frac{1}{2}$, we have

$$\begin{aligned}
\mathbb{E}[W(\mathcal{S})] &= \tfrac{2}{3}\sum_{i<j} w_{ij}(1 - \mathrm{Re}\mathbb{E}[y_i\overline{y}_j])\\
&\geq \tfrac{2}{3}\sum_{i<j} w_{ij}(1 - f(\mathrm{Re}(\theta Y_{ij})))\\
&= \tfrac{2}{3}\sum_{i<j} \tfrac{(1-f(\mathrm{Re}(\theta Y_{ij})))}{1-\mathrm{Re}(Y_{ij})} w_{ij}(1 - \mathrm{Re}(Y_{ij}))\\
&\geq \alpha(\theta)W^*
\end{aligned}$$

The proof is then finished. □

In order to measure the whole deviation of partition $\mathcal{S}$ for the constraint $\sum_{i=1}^n y_i = 0$, we introduce a new random variable M, that is,

$$M = |S_0||S_1| + |S_0||S_2| + |S_2||S_1|.$$

Clearly,

$$M \leq \frac{(|S_0| + |S_1| + |S_0|)^2}{3} = \frac{n^2}{3}. \tag{2.6}$$

The equality $M = \frac{n^2}{3}$ holds if and only if $|S_0| = |S_1| = |S_0| = \frac{n}{3}$. Since

$$\mathbf{e}^T\mathbf{y} = |S_0| + \omega|S_1| + \omega^2|S_2|.$$

It follows that

$$\begin{aligned} M &= \tfrac{1}{3}n^2 - \tfrac{1}{3}(\mathbf{e}^T\mathbf{y})^*(\mathbf{e}^T\mathbf{y}) \\ &= \tfrac{1}{3}n^2 - \tfrac{1}{3}\sum_{i,j} y_i\overline{y}_j. \end{aligned} \tag{2.7}$$

Denote

$$b(\theta) = 1 - f(\theta),\ c(\theta) = \min_{-\frac{1}{2}\le x<1} \frac{f(\theta) - f(\theta x)}{1-x}.$$

Then we have the following result.

Lemma 2

$$\mathbb{E}[\frac{M}{M^*}] \ge \beta(\theta),$$

where $M^* = \frac{1}{3}n^2$, $\beta(\theta) = (1-\frac{1}{n})b(\theta) + c(\theta)$.

Proof. By (2.7), it follows that

$$\begin{aligned} \mathbb{E}[M] &= \tfrac{1}{3}n^2 - \tfrac{1}{3}\sum_{i,j}\mathbb{E}[y_i\overline{y}_j] = \tfrac{1}{3}n^2 - \tfrac{2}{3}\sum_{i<j}\mathrm{Re}(\mathbb{E}[y_i\overline{y}_j]) - \tfrac{1}{3}n \\ &= \tfrac{2}{3}\sum_{i<j}(1 - \mathrm{Re}(\mathbb{E}[y_i\overline{y}_j])) \\ &\ge \tfrac{2}{3}\sum_{i<j}(1 - f(\theta Y_{ij})) \\ &= \tfrac{2}{3}\sum_{i<j}[(1-f(\theta)) + (f(\theta) - f(\theta Y_{ij}))] \\ &= \tfrac{2}{3}\sum_{i<j}(1-f(\theta)) + \tfrac{2}{3}\sum_{i<j}(f(\theta) - f(\theta Y_{ij})) \\ &= \tfrac{2}{3}\sum_{i<j}(1-f(\theta)) + \tfrac{2}{3}\sum_{i<j}\tfrac{f(\theta)-f(\theta Y_{ij})}{1-Y_{ij}}(1-Y_{ij}) \\ &\ge \tfrac{2}{3}\sum_{i<j} b(\theta) + \tfrac{2}{3}c(\theta)\sum_{i<j}(1-Y_{ij}) \\ &= \tfrac{1}{3}[(1-\tfrac{1}{n})b(\theta) + c(\theta)]n^2 = \beta(\theta)M^*. \end{aligned}$$

This completes the proof. □

By the structure of M for Max 3-section, we surprisedly find that the form of $\beta(\theta)$ is exactly the same as the case of Max bisection considered in [17]. Note that θ is used in the rounding method. If $\theta = 0.4035$ is selected, it can get that, when $n > 2000$,

$$\alpha(0.4035) \ge 0.7369,$$

$$b(0.4035) \ge 0.7636, \quad c(0.4035) \ge 0.2279.$$

This obtains that

$$\beta(0.4035) \ge 0.9911.$$

We mention that the value $\beta(0.4035)$ is greater than $\beta(0.89)$ given by Ye[17]. This means, As far as averagely speaking, from (2.6) that our rounding method can get a near 3-section partition, but the performance ratio will decrease from 0.8360 to 0.7369.

3 Size Adjusting Procedure

For the sake of analysis, without loss of generality, we assume that $|S_0| = \max\{|S_0|, |S_1|, |S_2|\}$. If $|S_k| = \max\{|S_0|, |S_1|, |S_2|\}, k \neq 0$, then we may set $y_i^N = \overline{w}^k y_i, i = 1, \cdots, n$. The resulted new solution $\mathbf{y}^N = (y_1^N, \cdots, y_i^N)$ will not change the objective value and moreover, the new partition $\mathcal{S}^N = \{S_0^N, S_1^N, S_2^N\}$ based on $\mathbf{y}^N$ satisfies $|S_0^N| = \max\{|S_0^N|, |S_1^N|, |S_2^N|\}$. By the assumption, the partition $\mathcal{S} = \{S_0, S_1, S_2\}$ still exist four possible cases:

Case 1. $|S_0| \geq |S_1| \geq \frac{n}{3} \geq |S_2|$. **Case 2**. $|S_0| \geq \frac{n}{3} \geq |S_1| \geq |S_2|$.
Case 3. $|S_0| \geq |S_2| \geq \frac{n}{3} \geq |S_1|$. **Case 4**. $|S_0| \geq \frac{n}{3} \geq |S_2| \geq |S_1|$.

The size adjusting procedures of Case 3 and Case 4 are similar to Case 1 and Case 2. And because of the limitation of space, we mainly consider Case 1 and Case 2 is omitted for adjusting the partition of V from $\mathcal{S} = \{S_0, S_1, S_2\}$ to $\widetilde{\mathcal{S}} = \{\widetilde{S}_0, \widetilde{S}_1, \widetilde{S}_2\}$ such that $|\widetilde{S}_k| = n/3, k = 0, 1, 2$. Denote

$$\delta_0(i) = \sum_{j \in S_1 \cup S_2} w_{ij},\ i \in S_0,$$

$$\delta_{01}(i) = \sum_{j \in S_1} w_{ij},\ i \in S_0,\ \delta_{10}(i) = \sum_{j \in S_0} w_{ij}, i \in S_1,$$

$$\delta_{02}(i) = \sum_{j \in S_2} w_{ij},\ i \in S_0,\ \delta_{20}(i) = \sum_{j \in S_0} w_{ij},\ i \in S_2,$$

and

$$\delta_{12}(i) = \sum_{j \in S_2} w_{ij},\ i \in S_1,\ \delta_{21}(i) = \sum_{j \in S_1} w_{ij},\ i \in S_2.$$

Then, it follows from simple computation that

$$\delta_0(i) = \delta_{01}(i) + \delta_{02}(i),\ \text{for each } i \in S_0, \tag{3.1}$$

$$\sum_{i \in S_k} \delta_{kl}(i) = \sum_{i \in S_l} \delta_{lk}(i),\ k, l = 0, 1, 2, k \neq l$$

and

$$\begin{aligned} W(\mathcal{S}) &= \sum_{i \in S_0} \delta_0(i) + \sum_{i \in S_1} \delta_{12}(i) \\ &= \sum_{i \in S_0} \delta_{01}(i) + \sum_{i \in S_0} \delta_{02}(i) + \sum_{i \in S_1} \delta_{12}(i) \\ &= d_{01} + d_{02} + d_{12}. \end{aligned} \tag{3.2}$$

where $d_{01} = \sum_{i \in S_0} \delta_{01}(i)$, $d_{02} = \sum_{i \in S_0} \delta_{02}(i)$, $d_{12} = \sum_{i \in S_1} \delta_{12}(i)$.

Size adjusting Procedure for Case 1: SAPC1.

1. Calculate

$$m_{02} = \frac{\sum_{i \in S_0} \delta_{02}(i)}{|S_0|},\ m_{12} = \frac{\sum_{i \in S_1} \delta_{12}(i)}{|S_1|}.$$

2. If $m_{02} \geq m_{12}$. Let $S_1 = \{j_1, j_2, \cdots, j_{|S_1|}\}$, where $\delta_{12}(j_l) \geq \delta_{12}(j_{l+1}), l = 1, 2, \cdots, |S_1|$. set $\widetilde{S}_1 = \{j_1, j_2, \cdots, j_{\frac{n}{3}}\}$, $\widehat{S}_2 = S_2 \bigcup (S_1 \setminus \widetilde{S}_1)$ and renew to calculate
$$\delta'_{02}(i) = \sum_{j \in \widehat{S}_2} w_{ij},$$
for each $i \in S_0$. Let $S_0 = \{i_1, i_2, \cdots, i_{|S_0|}\}$, where $\delta'_{02}(i_k) \geq \delta'_{02}(i_{k+1})$. Set $\widetilde{S}_0 = \{i_1, i_2, \cdots, i_{\frac{n}{3}}\}$ and $\widetilde{S}_2 = \widehat{S}_2 \bigcup (S_0 \setminus \widetilde{S}_0)$.
3. If $m_{02} < m_{12}$. Let $S_0 = \{i_1, i_2, \cdots, i_{|S_0|}\}$, where $\delta_{02}(i_k) \geq \delta_{02}(i_{k+1}), k = 1, 2, \cdots, |S_0|$, set $\widetilde{S}_0 = \{i_1, i_2, \cdots, i_{\frac{n}{3}}\}$, $\widehat{S}_2 = S_2 \bigcup (S_0 \setminus \widetilde{S}_0)$ and then renew to calculate
$$\delta'_{12}(i) = \sum_{j \in \widehat{S}_2} w_{ij},$$
for each $i \in S_1$. Set $\widetilde{S}_1 = \{j_1, j_2, \cdots, j_{\frac{n}{3}}\}$ and $\widetilde{S}_2 = \widehat{S}_2 \bigcup (S_1 \setminus \widetilde{S}_1)$, where $\delta'_{12}(j_k) \geq \delta'_{12}(j_{k+1})$ here.
4. Return the current partition $\widetilde{\mathcal{S}} = \{\widetilde{S}_0, \widetilde{S}_1, \widetilde{S}_2\}$, stop.

Let $x_0 = |S_0|/n$, $x_1 = |S_1|/n$, $x_2 = |S_2|/n$. In order to estimate the bound of r, we first introduce the following lemma.

Lemma 3. *Let a, b, c, d, c_1 and d_1 be any positive real numbers and satisfy $c > c_1$, $d > d_1$ and*
$$\frac{a}{b} \geq \frac{c}{d}, \ \frac{c_1}{d_1} \geq \frac{c}{d}. \tag{3.3}$$
Then
$$\frac{a + c_1}{b + d_1} \geq \frac{a + c}{b + d}. \tag{3.4}$$

Now we state the bound of r using the following lemma.

Lemma 4. *For SAPC1, we have*
$$r(x) \geq \frac{2}{3(x_0 + x_1)},$$
where $x = (x_0, x_1)^T$.

Proof. When $m_{02} \geq m_{12}$, then subset S_1 is firstly adjusted to $\widetilde{S}_1$. It follows that
$$\frac{\sum_{i \in \widetilde{S}_1} \delta'_{12}(i)}{n/3} \geq \frac{\sum_{i \in S_1} \delta_{12}(i)}{|S_1|}, \tag{3.5}$$
where $\delta'_{12}(i) = \sum_{j \in \widehat{S}_2} w_{ij}$, for each $i \in \widetilde{S}_0$. Noting that the value $\sum_{i \in S_0} \delta_0(i)$ will not change when S_1 is adjusted to $\widetilde{S}_1$ and before S_0 is adjusted. Denote
$$\delta'_{01}(i) = \sum_{j \in \widetilde{S}_1} w_{ij}, i \in S_0, \ \text{ and } \ d'_{01} = \sum_{i \in S_0} \delta'_{01}(i).$$

Then

$$\sum_{i\in S_0}\delta_0(i) = d'_{01} + d'_{02},$$

where $d'_{02} = \sum_{i\in S_0}\delta'_{02}(i)$ and $\delta'_{02}(i)$ given by Step 2 of SAPC1. Since $\sum_{i\in S_0}\delta_0(i) > \sum_{i\in S_0}\delta_{02}(i)$, it follows that

$$\frac{\sum_{i\in S_0}\delta_0(i)}{|S_0|} > m_{02} \geq m_{12} = \frac{\sum_{i\in S_1}\delta_{12}(i)}{|S_1|}.$$

Combining (3.5) and Lemma 4, it yields that

$$\begin{aligned}\frac{\sum_{i\in S_0}\delta_0(i)+\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{|S_0|+n/3} &= \frac{d'_{02}+d'_{01}+\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{|S_0|+n/3}\\ &\geq \frac{d'_{01}+d'_{02}+\sum_{i\in S_1}\delta_{12}(i)}{|S_0|+|S_1|}\\ &= \frac{W(\mathcal{S})}{|S_0|+|S_1|}.\end{aligned} \tag{3.6}$$

For the average weights, $\frac{\sum_{i\in S_0}\delta'_{02}(i)}{|S_0|}$, between S_0 and $\widehat{S}_2$ before S_0 is adjusted to $\widetilde{S}_0$, there exist two possible cases. That is either

$$\frac{\sum_{i\in S_0}\delta'_{02}(i)}{|S_0|} \leq \frac{\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{n/3}, \quad \text{or} \quad \frac{\sum_{i\in S_0}\delta'_{02}(i)}{|S_0|} > \frac{\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{n/3}. \tag{3.7}$$

After S_0 is adjusted to $\widetilde{S}_0$ by SAPC1,

$$\frac{\sum_{i\in\widetilde{S}_0}\delta'_{02}(i)}{n/3} \geq \frac{\sum_{i\in S_0}\delta'_{02}(i)}{|S_0|} \tag{3.8}$$

holds always. If the first mathematical relationship of (3.7) holds, i.e.

$$\frac{\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{n/3} \geq \frac{\sum_{i\in S_0}\delta'_{02}(i)}{|S_0|},$$

then

$$\frac{d'_{01}+\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{n/3} > \frac{\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{n/3} \geq \frac{\sum_{i\in S_0}\delta'_{02}(i)}{|S_0|}. \tag{3.9}$$

Hence from (3.8), Lemma 4 and (3.6), we have

$$\begin{aligned}\frac{W(\widetilde{\mathcal{S}})}{2n/3} &= \frac{\sum_{i\in\widetilde{S}_0}\delta_0(i)+\sum_{i\in\widetilde{S}_1}\delta''_{12}(i)}{2n/3}\\ &= \frac{\sum_{i\in\widetilde{S}_0}\delta'_{02}(i)+d'_{01}+\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{2n/3}\\ &\geq \frac{\sum_{i\in S_0}\delta'_{02}(i)+d'_{01}+\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{|S_0|+n/3}\\ &= \frac{d'_{02}+d'_{01}+\sum_{i\in\widetilde{S}_1}\delta'_{12}(i)}{|S_0|+n/3}\\ &\geq \frac{W(\mathcal{S})}{|S_0|+|S_1|},\end{aligned}$$

where $\delta''_{12}(i) = \sum_{j\in\widetilde{S}_2} w_{ij}, i \in \widetilde{S}_1$. This yields $r(x) \geq \frac{2}{3(x_0+x_1)}$. Similarly, we can get the same conclusions if the second mathematical relationship of (3.7) holds. The proof is then finished. □

4 Analysis of the Algorithm

To obtain an expected lower bound of $W(\widetilde{\mathcal{S}})$, from previous section, we need to lower bound the expected value $\mathbb{E}[r(x) \cdot W(\mathcal{S})]/W^*$. However, it is essentially hard to calculate this expected value since $r(x)$, $W(\mathcal{S})$ and their products are random numbers. Our idea, initially due to Frieze and Jerrum [6], Ye [17] and later extended in [5,11,12,13,15], is to construct a family of artificial random variables, $z_{\mathcal{S}}$ say, whose expected value would be easily estimated and bounded, Particularly, whenever $z_{\mathcal{S}}$ meets its expectation, i.e. $z_{\mathcal{S}}$ is equal to or greater than its expectation, we can get a performance ratio $R(z_{\mathcal{S}})$ satisfying $r(x) \cdot W(\mathcal{S}) \geq R(z_{\mathcal{S}}) \cdot W^*$. Moreover, we will to select the $z_{\mathcal{S}}$ such that the largest $R(z_{\mathcal{S}})$ is hit.

Recall that $M = |S_0||S_1| + |S_0||S_2| + |S_2||S_1|$ and $M^* = n^2/3$. Let $\gamma \geq 0$ be given and define

$$\begin{aligned} z(\gamma) :&= \frac{W(\mathcal{S})}{W^*} + \gamma \frac{|S_0||S_1|+|S_0||S_2|+|S_2||S_1|}{n^2/3} \\ &= \frac{W(\mathcal{S})}{W^*} + \gamma \frac{M}{M^*}. \end{aligned} \tag{4.1}$$

To estimate the expectation $\mathbb{E}[z(\gamma)]$, we need to get the expectations $\mathbb{E}[W(\mathcal{S})/W^*]$ and $\mathbb{E}[M]$. By Lemma 1 and Lemma 2,

$$\mathbb{E}[z(\gamma)] = \mathbb{E}\Big[\frac{W(\mathcal{S})}{W^*}\Big] + \mathbb{E}\Big[\gamma \frac{M}{M^*}\Big] \geq R(\gamma) := \alpha(\theta) + \gamma\beta(\theta).$$

Since $0 < \frac{W(\mathcal{S})}{W^*} \leq 1$, $0 < \frac{M}{M^*} \leq 1$, the random variable $z(\gamma)$ is bounded above by a constant c_0, i.e. $0 < z(\gamma) < 1 + \gamma = c_0$ for a given γ. Thus, from the approach of [6,17] and Proposition 1 of [12], for any small $\epsilon > 0$, we may independently generate complex vector ξ at most $K = O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}))$ times, then we will obtain K values $z_i(\gamma)(i = 1, \cdots, K)$ of random variable $z(\gamma)$, denote $z^K(\gamma) = \max\limits_i\{z_i(\gamma)\}$. It follows that

$$Pr\{z^K(\gamma) \leq R(\gamma) - (c_0 - R(\gamma))\epsilon\} \leq \epsilon.$$

Since ϵ can be an arbitrarily small positive constant, we may assume that $z^K(\gamma) \geq R(\gamma)$ with probability almost one. Hence, for simplicity, we remove this term $(c_0 - R(\gamma))\epsilon$ and always assume that $z(\gamma) \geq R(\gamma)$, that is,

$$\frac{W(\mathcal{S})}{W^*} + \gamma \frac{M}{M^*} \geq \alpha(\theta) + \gamma\beta(\theta). \tag{4.2}$$

holds always.

Let $\lambda = \frac{W(\mathcal{S})}{W^*}$ and $\mu = M/M^*$. Recalling that $|S_k| = x_k n (k = 0, 1, 2)$ and $x = (x_0, x_1)^T$, we may write μ as $\mu(x) := \mu = x_0x_1 + (x_0 + x_1)(1 - x_0 - x_1)$. It follows from (4.2) that

$$\lambda \geq \alpha(\theta) + \gamma\beta(\theta) - \gamma\mu(x).$$

Applying Lemma 4, we have

$$\frac{W(\widetilde{\mathcal{S}})}{W^*} \geq r(x)\lambda = \frac{2[\alpha(\theta) + \gamma\beta(\theta) - \gamma\mu(x)]}{3(x_0 + x_1)}. \tag{4.3}$$

Denote $R_0(\theta,\gamma;x) = \frac{2[\alpha(\theta)+\gamma\beta(\theta)-\gamma\mu(x)]}{3(x_0+x_1)}$. Minimizing $R_0(\theta,x)$ with respect to $x_0, x_1 \in (0,1)$ yields that

$$x_0 = x_1 = \frac{\sqrt{\alpha(\theta)+\gamma\beta(\theta)}}{3\sqrt{\gamma}}$$

and

$$R_1(\theta,\gamma) := \min_x R_0(\theta,\gamma;x) = 2(\sqrt{\gamma(\alpha(\theta)+\gamma\beta(\theta))}-\gamma).$$

We mention that the form of $R_1(\theta,\gamma)$ is exactly coincide with the result of [17] for Max-bisection. Maximizing $R_1(\theta,\gamma)$ with respect to γ yields that

$$\gamma = \frac{\alpha(\theta)}{2\beta(\theta)}\Big(\frac{1}{\sqrt{1-\beta(\theta)}}-1\Big) \tag{4.4}$$

and

$$\mathcal{R}(\theta) := \max_\gamma R_1(\theta,\gamma) = \frac{\alpha(\theta)}{1+\sqrt{1-\beta(\theta)}}.$$

Based on the analysis above, we have the following theorem that extends the result of [17] to the Max 3-section problem.

Theorem 5. *For a fixed θ and any given $\gamma \in (\frac{\alpha(\theta)}{9-\beta(\theta)}, \frac{\alpha(\theta)}{1-\beta(\theta)}]$, if random variable $z(\gamma)$ meets its expectation, i.e. $z(\gamma) \geq \alpha(\theta)+\gamma\beta(\theta)$, then*

$$W(\widetilde{\mathcal{S}}) \geq R_1(\theta,\gamma)\cdot W^*.$$

In particular, if γ is given by (4.4) which belongs to $(\frac{\alpha(\theta)}{9-\beta(\theta)}, \frac{\alpha(\theta)}{1-\beta(\theta)}]$, then

$$W(\widetilde{\mathcal{S}}) \geq \mathcal{R}(\theta)\cdot W^*. \tag{4.5}$$

Theorem 5 indicates that for any given $\theta \in [0,1]$, if only random variable $z(\gamma)$ of (4.1) meets its expectation, our rounding method can generate a $\mathcal{R}(\theta)$-approximate algorithm for Max 3-section. Hence, we can maximize $\mathcal{R}(\theta)$ with respect to variable θ, such that the approximate ratio $\mathcal{R}(\theta)$ is as large as possible. Let θ^* maximize $\mathcal{R}(\theta)$ and $\mathcal{R}^*$ is the maximum value of $\mathcal{R}(\theta)$, that is

$$\theta^* = \arg\max_{\theta\in[0,1]} \mathcal{R}(\theta).$$

Then

$$\mathcal{R}^* = \mathcal{R}(\theta^*) = \frac{\alpha(\theta^*)}{1+\sqrt{1-\beta(\theta^*)}}$$

is the best approximate ratio for our rounding method.

If $\theta = 0.4035$ is used in our rounding method, then, for sufficiently large n (e.g. $n > 10^4$),

$$b(0.4035) \geq 0.7636, \quad c(0.4035) \geq 0.2279$$

implies that

$$\beta(0.4035) = \Big(1 - \frac{1}{n}\Big)b(0.4035) + c(0.4035) \geq 0.9914,$$

and

$$\alpha(0.4035) \geq 0.7369.$$

This yields that

$$\mathcal{R}^* \geq \frac{\alpha(0.4035)}{1 + \sqrt{1 - \beta(0.4035)}} \geq 0.6733.$$

This sightly improves the $\frac{2}{3} + O(\frac{1}{n^3})$ lower bound obtained by Andersson [1] and Gaur et al. [8] for sufficiently large n.

5 Conclusions

We proposed a random rounding method for the Max 3-section problem based on complex semidefinite programs. the proposed method can beat the 2/3 lower bound. It is very worth to consider the numerical application for the proposed algorithm. We will further discuss these problems in the sequel work.

References

1. Andersson, G.: An Approximation Algorithm for Max p-Section. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 237–247. Springer, Heidelberg (1999)
2. Barahona, F., Grötschel, M., Reinelt, G.: An Application of Combinotiorial Optimization to Statistical Physics and Circuit Layout Design. Oper. Res. 36, 493–513 (1988)
3. Bertsimas, D., Ye, Y.: Semidefinite Relaxations, Multivariate Normal Distributions, and Order Statistics. In: Handbook of Combinatorial Optimization, vol. 3, pp. 1–19. Kluwer Academic Publishers, Dordrecht (1998)
4. Feige, U., Langberg, M.: Approximation Algorithms for Maximization Problems Arising in Graph Partitioning. Journal of Algorithms 41, 174–211 (2001)
5. Feige, U., Langberg, M.: The RPR^2 Rounding Technique for Semidefinite Programs. Journal of Algorithms 60, 1–23 (2006)
6. Frieze, A., Jerrum, M.: Improved Approximation Algorithms for MAX k-CUT and MAX BISECTION. In: Balas, E., Clausen, J. (eds.) IPCO 1995. LNCS, vol. 920, pp. 1–13. Springer, Heidelberg (1995)
7. Garey, M., Johnson, D., Stochmeter, L.: Some Simplified NP-Complete Graph Problems. Theoret. Comput. Sci. 1, 237–267 (1976)
8. Gaur, D.R., Ramesh, K., Kohli, R.: The Capacitated Max k-Cut Problem. Math. Program 115, 65–72 (2008)
9. Goemans, M.X., Williamson, D.P.: Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. J. Assoc. Comput. Mach. 42, 1115–1145 (1995)

10. Goemans, M.X., Williamson, D.P.: Approximation Algorithms for MAX-3-CUT and other Problems via Complex Semidefinite Programming. Journal of Computer and System Sciences 68, 442–470 (2004)
11. Halperin, E., Zwick, U.: A Unified Framework for Obtaining Improved Approximation Algorithms for Maximum Graph Bisection Problems. In: Aardal, K., Gerards, B. (eds.) IPCO 2001. LNCS, vol. 2081, pp. 202–217. Springer, Heidelberg (2001)
12. Han, Q., Ye, Y., Zhang, J.: An Improved Rounding Method and Semidefinite Programming Relaxation for Graph Partition. Math. Program. Ser. B 92, 509–535 (2002)
13. Jäger, G., Srjvastav, A.: Improved Approximation Algorithms for Maximum Graph Partitioning Problems. Journal of Combinatorial Optimization 10, 133–167 (2005)
14. Karp, R.M.: Reducibility among Combinatorial Problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
15. Ling, A.-F., Xu, C.-X., Xu, F.-M.: A Discrete Filled Function Algorithm Embedded with Continuous Approximation for Solving Max-Cut Problems. European Journal of Operational Research 197, 519–531 (2009)
16. Xu, C., Zhang, J.: Survey of Quasi-Newton Equations and Quasi-Newton Methods for Optimization. Annals of Operations Reseach 103, 213–234 (2001)
17. Ye, Y.: A 0.699-Approximation algorithm for Max-Bisection. Math. Program. 90, 101–111 (2001)
18. Zhang, S., Huang, Y.: Complex Quadratic Optimization and Semidefinite Programming. SIAM J. Optm. 16, 871–890 (2006)

Hamiltonian Decomposition of Some Interconnection Networks

Hai-zhong Shi and Pan-feng Niu

College of Mathematics and Information Science, Northwest Normal University
Lanzhou Gansu Province 730070, China
shihaizhong@nwnu.edu.cn

Abstract. Cayley graphs arise naturally in interconnection networks design in study of Akers and Krishnamurthy in 1989 [1]. Lakshmivarahan, Jwo and Dhall studied a number of interconnection networks as graphs in 1993 [12]. In this paper, we propose some conjectures related to complete-transposition graph, alternating-group graph, folded hypercube and binary orthogonal graph, respectively. The conjectures claim that each of these graphs is Hamiltonian decomposable. In addition, we prove that above conjectures are true for smaller order of the networks.

Keywords: Cayley graph; Hamiltonian decomposition.

1 Introduction

The ultimate utility of a parallel computer is largely dependent on the properties of the interconnection network that connects processors to memory or processors among themselves. Almost all of the interconnection network may broadly be classified into two categories-dynamic and static networks. Static networks can be conveniently modeled using tools from graph theory [5]. According, the processors correspond to the vertices and the communication links between processors are edges connecting the vertices. Henceforth we will use the terms networks and graphs interchangeably. If each processor in the network is connected to a fixed number of neighbors, then the underlying graph is regular graph. We consider the networks based on regular graphs. There are several considerations leading to the choice of a network in the development of parallel computers on a commercial basis. The simplest of these include the degree, diameter,the distribution of the node disjoint paths between a pair of vertices in the graph. More complex attributes need to be considered. These include:

(a) optimal algorithms for various made of packet communication,
(b) embeddability,
(c) Hamiltonian decomposition,
(d) symmetry properties,
(e) recursive scalability.

In this paper, we discuss Hamiltonian decomposition of the networks presented in [12].

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 231–237, 2009.

Let G be a regular graph with edge set $E(G)$. We say that G is Hamiltonian decomposable if either (i) $deg(G) = 2k$ and $E(G)$ can be partitioned into k Hamiltonian cycles, or (ii) $deg(G) = 2k + 1$ and $E(G)$ can be partitioned into k Hamiltonian cycles and a perfect matching as in [6,2].

2 Main Results

We first study the Hamiltonian decompositions of complete-transposition graphs. Complete-transposition graphs CT_n are the Cayley graphs generated by $\Omega_1 = \{(ij)|1 \leqslant i < j \leqslant n\}$. in the other words, graph $CT_n =< V, E >$ is a Cayley graph of the symmetric group S_n based on the generator set $\Omega_1 = \{(ij)|1 \leqslant i < j \leqslant n\}$. CT_n is a regular bipartite graph of degree $\frac{n(n-1)}{2}$ and order $n!$. For CT_n, we have

Conjecture 1. If $\frac{n(n-1)}{2}$ is even, then CT_n is a union of $\frac{n(n-1)}{4}$ edge-disjoint Hamiltonian cycles; If $\frac{n(n-1)}{2}$ is odd, then CT_n is union of $(\frac{n(n-1)}{2} - 1)/2$ edge-disjoint Hamiltonian cycles and a perfect matching.

In other words, CT_n is Hamiltonian decomposable. In the following we give an algorithm for searching above Hamiltonian cycles.

Let $n = 2, \frac{n(n-1)}{2} = 1$. Clearly, CT_1 is a perfect matching: $12 - 21$.

Let $n = 3, \frac{n(n-1)}{2} = 3$. CT_3 is union of a Hamiltonian cycle H_1 and a perfect matching M_1, where H_1 and M_1 follows:

H_1: $123 - 213 - 231 - 321 - 312 - 132 - 123$

M_1: $123 - 321$, $213 - 312$, $231 - 132$

These verified that Conjecture 1 is true for $n = 2, 3$.

We next study the Hamiltonian decomposition of extension of Hypercube version 1 denoted by EC_n. Let $\Omega_2 = \{(3i-2, 3i-1)|1 \leqslant i \leqslant n\} \cup \{(3i-2, 3i)|1 \leqslant i \leqslant n\}$. $(< 3n >, \Omega_2)$ is the transposition graph of order $3n$. EC_n is the Cayley graph as generated by Ω_2 in [12]. The degree of EC_n is $2n$, the order of EG_n is 6^n. For EC_n, we have

Conjecture 2. EC_n is union of n edge-disjoint Hamiltonian cycles.

In other words, EC_n is Hamiltonian decomposable. In the following we give an algorithm searching the Hamiltonian cycles.

Let $n = 1$, EC_1 is one Hamiltonian cycle H_1, where H_1 is defined as follows:
H_1: $123 - 213 - 312 - 132 - 231 - 321 - 123$.

This verify that Conjecture 2 is true for $n = 1$.

We then study the Hamiltonian decomposition of extension of Hypercube version 2 denoted by FC_n. Let $\Omega_3 = \Omega_2 \cup \{(3i - 13i)|1 \leqslant i \leqslant n\} = \{(3i - 23i - 1)|1 \leqslant i \leqslant n\} \cup \{(3i - 23i)|1 \leqslant i \leqslant n\}$. $(< 3n >, \Omega_3)$ is the transposition graph of order $3n$. Extension of hypercube of version 2 FC_n is the Cayley graph as generated by Ω_3 in [12]. The degree and the order of EC_n are $3n$ and 6^n, respectively. We thus have:

Conjecture 3. FC_n is a union of n edge-disjoint Hamiltonian cycles and one perfect matching.

In other words, FC_n is Hamiltonian decomposable. In the following we give an algorithm for searching above n edge-disjoint Hamiltonian cycles and one perfect matching.

Let $n = 1$, FC_1 is union of the Hamiltonian cycle H_1 and the perfect matching where H_1 and M_1 follows:

H_1: $123 - 213 - 312 - 132 - 231 - 321 - 123$

M_1: $123 - 132$, $213 - 231$, $312 - 321$

This verifies that Conjecture 3 is true for $n = 1$.

We now study the Hamiltonian decomposition of alternating-group graph. The alternating group $A_n \subseteq S_n$ consists of the set of all even permutations. It can be show that $\Omega_4 = \{(2i), (i2) | 3 \leqslant i \leqslant n\}$ generates A_n as in [13]. The Cayley graph generated by Ω_4 is called alternating group graph AG_n as in [12,10]. AG_n is a regular graph with degree $2(n-2)$, order $\frac{n!}{2}$ and the number of edges $\frac{(n-2)n!}{2}$. Hence we have

Conjecture 4. AG_n is a union of $n-2$ edge-disjoint Hamiltonian cycles.

In other words, AG_4 is Hamiltonian decomposable. In the following we give an algorithm searching the above Hamiltonian cycles.

Let $n = 3$, AG_3 is a Hamiltonian cycle H_1: $123 - 231 - 312$

Let $n = 4$. AG_4 is union of two edge-disjoint Hamiltonian cycles H_1 and H_2, where

H_1: $1342 - 3412 - 4213 - 2143 - 1423 - 3124 -$
$2314 - 1234 - 4132 - 2431 - 4321 - 3241 - 1342$

H_2: $1342 - 2143 - 3241 - 2431 - 1234 - 3124 -$
$4321 - 1423 - 4213 - 2314 - 3412 - 4132 - 1342$

Therefore, AG_3 and AG_4 both are Hamiltonian decomposable.

Now we study the Hamiltonian decomposition of base-b hypercube or generalized hypercube. Base-b hypercube or generalized hypercube $GC_n(b)$ is the Cayley graph generated by Ω_5, which is defined as follows:

$$\Omega_5 = \{(jj+1\cdots j+b-1)^i | j = 1+bk, 0 \leqslant k \leqslant n-1 \text{ and } 1 \leqslant i \leqslant b-1\},$$

where $b \geqslant 2$ and $n \geqslant 1$ are integers as in [12]:

$$GC_n(b) \;=\; GC_{n-1}(b) \times K_b,$$

where '$\times$' denotes the standard Cartesian product operation on graphs [9,17]. It can be readily verified that $GC_n(2)$ is the n-dimensional binary (base $b = 2$) hypercube as in [11,14,15] and $GC_n(b)$ corresponds to the n-dimensional base-b hypercube as in [11,3].

Let $V(GC_n(b)) = \{(12\cdots b)^{i_1}(b+1b+2\cdots 2b)^{i_2}((n-1)b+1(n-1)b+2\cdots nb)^{i_n} | 0 \leqslant i_j \leqslant b-1 \text{ and } 1 \leqslant j \leqslant n\}$. Then the order and degree of $GC_n(b)$ are b^n and $(b-1)n$, respectively. Therefore, we have

Conjecture 5. If $(b-1)n$ is even then $GC_n(b)$ is a union of $\frac{(b-1)n}{2}$ edge-disjoint Hamiltonian cycles; $If (b-1)n$ is odd, then $GC_n(b)$ is a union of $\frac{(b-1)n-1}{2}$ edge-disjoint Hamiltonian cycles and a perfect matching.

In other words, $GC_n(b)$ is Hamiltonian decomposable. In the following, we give an algorithm searching above Hamiltonian cycles.

Clearly, $GC_n(2)$ is a Hamiltonian cycle H_1: $1243 - 1234 - 2134 - 2143$.

$GC_2(3)$ is a union of 2 edge-disjoint Hamiltonian cycles H_1 and H_2, where

H_1: $231564 - 123564 - 312564 - 312645 -$
$123645 - 123456 - 312456 - 231456 - 231645 - 231564$

H_2: $231564 - 312564 - 312456 - 312645 -$
$231645 - 123645 - 123564 - 123456 - 231456 - 231564$

Hence, for $GC_2(2)$ and $GC_2(3)$, Conjecture 5 is true.

We move on to study the Hamiltonian decomposition of generalized base-b orthogonal graph denoted by $OG_4(b)$. As in [12], let $\sum_b = \{0, 1, \cdots, b-1\}$. Then $(\sum_b)^n$, the set of all strings of length n over $\sum_b$, denotes the set of all base-b integers of length n. We define an encoding as follows:

$$f_b : V(GC_n(b)) \longrightarrow \Big(\sum\nolimits_b\Big)^n \tag{1}$$

as $f_b((1\ 2\cdots b)^{i_1}(b{+}1\ b{+}2\cdots 2b)^{i_2}((n{-}1)b{+}1\ (n{-}1)b{+}2\cdots nb)^{i_n}) = i_1\ i_2\cdots i_n$.

We can denote $GC_n(b) = (V, E)$ as the base-b hypercube of dimension n, where

$$V = \Big\{x_1\ x_2\cdots x_n \big| x_i \in \sum\nolimits_b\Big\} = \Big(\sum_b\Big)^n,$$

and $E = \{(x, y) | x = x_1\ x_2\cdots x_n, y = y_1\ y_2\cdots y_n$ and for some $i, 1 \leqslant i \leqslant n, x_i \neq y_i$ and $x_j = y_j$, for $j \neq i\}$.

A further generalization of $GC_n(b)$ is presented in [12]. For $1 \leqslant i \leqslant n$ and $1 \leqslant k \leqslant b-1$, define $Z_i(k) = \{x | x \in \sum_b^n$ and $x_i = k$ and $x_j = 0$ where $j \neq i\}$. If $y(i) \in (\sum_b)^n, 1 \leqslant i \leqslant m$, then define

$$\prod_{i=1}^{m} y(i) = y(1)y(2)\cdots y(m)$$

the concatenation of $y(1), y(2), \cdots, y(n)$. Clearly, for each $1 \leqslant k \leqslant b-1, Z_i(k) \subseteq (\sum_b)^n$ and it can be verified that

$$Z = \prod_{i=1}^{n}\Big(\prod_{k=1}^{b-1} Z_i(k)\Big)$$

is an encoding of Ω_5 using f_b which is defined in equation (1).

Let $\mathcal{A}$ be a collection of non-empty subsets of $< n >$. Define ,for $A \in \mathcal{A}$,

$$Z(A) = \prod_{i \in A} \Big(\prod_{k=1}^{b-1} Z_i(b) \Big).$$

Given $\mathcal{A}$, consider

$$\Omega_6 = \Omega_5 \bigcup \Big\{ \bigcup_{A \in \mathcal{A}} Z(A) \Big\}.$$

The Cayley graph generalized by this generator set Ω_6 is called base-b generalized orthogonal graph of dimension n and is denoted by $OG_n(b)$.

When $b = 2, \mathcal{A} = \{A\}$ and $A =< n >$, we have

$$\Omega_6 = \{(2i-1\ 2i) | 1 \leqslant i \leqslant n\} \cup \{(12)(34)\cdots(2n-1\ 2n)\}$$

generated the binary folded hypercube of dimension n as in [8] and denoted $BFH(n)$. Thus we have

Conjecture 6. If n is even, then the binary folded hypercube of dimension n $BFH(n)$ is a union of $\frac{n}{2}$ edge-disjoint Hamiltonian cycles and a perfect matching; If n is odd, then the binary folded hypercube of dimension n $BFH(n)$ is a union of $\frac{n+1}{2}$ edge-disjoint Hamiltonian cycles.

In other words, $BFH(n)$ is Hamiltonian decomposable. Furthermore, we give an algorithm searching above Hamiltonian cycles.

Let $n = 3$. $BFH(3)$ is a union of edge-disjoint Hamiltonian cycles H_1 and H_2, where

H_1: 123456 − 213456 − 214356 − 124356 −
124365 − 214365 − 213465 − 123465 − 123456

H_2: 123456 − 124356 − 213465 − 213456 −
124365 − 123465 − 214356 − 214365 − 123456

Hence, for $BFH(3)$, Conjecture 6 is true.

In the end we study the Hamiltonian decomposition of CG_n^i. It is well known [10] that for any $1 \leqslant i < n$, the set $\{(1\ 2 \cdots n), (i\ i+1)\}$ generates S_n. However, since this set does not satisfy the condition for being a generator set for a Cayley graph, we define

$$\Omega_7^i = \{(1\ 2\ 3 \cdots n), (1\ n\ n-1 \cdots 3\ 2), (i\ i+1)\}$$

as a generator set. The Cayley graph CG_n^i generated by Ω_7^i is a regular graph of degree three. We thus have

Conjecture 7. For any $1 \leqslant i < n$, CG_n^i is a union of a Hamiltonian cycle and a perfect matching.

Let $n = 3, i = 1$. CG_3^1 is a union of a Hamiltonian cycle H_{11}^3 and a perfect matching M_{11}^3, where

$H_{11}^3 : 123 - 231 - 312 - 132 - 321 - 213 - 123;$

$M_{11}^3 : 123 - 312,\ 231 - 321,\ 132 - 213.$

Let $n = 3, i = 2$. CG_3^2 is a union of a Hamiltonian cycle H_{21}^3 and a perfect matching M_{21}^3, where

$H_{21}^3 : 123 - 231 - 312 - 321 - 213 - 132 - 123;$

$M_{21}^3 : 123 - 312,\ 231 - 213,\ 312 - 132.$

Let $n = 4, i = 1$. CG_4^1 is union of a Hamiltonian cycle H_{11}^4 and a perfect matching M_{11}^4, where

H_{11}^4: $1234 - 2134 - 4213 - 2413 - 3241 - 2341 -$
$3412 - 4312 - 3124 - 1324 - 4132 - 1432 -$
$4321 - 3421 - 1342 - 3142 - 2314 - 3214 -$
$2143 - 1243 - 2431 - 4231 - 1423 - 4123 - 1234.$

$M_{11}^4 : 1234 - 2341,\ 2134 - 1342,\ 4213 - 3421,$
$2413 - 4132,\ 3241 - 1342,\ 3412 - 4123,$
$4312 - 2431,\ 3124 - 1243,\ 1432 - 2143,$
$4321 - 3214,\ 3142 - 1423,\ 2314 - 4231.$

Hence, for CG_3^1, CG_3^2, CG_4^1, Conjecture 7 is true.

References

1. Akers, S.B., Krishnamurthy, B.: A Group-Theoretic Modle for Symmetric Interconnection Networks. IEEE Trans. Comput. 38, 555–565 (1989)
2. Alspach, B., Bermond, J.C., Sotteau, D.: Decompositions into Cycles I: Hamilton Decompositions. In: Hahn, G. (ed.) Cycles and Rays, pp. 8–19. Kluwer Academic Publishers, Netherlands (1990)
3. Bhuyan, L.N., Agrawal, D.T.: Generalized Hypercube and Hyperbus Stuctures for a Computer Network. IEEE Trans. Comput. 33, 323–333 (1984)
4. Biggs, N.L.: Algebraic Graph Theory. Cambrridge University Press, Cambridge (1974)
5. Bondy, J.A., Murthy, U.S.R.: Graph Theory with Application. American Elserier, New York (1977)
6. Curren, S.J., Gallian, J.A.: Hamiltonian Cycles and Paths in Cayley Graphs and Digraphs - A Survey. Discrete Mathematics 156, 1–18 (1996)
7. Du, D.Z., Hsu, D.F., Huang, F.K., Zhang, X.M.: The Hamiltonian Property of Gemeralized de Brujin Digraphs. Journal of Combinatorial Theory Series B 52(1), 1–8 (1991)
8. El-Amawy, A., Latitifi, S.: Properties and Performance of Folded Hypercubes. IEEE Trans Parallel Ditribute Syst. 2(1), 31–42 (1991)
9. Harary, F.: Graph Theory. Addison-Wesley, Reading (1969)
10. Jwo, J.-S.: Analysis of Interconnection Networks Based on Cayley Graphs Related to Permutation Groups. Ph.D. Dissertation, School Electrical Engineering and Computer Science, University of Oklahoma, Norman, OK (1991)
11. Lakshmivarahan, S., Dhall, S.K.: A New Hierarchy of Hypercube Interconnection Schemes for Paralled Computers. J. Supercomputer 2, 81–108 (1988)
12. Lakshmivarahan, S., Jwo, J.-S., Dhall, S.K.: Symmetry in Interconnection Networks Based on Cayley Graphs of Permutation Groups: a Survey. Paralled Computing 19(4), 361–407 (1993)

13. Ledermann, W.: Introduction to the Theory of Finite Groups. Oliver and Boyd, London (1949)
14. Saad, Y., Schultz, M.H.: Topological Properties of Hypercubes. Technical Report DCR-RR-389, Department of Computer Science, Yale University (1985)
15. Seitz, C.L.: The Cosmic Cubes. Comm. ACM 28, 22–33 (1985)
16. Shi, H.-Z.: A Ring-Theoritic Model-For Interconnection Network. Ph. D. Dissertation, Institute of Applied Mathematics, Chinese Academy of Sciences, Beijing, China (1998)
17. White, A.T.: Graphs, Groups and Surfaces. North-Holland Mathematics Studies 8 (1984)
18. Xu, J.-M.: Jopological Structure and Analysis of Interconnection Networks. Kluwer Academic Publishers, Dordrecht (2001)

Infinite Family from Each Vertex k-Critical Graph without Any Critical Edge

Jixing Wang

Department of Mathematics, Yunnan University, Kunming 650091, China
kingwang@ynu.edu.cn

Abstract. In 1970, Dirac conjectured that for each integer $k \geq 4$, there exists a vertex k-critical graph without any critical edge. In this paper, we introduce strict-vertex critically $(k-1)$-de-chromatic pair, strict 2-vertex decomposition graph and so on. By studying their relevant properties, we obtain that any vertex k-critical graph without any critical edge generates an infinite family of vertex k-critical graphs without critical edges.

Keywords: Decomposition, Composition, De-chromatic, Critical graph, Dirac conjecture.

1 Introduction

Dirac conjectured that for each integer $k \geq 4$, there exists a vertex k-critical graph without any critical edge, in 1970. A formal statement of this conjecture can be found in [2]. There was little progress within near 20 years after the presentation of the conjecture. It is excited that more and more papers such as [1,3] have presented some good results on the conjecture since 1992. Especially, in 2002, Lattanzio [6] presented vertex k-critical graphs without critical edges for all $k \geq 5$ except prime integers.

However, there remains a problem: almost each method is special. In other words, if G is found as a vertex k-critical graph without any critical edge, then no others relative to G, except G its self, are found. It raises a problem: Can we construct other vertex k-critical graphs without critical edges from a given vertex k-critical graph without any critical edge?

In order to solve this problem, we introduce $(k-1)$-de-chromatic pairs and strict-vertex critically $(k-1)$-de-chromatic pairs which are obtained by studying vertex decomposition of vertex k-critical graph without any critical edge. Then, we obtain, for each $k \geq 4$, any vertex k-critical graph without any critical edge constructs an infinite family of vertex k-critical graphs without critical edges.

The graphs considered in this paper are finite, undirected and simple. For a given graph G, we denote the vertex set of G by $V(G)$ and the edge set of G by $E(G)$. Also, we set $n(G) = |V(G)|$. A graph G is said to have a k-coloring if and only if its all vertices can be painted with k different colors such that no two adjacent vertices receive the same color. k-colorings are often denoted by $P_1, P_2, P_3, \cdots$ in this paper. A graph G is said to be k-colorable if it has

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 238–248, 2009.

a k-coloring. The chromatic number $\mathcal{X}(G)$ of G is the least k such that G is k-colorable. A graph G is k-chromatic if $\mathcal{X}(G) = k$. k distinct colors, used by a k-coloring of a k-chromatic graph, are often denoted by $C_1, C_2, \cdots, C_k$ in the text, and also by $1, 2, \cdots, k$ in the figures. The color of vertex v under coloring P is written as $C^P(v)$ or simply $C(v)$.

A graph G is said to be vertex $k-$critical if $\mathcal{X}(G) = k$ and $\mathcal{X}(G-v) \leq k-1$ for every vertex $v \in V(G)$. An edge $e \in E(G)$ is said to be critical in G whenever $\mathcal{X}(G-e) \leq \mathcal{X}(G) - 1$. A graph G is said to be $k-$critical if $\mathcal{X}(G) = k$ and each edge $e \in E(G)$ is critical. Let H be a subgraph of G. $N_H(v)$, called the neighborhood of vertex v in subgraph H, is defined as all those vertices of H which are adjacent to v.

2 Conceptions and Properties

In this section, we will introduce some basic conceptions, such as k-de-chromatic pair and strict 2-vertex decomposition, which play an important role in this paper.

Definition 1. *Let u and v be two nonadjacent vertices of a graph G. u and v are said to be a* k-de-chromatic pair *of G if and only if $C^P(u) \neq C^P(v)$ under any k-coloring P of G, denoted by $u \overset{G,k}{--} v$, or $u \overset{G}{--} v$ or $u \overset{k}{--} v$ or simply $u--v$.*

If u and v are a k-de-chromatic pair of G, we often say u and v are k-de-chromatic in G, or G has a k-de-chromatic pair. Thus two vertices u and v of a graph G are not k-de-chromatic in G if and only if there is a k-coloring P of G such that $C^P(u) = C^P(v)$. If G has a k-de-chromatic pair, then $n(G) \geq k+1$.

Example 1. In Figure 1, two vertices h and t of Graph (1) are 3-de-chromatic, since $C^P(h) \neq C^P(t)$ under any 3-coloring of Graph (1); and, h and t, u and v of Graph (2) are also.

Definition 2. *Let P_1 and P_2 be two k-colorings of a k-chromatic graph G. P_2 is said to be a* k-coloring of G based on P_1 *if and only if there is a nonempty vertex set $V' \subset V(G)$ such that $C^{P_1}(v) = C^{P_2}(v)$ for all $v \in V'$.*

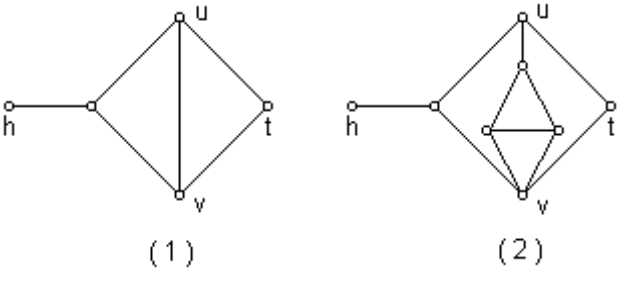


Fig. 1. Some 3-de-chromatic pairs

Theorem 1. *If G has a k-de-chromatic pair $u--v$, then $\mathcal{X}(G) = k$.*

Proof. According to the definition of k-de-chromatic pair, G has at least one k-coloring. Thus, $\mathcal{X}(G) \leq k$. We will now prove $\mathcal{X}(G) > k-1$ by converse method. Suppose there is a $(k-1)$-coloring P of G, whose $k-1$ different painting colors are denoted as $C_1, C_2, \cdots, C_{k-1}$.

If $C^P(u) = C^P(v)$, then there are at leat two vertices w and w' (other than u and v) of G painted by some color of $C_1, C_2, \cdots, C_{k-1}$, since G has at least $k+1$ vertices. Painting one of them such as w by a new color C_k to P, we obtain a k-coloring P_1 of G based on the $(k-1)$-coloring P such that $C^{P_1}(u) = C^{P_1}(v) = C^P(u) = C^P(v)$, and $C^{P_1}(w) = C_k$, and $C^{P_1}(v') = C^P(v')$ for any $v' \in V(G) - \{u, v, w\}$. Contrary to that u and v are k-de-chromatic in G.

If $C^P(u) \neq C^P(v)$, there are three cases.

The first case is that there are two vertices (other than u and v) which are painted by $C^P(u)$ and $C^P(v)$ respectively. Then, painting u and v by a new color to P such as C_k, we obtain a k-coloring P_2 of G based on the $(k-1)$-coloring P such that $C^{P_2}(u) = C^{P_2}(v) = C_k$, and $C^{P_2}(v') = C^P(v')$ for all $v' \in V(G) - \{u, v\}$. Contrary.

The second case is that there are no other vertices which are painted by $C^P(u)$ or $C^P(v)$. Then, there are at least four vertices (other than u and v) painted by two different colors of $\{C_1, C_2, \cdots, C_{k-1}\} - \{C^P(u), C^P(v)\}$, since $n(G) - [(k-1) - 2] \geq k + 1 - k + 3 = 4$. Choose two of them, denoted by a and b, and let a and b be painted by $C^P(u)$ and $C^P(v)$ respectively such that not to decrease $k-1$ different painting colors of P so that this case is put into the first case.

The third case is that case which belongs to neither the first nor the second. We can deal with it similarly.

These imply $\mathcal{X}(G) > k-1$. Thus, $\mathcal{X}(G) = k$. □

According to this theorem, we have that, if u and v are k-de-chromatic in G, then they are absolutely not t-de-chromatic for $t \neq k$.

Definition 3. *Let u and v be a k-de-chromatic pair of a graph G. An edge e of G is said to be* k-de-chromatically critical for u and v *if and only if u and v are not k-de-chromatic in graph $G - e$. The collection of all k-de-chromatically critical edges for u and v is denoted by $CriE(u \overset{G,\,k}{--} v)$ or $CriE(u \overset{G}{--} v)$ or $CriE(u \overset{k}{--} v)$ or simply $CriE(u--v)$. A vertex $w \in V(G) - \{u, v\}$ is said to be* k-de-chromatically critical for u and v *if and only if u and v are not k-de-chromatic in graph $G - w$. The collection of all k-de-chromatically critical vertices is denoted by $CriV(u \overset{G,\,k}{--} v)$ or $CriV(u \overset{G}{--} v)$ or $CriV(u \overset{k}{--} v)$ or simply $CriV(u--v)$.*

Let u and v be a k-de-chromatic pair of a graph G. If an edge e of G is k-de-chromatically critical for u and v, we often written as *e is critical for u and v in G*. Similarly, if a vertex w of G is k-de-chromatically critical for u and v, we often written as *w is critical for u and v in G*.

Definition 4. *Let u and v be a k-de-chromatic pair of a graph G. u and v are said to be a* vertex critically k-de-chromatic pair of G *if and only if $CriV(u \overset{G,\,k}{--} v) = V(G) - \{u, v\}$, denoted by $[u \overset{G,\,k}{--} v\}$ or $[u \overset{G}{--} v\}$ or $[u \overset{k}{--} v\}$ or simply $[u--v\}$. u and v are said to be a* strict-vertex critically k-de-chromatic pair *of G if and only if $CriV(u \overset{k}{--} v) = V(G) - \{u, v\}$ and there is no k-de-chromatically critical edge for u and v, denoted by $\{u \overset{G,\,k}{--} v\}$ or $\{u \overset{G}{--} v\}$ or $\{u \overset{k}{--} v\}$ or simply $\{u--v\}$.*

In other words, u and v are a strict-vertex critically k-de-chromatic pair of G if u and v are k-de-chromatic in G, and each vertex of G is k-de-chromatically critical for u and v, and each edge of G is not k-de-chromatically critical for u and v. It is easy to verify that if u and v are a strict-vertex critically k-de-chromatic pair of G, then u and v are still k-de-chromatic in $G - e$ for each $e \in E(G)$.

If u and v are a strict-vertex critically k-de-chromatic pair of G we often written as u and v are strict-vertex critically k-de-chromatic in G, or G has a strict-vertex critically k-de-chromatic pair. If G has a strict-vertex critically k-de-chromatic pair $\{u--v\}$, then G is often written as $G_{\{u \overset{k}{--} v\}}$ or $G_{\{u--v\}}$.

Definition 5. *Let w be an arbitrary vertex of a graph H which has two nonempty neighborhoods $V_1, V_2 \subset N_H(w)$ such that $V_1 \cap V_2 = \phi$ and $V_1 \cup V_2 = N_H(w)$. A new graph G is said to be the* strict 2-vertex decomposition graph by decomposing vertex w into u and v *if and only if we replace w by two new vertices u and v and join u to all vertices of V_1 and v to all vertices of V_2, denoted by $H(w_{\{V_1,V_2\}} \to \{u, v\})$, simply $H(w \to \{u, v\})$. Conversely, H is said to be the* strict 2-vertex composition graph by composing u and v into w, *denoted by $G(\{u, v\} \to w)$ (see Example 2).*

Example 2. In Figure 2, Graph (4) is a strict 2-vertex decomposition graph by decomposing vertex w of Graph (3) into u and v. Graph (3), of course, is the strict 2-vertex composition graph by composing u and v of Graph (4) into vertex w. Let H indicate Graph (3) with $V_1 = \{a, d\}$ and $V_2 = \{b, c\}$, and G Graph (4). Then, $G = H(w_{\{V_1,V_2\}} \to \{u, v\})$ and $H = G(\{u, v\} \to w)$.

Definition 6. *Let H and H' be two graphs with $w_1w_2 \in E(H)$ and $h, t \in V(H')$. The new graph $H(w_1w_2 \to H'|_h^t)$, obtained from H and H', is said to be the* edge expansion graph of w_1w_2 replaced by H' on h and t *if and only if we replace w_1w_2 by H' and h replaces w_1 and t replaces w_2. Sometimes, $H(w_1w_2 \to H'|_h^t)$ is simply written as $H(w_1w_2 \to H')$ if no confusion (see Example 3).*

Definition 7. *Let $u_1v_1, u_2v_2, \cdots, u_mv_m$ be m edges of a graph H. Let h_i, t_i be two vertices of a graph H_i, $i = 1, 2, \cdots, m$. Denote $H^0 = H$ and $H^1 = H^0(u_1v_1 \to H_1|_{h_1}^{t_1})$, and $H^i = H^{i-1}(u_iv_i \to H_i|_{h_i}^{t_i}) = H(u_1v_1 \to H_1|_{h_1}^{t_1}, u_2v_2 \to H_2|_{h_2}^{t_2}, \cdots, u_iv_i \to H_i|_{h_i}^{t_i})$, $i = 1, 2, \cdots, m$. H^m is called the* edge expansion graph of $u_1v_1, u_2v_2, \cdots, u_mv_m$ replaced by $H_1, H_2, \cdots, H_m$ on corresponding h_i and t_i for $i = 1, 2, \cdots, m$, *simply denoted by $H(\sum_{i=1}^{m}(u_iv_i \to H_i|_{h_i}^{t_i}))$ (see Example 3).*

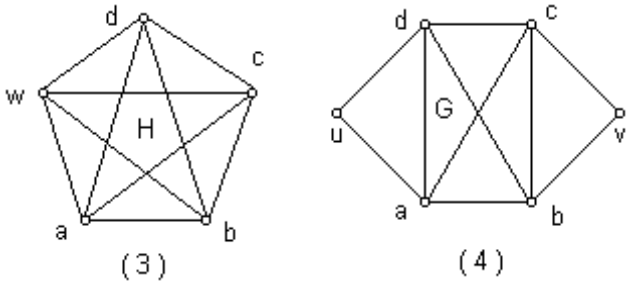


Fig. 2. Decomposition and composition

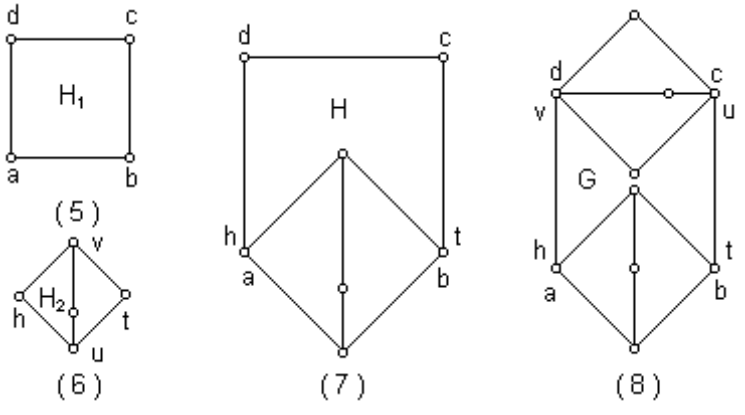


Fig. 3. Edge expansion graphs

Example 3. In Figure 3, Graph (7) is the edge expansion graph of edge ab of Graph (5) replaced by Graph (6) on h and t. Graph (8) is the edge expansion graph of two edges ab and cd of Graph (5) replaced by Graph (6) on h and t, and u and v, that is, the edge expansion graph of cd of Graph (7) replaced by Graph (6) on u and v. Let H_1 be Graph (5), H_2 be Graph (6). Let H be Graph (7), and G be Graph (8). Then, $H = H_1(ab \to H_2|_h^t)$, and $G = H_1(ab \to H_2|_h^t, cd \to H_2|_u^v) = H(cd \to H_2|_u^v)$.

3 Main Results

In this section, we will show how to obtain infinite family of vertex k-critical graphs without critical edges from a given vertex k-critical graph without any critical edge, where $k \geq 4$. If no special statement, we always let $k \geq 4$ in this section.

Theorem 2. *Let G be a vertex k-critical graph without any critical edge, and w be an arbitrary vertex of G. Then, a strict 2-vertex decomposition graph $G(w \to$*

$\{h,t\}$) *of* G *has the strict-vertex critically* $(k-1)$*-de-chromatic pair* $\{h--t\}$ *if and only if* $\mathcal{X}(G(w \to \{h,t\})) \leq k-1$.

Proof. The necessity is obvious. Now, we prove the sufficiency.

Firstly, we prove h and t are $(k-1)$-de-chromatic in $G(w \to \{h,t\})$ when $\mathcal{X}(G(w \to \{h,t\})) \leq k-1$. In fact, if there is a $(k-1)$-coloring P' of $G(w \to \{h,t\})$ such that $C^{P'}(h) = C^{P'}(t)$, then, by composing h and t into one vertex (simply, using one vertex to replace two vertices h and t, and joining it to all neighbors of h and t), P' becomes a $(k-1)$-coloring of G. Contrary to that G is a vertex k-critical graph. This implies that h and t are $(k-1)$-de-chromatic in $G(w \to \{h,t\})$.

Secondly, we proceed to prove each edge of $G(w \to \{h,t\})$ is not critical for h and t. Let e be an arbitrary edge of $G(w \to \{h,t\})$. Then, there is no $(k-1)$-coloring P_1 of $G(w \to \{h,t\}) - e$ such that $C^{P_1}(h) = C^{P_1}(t)$. Otherwise, by composing h and t into one vertex, P_1 becomes a $(k-1)$-coloring P_1 of $G-e$. Contrary to that G is a vertex k-critical graph without any critical edge. This implies that each edge of $G(w \to \{h,t\})$ is not critical for h and t.

Thirdly, we prove each vertex of $G(w \to \{h,t\}) - \{h,t\}$ is critical for h and t. Let u be an arbitrary vertex of $G(w \to \{h,t\}) - \{h,t\}$. Then, there is a $(k-1)$-coloring P_2 of $G-u$, since G is a vertex k-critical graph. Of course, P_2 is also a $(k-1)$-coloring of $G(w \to \{h,t\}) - u = (G-u)(w \to \{h,t\})$ such that $C^{P_2}(h) = C^{P_2}(t) = C^{P_2}(w)$. This shows that u is critical for h and t.

These imply that h and t are a strict-vertex critically $(k-1)$-de-chromatic pair of $G(w \to \{h,t\})$. Thus, the sufficiency is finished. □

Theorem 3. *Let* G *be a vertex* k*-critical graph without any critical edge, and* w *be an arbitrary vertex of* G*. Then, we have at least one strict 2-vertex decomposition graph by decomposing* w *into* h *and* t *which has a strict-vertex critically* $(k-1)$*-de-chromatic pair* $\{h--t\}$.

Proof. Since G is a vertex k-critical graph without any critical edge, there is a $(k-1)$-coloring P of $G-w$. Let C_1 be an arbitrary color, used by P. After we use two vertices h and t to replace w of G, and join h to all vertices $V_1 \subset N_G(w)$ which are not colored by C_1, and join t to all vertices $V_2 \subset N_G(w)$ which are colored by C_1, and paint h with C_1, and paint t with any color which is not C_1, we obtain a strict 2-vertex decomposition graph $G(w_{\{V_1,V_2\}} \to \{h,t\})$. From the construction, it has a $(k-1)$-coloring. From Theorem 2, the theorem holds. □

According to the proof of the above theorem, we may have different strict-vertex critically $(k-1)$-de-chromatic pairs from a given vertex k-critical graph without any critical edge with change of w or color C_1.

Lemma 1. *Let* u *and* v *be vertex critically* $(k-1)$*-de-chromatic in a graph* G*, and* C_1 *and* C_2 *be two arbitrary different painting colors. Then there is a* $(k-1)$*-coloring* P *of* G *such that* $C^P(u) = C_1$ *and* $C^P(v) = C_2$.

Proof. Since u and v are vertex critically $(k-1)$-de-chromatic in G, there is a $(k-1)$-coloring P_1 of G such that $C^{P_1}(u) \neq C^{P_1}(v)$. If $C^{P_1}(u) \neq C_1$, we

construct a $(k-1)$-coloring P_2 of G based on P_1 by letting all vertices of G, which are colored by $C^{P_1}(u)$ under P_1, be painted by C_1, and all vertices of G, which are colored by C_1 under P_1, be painted by $C^{P_1}(u)$. Thus, under the $(k-1)$-coloring P_2 of G, we have $C^{P_2}(u) = C_1$. Similarly, if $C^{P_2}(v) \neq C_2$, then we construct a $(k-1)$-coloring P of G based on P_2 such that $C^P(v) = C_2$ and $C^P(u) = C_1$. □

Lemma 2. *Let u and v be vertex critically $(k-1)$-de-chromatic in a graph G, and w be an arbitrary vertex of G, and C_1 be an arbitrary painting color. Then we have*

(1) if $w \in CriV(u--v)$ (that is, $w \neq u$ and $w \neq v$), then there is a $(k-1)$-coloring P_1 of $G-w$ such that $C^{P_1}(u) = C^{P_1}(v) = C_1$;

(2) if $w \notin CriV(u--v)$ (that is, $w = u$ or $w = v$), then there is a $(k-1)$-coloring P_2 of $G-w$ such that $C^{P_2}(u) = C_1$ when $w = v$, or $C^{P_2}(v) = C_1$ when $w = u$.

Proof. (1) Since u and v are vertex critically $(k-1)$-de-chromatic in G, there is a $(k-1)$-coloring P_1 of $G-w$ such that $C^{P_1}(u) = C^{P_1}(v)$. If $C^{P_1}(u) = C^{P_1}(v) \neq C_1$, then we construct a $(k-1)$-coloring P of $G-w$ based on P_1 by letting all vertices of $G-w$, which are colored by $C^{P_1}(u)$ under P_1, be painted by C_1, and all vertices of $G-w$, which are colored by C_1 under P_1, be painted by $C^{P_1}(u)$. Thus, under the $(k-1)$-coloring P of $G-w$, we have $C^P(u) = C^P(v) = C_1$.

(2) Item 2 is easy to verify. □

Theorem 4. *Let G be a k-critical graph with $|E(G)|$ edges $e_1, e_2, \cdots, e_{|E(G)|}$, and H be another graph which has a strict-vertex critically $(k-1)$-de-chromatic pair $\{h--t\}$. Then, the edge expansion graph $G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t))$ is a vertex k-critical graph without any critical edge.*

Proof. Obviously, $\mathcal{X}(G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t))) = k$. Let vv' be an arbitrary edge of G. In order to prove conveniently, let $H_{\{h_{vv'}--t_{vv'}\}}$ denote the graph H (the subgraph of $G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t)))$ which replaces the edge vv' of G on h and t.

Now, we prove that each edge of $G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t))$ is not k-critical. Let e be an arbitrary edge of $G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t))$. Then, there is an edge ww' of G such that $e \in E(H_{\{h_{ww'}--t_{ww'}\}})$. Since $\{h_{ww'}--t_{ww'}\}$ is a strict-vertex critically $(k-1)$-de-chromatic pair, $h_{ww'}$ and $t_{ww'}$ are also $(k-1)$-de-chromatic in $H_{\{h_{ww'}--t_{ww'}\}} - e$. Thus, $\mathcal{X}(G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t)) - e) = k$ for any $e \in E(G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t)))$. In other words, each edge of $G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t))$ is not critical.

Now, we proceed to prove that each vertex of $G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t))$ is k-critical. Let u be an arbitrary vertex of $G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t))$. It has two cases. One is that there is an edge w_1w_2 of G such that $u \in CriV(\{h_{w_1w_2}--t_{w_1w_2}\})$. Another is that $u = w_1$ or $u = w_2$, that is, $u \notin CriV(\{h_{w_1w_2}--t_{w_1w_2}\})$ or belongs to $V(G)$.

For the first case, since G is k-critical, there is $(k-1)$-coloring P_1 of $G-w_1w_2$ such that $C^{P_1}(w_1) = C^{P_1}(w_2)$. According to Item (1) of Lemma 2, there is

a $(k-1)$-coloring $P_{w_1w_2}$ of $H_{\{h_{w_1w_2}--t_{w_1w_2}\}} - u$ such that $C^{P_{w_1w_2}}(h_{w_1w_2}) = C^{P_{w_1w_2}}(t_{w_1w_2}) = C^{P_1}(w_1) = C^{P_1}(w_2)$. Let ww' be an arbitrary edge of $G - w_1w_2$. Then, there is a $(k-1)$-coloring $P_{ww'}$ of $H_{\{h_{ww'}--t_{ww'}\}}$ such that $C^{P_{ww'}}(h_{ww'}) = C^{P_1}(w)$ and $C^{P_{ww'}}(t_{ww'}) = C^{P_1}(w')$ according to Lemma 1. Combining these $|E(G)| - 1$ $(k-1)$-colorings and P_1, we obtain a $(k-1)$-coloring of $G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t)) - u$.

For the second case, there is a $(k-1)$-coloring P_2 of $G - u$, since G is k-critical. Let w_3w_4 be an arbitrary edge of $G - u$. Then, there is a $(k-1)$-coloring $P_{w_3w_4}$ of $H_{\{h_{w_3w_4}--t_{w_3w_4}\}}$ such that $C^{P_{w_3w_4}}(h_{w_3w_4}) = C^{P_2}(w_3)$ and $C^{P_{w_3w_4}}(t_{w_3w_4}) = C^{P_2}(w_4)$ according to Lemma 1. The number of such $(k-1)$-colorings is $|E(G)| - d_G(u)$ where $d_G(u)$ indicates the degree of u in G. Let w_5w_6 be an arbitrary edge of G whose one endpoint is u, and another is not. We may assume w_5 is u. Then, there is a $(k-1)$-coloring $P_{w_5w_6}$ of $H_{\{h_{w_5w_6}--t_{w_5w_6}\}} - u = H_{\{h_{uw_6}--t_{uw_6}\}} - u$ such that $C^{P_{w_5w_6}}(t_{w_5w_6}) = C^{P_{w_5w_6}}(t_{uw_6}) = C^{P_2}(w_6)$, according to Item (2) of Lemma 2. The number of such $(k-1)$-colorings is $d_G(u)$. Combining all $|E(G)| = (|E(G)| - d_G(u)) + d_G(u)$ $(k-1)$-colorings, we obtain a $(k-1)$-coloring of $G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t)) - u$.

These imply $\mathcal{X}(G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t)) - u) \leq k-1$ for any $u \in V(G(\sum_{i=1}^{|E(G)|}(e_i \to H|_h^t)))$. Thus, the theorem holds. □

Theorem 5. *Let G be a k-critical graph with $|E(G)|$ edges $e_1, e_2, \cdots, e_{|E(G)|}$. Let $\{h_i - -t_i\}$ be a strict-vertex critically $(k-1)$-de-chromatic pair of graph H_i for $i = 1, 2, \cdots, |E(G)|$. Then, the edge expansion graph $G(\sum_{i=1}^{|E(G)|}(e_i \to H_i|_{h_i}^{t_i}))$ is a vertex k-critical graph without any critical edge.*

Proof. The proof is similar to that of Theorem 4. □

Let G be a general vertex k-critical graph. Then, some edges of G are critical, and others not. Letting $CriE(G)$ denote the edge set of all critical edges of G, we have the following result.

Theorem 6. *Let G be a vertex k-critical graph with $|CriE(G)|$ critical edges $e_1, e_2, \cdots, e_{|CriE(G)|}$. Let $\{h_i - -t_i\}$ be a strict-vertex critically $(k-1)$-de-chromatic pair of graph H_i for $i = 1, 2, \cdots, |CriE(G)|$. Then, the edge expansion graph $G(\sum_{i=1}^{|CriE(G)|}(e_i \to H_i|_{h_i}^{t_i}))$ is a vertex k-critical graph without any critical edge.*

Proof. The proof is similar to that of Theorem 4. □

Corollary 1. *Any vertex k-critical graph without any critical edge constructs, by itself, infinite family of vertex k-critical graphs without critical edges.*

Proof. It follows from Theorem 3 and Theorem 6. □

This result is not empty, since Lattanzio [6] presented a vertex k-critical graph without any critical edge for any k (≥ 6 and not prime).

Corollary 2. *Let $H_1, H_2, \cdots, H_m$ be m vertex k-critical graphs without critical edges. Then, they generates infinite family $\mathcal{V}$ of vertex k-critical graphs without critical edges such that each member of $\mathcal{V}$ is relative to all given vertex k-critical graphs without critical edges.*

Proof. Let $\mathcal{G}$ be an infinite set of vertex k-critical graphs such that each graph has more than m critical edges. From Theorem 3, we obtain at least one strict 2-vertex decomposition graph $H_i(w_i \to \{h_i, t_i\})$ which has a strict-vertex critically $(k-1)$-de-chromatic pair $\{h_i - -t_i\}$ for $i = 1, 2, \cdots, m$. Then, It follows from Theorem 6. □

When $k = 4$, Dirac Conjecture becomes very difficult. It appears in open problems in several papers such as [3] [6]. If there is a graph which has a strict-vertex critically 3-de-chromatic pair, then there are infinite vertex 4-critical graphs without critical edges. Here, we present the following equivalent open problem.

Open Problem: Is there a graph which has a strict-vertex critically 3-de-chromatic pair?

4 Example

Circulant graph $C(17; 2, 6, 7, 8)$, denoted by $G_{C_{17}}$, is a vertex 5-critical graph without any critical edge (see Graph (9) in Figure 4). This can be easily verified by computer. Thus, the strict 2-vertex decomposition graph $G_{C_{17}}(w \to \{u, v\})$

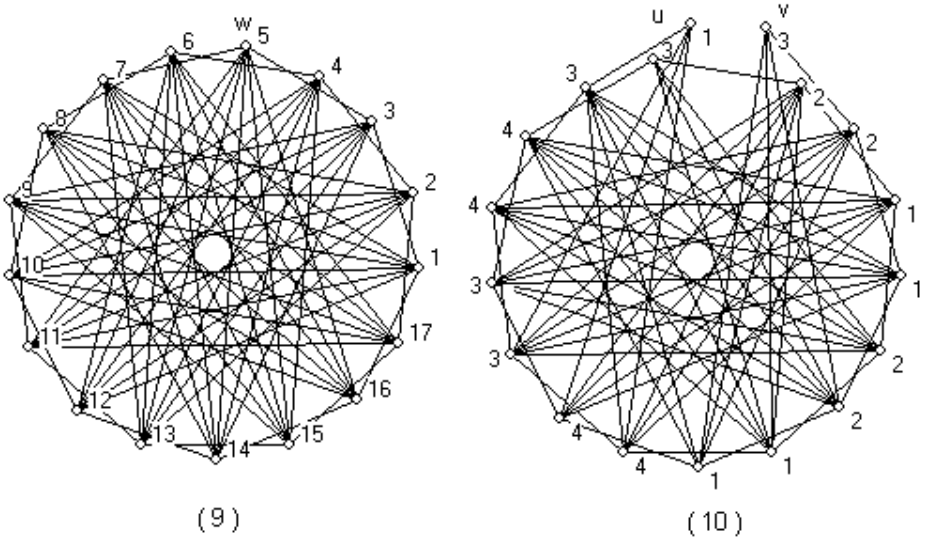


Fig. 4. A strict-vertex critically 4-de-chromatic pair from decomposition

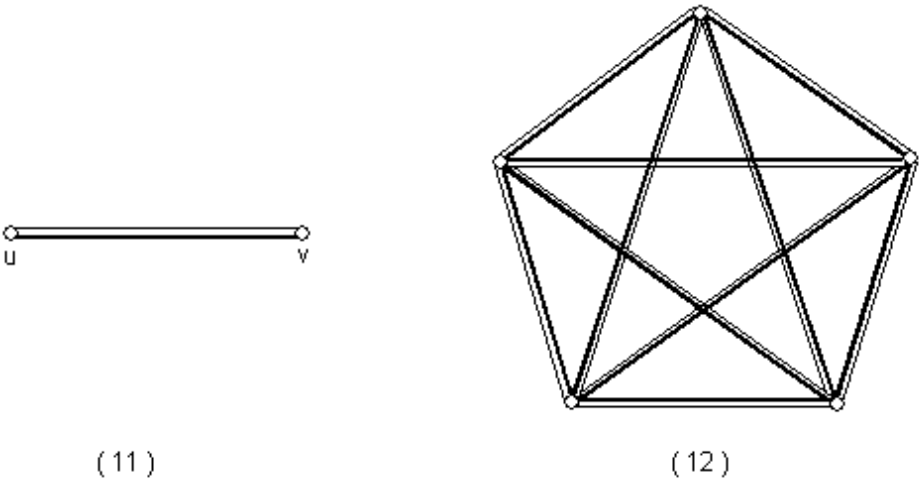


Fig. 5. A new vertex 5-critical graph without any critical edge

(see Graph (10) in Figure 4) has a strict-vertex critically 4-de-chromatic pair $\{u - -v\}$, according to Theorem 2.

Doubtlessly, the most important two vertices of Graph (10) are u and v so that in order to draw conveniently, we may simply draw Graph (10) as Graph (11). Then, when Graph (11) replaces all edges of Graph (3) (a famous 5-critical graph K_5), we obtain a new vertex 5-critical graph without any critical edge (see Graph (12) in Figure 5). Thus, we have, if there is a 5-critical graph H, then there is a vertex 5-critical graph without any critical edge, by Graph (11) replacing all edges of H. As we know, 5-critical graphs are infinite. Let $\mathcal{G}$ be an arbitrary infinite set of 5-critical graphs. Thus, when Graph (11) replaces all edges of each graph of $\mathcal{G}$, we obtain an infinite family of vertex 5-critical graphs without critical edges.

5 Conclusion

This paper shows that, any vertex k-critical graph without any critical edge has its own 'DNA': strict-vertex critically $(k-1)$-de-chromatic pair, so that, by such 'DNA', it easily generates new vertex k-critical graphs without critical edges.

Acknowledgment

This research has been supported by the nature science foundation of Yunnan University, under Grant No. 2008YB026.

References

1. Brown, J.I.: A Vertex Critical Graph without Critical Edges. Discrete Math. 102, 99–101 (1992)
2. Erdős, P.: On Some Aspects of My Work with Gabriel Dirac. In: Andersen, L.D., Jakobsen, I.T., Thomassen, C., Toft, B., Vestergaard, P.D. (eds.) Graph Thoery in Memory of G.A. Dirac, Annals of Discrete Mathmatics, vol. 41, pp. 111–116. North-Holland, Amsterdam (1989)
3. Jensen, T.R.: Dense Critical and Vertex-Critical Graphs. Discrete Math. 258, 63–84 (2002)
4. Jensen, T.R., Royle, G.F.: Small Graphs of Chromatic Number 5: A Computer Search. J. Graph Theory 19, 107–116 (1995)
5. Jensen, T.R., Toft, B.: Graph Coloring Problems. Wiley, New York (1995)
6. Lattanzio, J.J.: A Note on a Conjecture of Dirac. Discrete Math. 258, 323–330 (2002)

A Note on Edge Choosability and Degeneracy of Planar Graphs

Baoyindureng Wu and Xinhui An

College of Mathematics and System Sciences, Xinjiang University
Urumqi, Xinjiang 830046, China
baoyin@xju.edu.cn

Abstract. Let H be the graph obtained from a 6-cycle C_6 by adding an edge which joins a pair of two vertices with distance two. We show that if a planar graph does not contain H, then G is edge-t-choosable, where $t = 7$ if $\Delta(G) = 5$, and $t = \Delta(G) + 1$, otherwise. This extends the known results that a planar graph is edge-$(\Delta(G) + 1)$-choosable when $\Delta(G) \neq 7$ and G does not contain a k-cycle for some $k \in \{3, 5, 6\}$. It is well-known that $\{3, 5, 6\}$ are only integers for which the lack of a cycle of length in $\{3, 5, 6\}$ for a planar graph G implies 3-degeneracy of G. As a by-product, we prove that if a planar graph G contains at most seven 3-cycles, G is 3-degenerate. We also answer a problem of Raspaud and Wang (European J. Combin. 29(2008) 1064-1075) in negative.

Keywords: List coloring, Planar graphs.

1 Introduction

An edge-k-coloring of G is a mapping ϕ from $E(G)$ to the set of colors $\{1, 2, \cdots, k\}$ such that $\phi(e) \neq \phi(e')$ for any adjacent edges e and e' of G. The chromatic index $\chi'(G)$ is the smallest integer k for which G has an edge-k-coloring. The mapping L is said to be a list assignment of edges of G if it assigns a list $L(e)$ of possible colors to each edge e of G. If G has a proper edge coloring ϕ such that $\phi(e) \in L(e)$ for all edges e, then we say that G is edge-L-colorable or ϕ is an edge-L-coloring of G. We call G edge-k-choosable if it is edge-L-colorable for every list assignment L satisfying $|L(e)| \geq k$ for all edges e. The list chromatic index $\chi_l{}'(G)$ of G is the smallest k for which G is edge-k-choosable.

Throughout the paper, H denotes the graph obtained from a 6-cycle C_6 by adding an edge which joins a pair of two vertices with distance two. It is easy to see that H is the minimum supergraph of C_3, C_5 and C_6. The main result of this paper is to show that the following conjecture holds for a planar graph G without H as a subgraph and $\Delta(G) \neq 5$. This improves some earlier results in [11,12,14].

Conjecture 1.1. Every simple graph G is edge-$(\Delta(G) + 1)$-choosable.

The above conjecture was first proposed by Vizing (see [8]). An earlier result of Harris [6] asserts that, if G is a graph with $\Delta(G) \geq 3$, then $\chi_l{}'(G) \leq 2\Delta(G) - 2$.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 249–257, 2009.

This implies that Conjecture 1.1 holds if $\Delta(G) = 3$. Juvan, Mohar, and Škrekovski [7] settled the case when $\Delta(G) = 4$. Conjecture 1.1 was confirmed for other special cases such as complete graphs [5], and planar graphs with $\Delta(G) \geq 9$ [1]. Chen et al. [2] recently proved Conjecture 1.1 is true for planar graphs without $C_5 + e$, where $C_5 + e$ denotes the graph obtained from the five cycle C_5 by adding a chord.

At the end of this paper, we will prove that if a planar graph G contains at most seven 3-cycles, G is 3-degenerate. We also answer a problem of Raspaud and Wang [10] in negative.

2 Main Results

For a plane graph $G = (V(G), E(G))$, $F(G)$ denotes the set of faces of G. The well-known Euler's theorem states that $|V(G)| - |E(G)| + |F(G)| = 2$ if G is connected. For a vertex v, the *degree* of a vertex v, denoted by $d(v)$, is the number of edges incident with v in G. The minimum degree and the maximum degree of G are denoted $\delta(G)$ and $\Delta(G)$, respectively. For a face $f \in F(G)$, the degree of f, denoted by $d(f)$, is the number of edges incident with f, in which each cut edge is counted twice. For the sake of simplicity, we call a vertex v a k-vertex if $d(v) = k$. Similarly, a face f is called a k-face if $d(f) = k$. For an element $x \in V(G) \cup F(G)$ with $d(x) \geq k$, x is called a k^+-vertex if $x \in V(G)$, and a k^+-face, otherwise.

Lemma 1. *Let G be a planar graph without H as a subgraph. If $d(u) + d(v) \geq \Delta(G) + 3$ for any edge uv of G, then one of the following holds:*

(1) *G contains an edge uv such that $d(u) + d(v) \leq 8$.*
(2) *$\Delta(G) = 6$ and G contains a 4-cycle $C = v_1v_2v_3v_4v_1$ such that $d(v_1) = d(v_3) = 3$ and $d(v_2) = d(v_4) = 6$.*

Proof. Suppose that the lemma is false and let G be a counterexample with minimum order. Then G is connected. Since $d(u) + d(v) \geq \Delta(G) + 3$ for any edge uv of G, $\delta(G) \geq 3$. Assume that G is embedded on the plane such a way that the outer face of G is 6^+-face if G is not 2-connected (This can be done because the length of boundary of each end block is at least three). Hence the boundary of each k-faces are cycles when $k \leq 5$. In the rest of proof, we shall apply the well-known discharging method.

Initial weight. We assign a *weight* to the vertices and faces of G by taking $w(x) = 2d(x) - 6$ if $x \in V(G)$ and $w(x) = d(x) - 6$ if $x \in F(G)$. Euler's formula on a plane graph and the hand shaking lemma's for the degree of vertices and faces imply

$$\sum_{x \in V(G) \cup F(G)} w(x) = -12.$$

This shows that the total weight of vertices and faces of G is negative.

To obtain a contradiction we redistribute the weight of vertices and faces by applying Rule 1 to Rule 4 described later. A cluster of triangles is a subgraph

of G which consists of a non-empty minimal set of 3-faces such that no other 3-face is adjacent to a member of this set. Since G is not contains a subgraph isomorphic to H, there are four possible clusters of triangles in G as illustrated in Figure 1, in which all triangles have a common vertex.

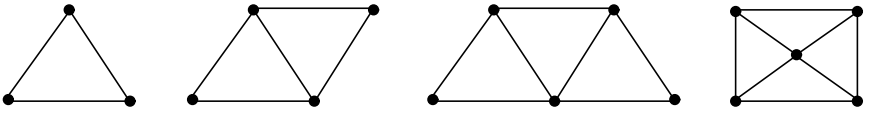


Fig. 1. The possible clusters of triangles in G

The number of triangles in a cluster of triangles is called the size of the cluster. We say that a face f is adjacent to a cluster $\mathcal{C}$ of triangles if f is adjacent to a face in $\mathcal{C}$.

Claim 1. For a vertex $v \in V(G)$ with $d(v) \geq 5$, there are at most $\lfloor \frac{3}{4}d(v) \rfloor$ triangles containing v.

Proof of Claim 1. This is an immediate consequence of the fact that the size of a cluster of triangles with a vertex v as their common vertex is at most three if $d(v) \geq 5$. □

A 4-face is adjacent to a a cluster of triangles of size two or three and a 5-face is adjacent to a triangle or a cluster of size three in Figure 2, where the vertices shown with black square is identified.

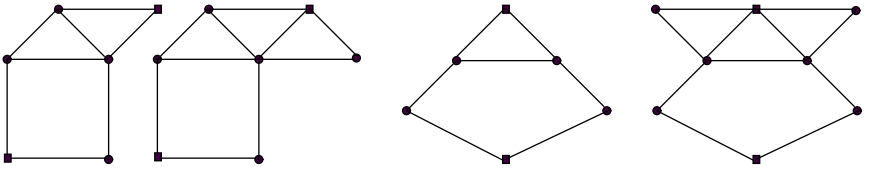


Fig. 2. Clusters of triangles and adjacent small faces

Claim 2. Let $v \in V(G)$ and $d(v) \geq 5$. Suppose v is incident to a cluster $\mathcal{C}$ of triangles of size two or three. If f_1 and f_2 are the two faces adjacent to $\mathcal{C}$ and are incident to v, then one of f_1 and f_2 is a 6^+-face.

Proof of Claim 2. It is not hard to see from Figure 2 that, if each of f_1 and f_2 is 4-face or 5-face, then G cannot be planar or contains a subgraph isomorphic to H. □

Claim 3. Let f_1, f_2 and f_3 be three faces incident to v. If both f_1 and f_3 are 3-faces are adjacent to f_2, then f_2 is neither a 4-face nor a 5-face.

Proof of Claim 3. Let $k = d(v)$. Let v_k, v_1, v_2, v_3 be the four neighbors of v that appears successively in clockwise order, and $V(f_1) = \{v, v_k, v_1\}$, $V(f_3) = \{v, v_2, v_3\}$, where $k = d(v)$. Suppose f_2 is a 4-face. Since f_2 is adjacent to both f_1 and f_2, $V(f) = \{v, v_1, v', v_2\}$, where $v' \in V(G)$ and $v' \notin \{v, v_k, v_1, v_2, v_3\}$. Clearly, $G[\{v, v_k, v_1, v', v_2, v_3\}]$ contains H as its subgraph, a contradiction. Similarly, we can see that if f_2 is a 5-face, $G[V(f_i) \cup V(f_2)]$ contains a subgraph isomorphic to H for some $i \in \{1, 3\}$. □

Claim 4. Let $\mathcal{C}$ be a cluster of triangle of size three, and v be the common vertex of those triangle. If f is a 5-face incident to v, then either f is not adjacent to $\mathcal{C}$, or f is adjacent to $\mathcal{C}$ in the way as shown in Figure 2.

Proof of Claim 4. It is obvious, so omitted.

Let v be vertex with $d(v) \geq 4$ and $f \in F_3(v) \cup F_4(v) \cup F_5(v)$. Let $z(v, f)$ be the amount of weight transferred from v to f. Now we redistribute the weight of G according to the following rules.

Rule 1. If $d(v) \geq 7$,

$$z(v,f) = \begin{cases} \frac{3}{2} & \text{if f} \in \text{F}_3\text{(v)}; \\ 1 & \text{if f} \in \text{F}_4\text{(v)}; \\ \frac{1}{2} & \text{if f} \in \text{F}_5\text{(v)}. \end{cases}$$

Rule 2. If $d(v) = 6$,

$$z(v,f) = \begin{cases} \frac{3}{2} & \text{if f} \in \text{F}_3\text{(v)}; \\ \frac{6-\frac{3}{2}\times|F_3(v)|}{|F_4(v)\cup F_5(v)|} & \text{if f} \in \text{F}_4\text{(v)} \cup \text{F}_5\text{(v)}. \end{cases}$$

Rule 3. Let $d(v) = 5$. If v is incident to a cluster of triangles of size three and $|F_4(v)| \neq 0$, then

$$z(v,f) = \begin{cases} \frac{1}{2} & \text{if f} \in \text{F}_4\text{(v)}; \\ \frac{7}{6} & \text{if f} \in \text{F}_3\text{(v)}, \end{cases}$$

otherwise,

$$z(v,f) = \begin{cases} \frac{4}{3} & \text{if f} \in \text{F}_3\text{(v)}; \\ \frac{4-\frac{4}{3}\times|F_3(v)|}{|F_4(v)\cup F_5(v)|} & \text{if f} \in \text{F}_4\text{(v)} \cup \text{F}_5\text{(v)}. \end{cases}$$

Rule 4. If $d(v) = 4$, $z(v, f) = \frac{1}{2}$ for any $f \in F_3(v) \cup F_4(v) \cup F_5(v)$.

Let $w'(x)$ be the new weight after the transfer is complete. We will check that $w'(x) \geq 0$ for any $x \in V(G) \cup F(G)$. Actually $w'(x) = w(x)$ if $x \in V(G)$ and $d(x) = 3$, or $x \in F(G)$ and $d(x) \geq 6$, then $w'(x) = w(x) \geq 0$. By Rule 4, if $x \in V(G)$ and $d(x) = 4$, then $w'(x) \geq w(x) - \frac{1}{2} \times 4 = 2 - 2 = 0$.

Now let $v \in V(G)$ with $d(v) \geq 5$. By Claim 1, since $|F_3(v)| \leq \lfloor \frac{3}{4}d(v) \rfloor$, we have $\frac{3}{2}|F_3(v)| + (d(v) - |F_3(v)|) \leq 2d(v) - 6$ for $d(v) \geq 9$. Thus, if $d(v) \geq 9$, $w'(v) = w(v) - \frac{3}{2}|F_3(v)| - (d(v) - |F_3(v)|) \geq 0$.

If $d(v) = 8$, then $|F_3(v)| \leq 6$. If $|F_3(v)| = 6$, by Claim 3, $|F_4(v) \cup F_5(v)| = 0$, and thus $w'(v) = w(v) - \frac{3}{2} \times 6 = (2 \times 8 - 6) - 9 = 1$. If $|F_3(v)| \leq 4$, then $w'(v) \geq w(v) - (\frac{3}{2} \times 4 + 4) = 0$. If $|F_3(v)| = 5$, v is incident to a clusters of triangles of size two or three. By Claim 2, there is a 6^+-face incident to v. So, $w'(v) \geq w(v) - (\frac{3}{2} \times 5 + 2) = 10 - 9.5 = 0.5$.

If $d(v) = 7$, then $|F_3(v)| \leq \lfloor \frac{3}{4}d(v) \rfloor = \lfloor \frac{3}{4} \times 7 \rfloor = 5$. If $|F_3(v)| \leq 2$, then $w'(v) \geq w(v) - (\frac{3}{2} \times 2 + 5) = 8 - 8 = 0$. If $|F_3(v)| = 5$, v is incident to exactly a clusters of triangles of size two and a cluster of triangles of size three. By Claim 3, the remaining two faces incident to v are 6^+-faces. Hence, $w'(v) = w(v) - (\frac{3}{2} \times 5) = 8 - 7.5 = 0.5$. If $|F_3(v)| = 3$ or $|F_3(v)| = 4$, there must be a 6^+-face incident to v by Claim 2 and Claim 3. Thus, $w'(v) \geq w(v) - (\frac{3}{2} \times 4 + 2) = 0$.

Suppose $d(v) = 6$. By Claim 1, $|F_3(v)| \leq 4$. If $|F_3(v)| = 4$, then the remaining two faces incident to v must be 6^+-faces by Claim 2 and Claim 3, thus $w'(v) = w(v) - (\frac{3}{2} \times 4) = 6 - 6 = 0$. If $|F_3(v)| \leq 3$, it is clear that $w'(v) \geq 0$ from Rule 2.

If $d(v) = 5$, $|F_3(v)| \leq 3$. If $|F_3(v)| \neq 3$ or $|F_4(v)| = 0$, then $w'(v) \geq 0$ by Rule 3. If $|F_3(v)| = 3$ and $|F_4(v)| \neq 0$, then by Claim 2, $|F_4(v)| = 1$, and thus $w'(v) = w(v) - \frac{1}{2} - \frac{7}{6} \times 3 = 0$.

Now let f be a face with $d(f) \leq 5$. We first consider the case $d(f) = 3$. For convenience, let $V(f) = \{v_1, v_2, v_3\}$, where $d(v_1) \leq d(v_2) \leq d(v_3)$. If $d(v_3) \geq d(v_2) \geq 6$, then by Rules 1 and 2, $w'(v) \geq w(v) + \frac{3}{2} \times 2 = 0$. If $d(v_1) = 3$, then $d(v_2) \geq 6$ and $d(v_3) \geq 6$ by the assumption that $d(u) + d(v) \geq max\{9, \Delta(G) + 3\}$ for any edge uv of G. So, we assume that $d(v_1) \geq 4$. Clearly $d(v_3) \geq d(v_2) \geq 5$. If $d(v_3) \geq 6$ then $w'(f) \geq w(f) + \frac{1}{2} + \frac{7}{6} + \frac{3}{2} > 0$ by Rules 1-4. So, it remains to consider the case when $d(v_2) = d(v_3) = 5$. We claim that $z(v_i, f) \geq \frac{4}{3}$ for some $i \in \{2, 3\}$. If this does not hold, then $z(v_2, f) = z(z_3, f) = \frac{7}{6}$, by Rule 3 and Claim 2, each of v_2 and v_3 is incident to a cluster of triangles of size three, and $|F_4(v_i)| = 1$ for $i \in \{2, 3\}$. But this implies that $d(v_1) = 3$, which contradicts $d(v_1) \geq 4$. Thus, we have $w'(v) \geq w(v) + \frac{1}{2} + \frac{7}{6} + \frac{4}{3} = 0$.

If $d(f) = 5$, there are three vertices of $V(f)$ with degree at least four. Let $v \in V(f)$ with $d(v) \geq 4$. We claim that $z(v, f) \geq \frac{1}{2}$. It is trivially true when $d(v) = 4$ or $d(v) \geq 7$, by Rule 1 and Rule 4. Now suppose $d(v) = 5$. Since v is incident to a 5-face (i.e. f), by Claim 4, $|F_3(v)| \leq 2$. Hence, if $|F_3(v)| \leq 1$, then

$$z(v, f) \geq \frac{4 - \frac{4}{3}}{4} = \frac{2}{3};$$

if $|F_3(v)| = 2$, then by Claim 2 and Claim 3, v is incident to a 6^+-face, and thus by Rule 3,

$$z(v, f) \geq \frac{4 - \frac{4}{3} \times |F_3(v)|}{|F_4(v) \cup F_5(v)|} \geq \frac{4 - \frac{4}{3} \times 2}{2} = \frac{2}{3}.$$

If $d(v) = 6$, by Claim 2 and Claim 3, $|F_3(v)| \leq 3$. By Rule 2,

$$z(v, f) = \frac{6 - \frac{3}{2}|F_3(v)|}{|F_4(v) \cup F_5(v)|} \geq \frac{1.5}{3} = \frac{1}{2}.$$

Summing up the above, $z(v,f) \geq \frac{1}{2}$ when $d(v) \geq 4$. So we have

$$w'(f) \geq w(f) + \frac{1}{2} \times 3 = -1 + \frac{3}{2} = \frac{1}{2} \geq 0.$$

Finally let us consider the case when $d(f) = 4$. If all vertices of $V(f)$ are 4^+-vertices, by the fact that $z(v,f) \geq \frac{1}{2}$ for any $v \in V(f)$ as we have seen in the previous paragraph, we have $w'(f) \geq w(f) + \frac{1}{2} \times 4 = -2 + 2 = 0$. Now assume that v_1, v_2, v_3, v_4 be the four vertices which appear on the boundary of f in clockwise, and $d(v_1) = 3$. Then $d(v_2) \geq max\{\Delta(G), 6\}$ and $d(v_4) \geq max\{\Delta(G), 6\}$. If $\Delta(G) \geq 7$, by Rule 1, $z(v_i, f) = 1$ for $i \in \{2, 4\}$, hence $w'(f) \geq w(f) + z(v_2, f) + z(v_4, f) = -2 + 2 = 0$. So, $\Delta(G) = 6$. Then $d(v_2) = d(v_4) = 6$. Since both v_2 and v_4 are adjacent to a 4-face (i.e. f), $|F_3(v_i)| \leq 3$ for each $i \in \{2, 4\}$ by Claim 3. If $|F_3(v_i)| = 3$, v_i is incident to a 6^+-face for $i \in \{2, 4\}$, hence by Rule 2,

$$z(v_i, f) = \frac{6 - \frac{3}{2} \times |F_3(v)|}{|F_4(v) \cup F_5(v)|} \geq \frac{1.5}{2} = \frac{3}{4};$$

if $|F_3(v_i)| \leq 2$,

$$z(v_i, f) = \frac{6 - \frac{3}{2} \times |F_3(v)|}{|F_4(v) \cup F_5(v)|} \geq \frac{3}{4}.$$

Moreover, by the the choice of G, $d(v_3) \geq 4$. Thus $z(v_3, f) \geq \frac{1}{2}$. So,

$$w'(f) = w(f) + z(v_2, f) + z(v_3, f) + z(v_4, f) \geq -2 + \frac{3}{4} \times 2 + \frac{1}{2} = 0.$$

Therefore,

$$-12 = \sum_{x \in V(G) \cup F(G)} w(x) = \sum_{x \in V(G) \cup F(G)} w'(x) \geq 0,$$

a contradiction. The result follows. □

Theorem 1. *If a planar graph G does not contain a subgraph isomorphic to H, then G is edge-t-choosable, where $t = 7$ if $\Delta(G) = 5$, and $t = \Delta(G) + 1$, otherwise.*

Proof. The proof is by induction on the number of edges of G. Let L be a list assignment of edges of G with $|L(e)| \geq t$, where $t = 7$ if $\Delta(G) = 5$, and $t = \Delta(G) + 1$ otherwise. If $\Delta(G) \leq 4$, then by the result of [7], G is edge $(\Delta(G)+1)$-choosable. So we suppose $\Delta(G) \geq 5$ and $|E(G)| \geq 5$. First assume G contains an edge $e^* = uv$ such that $d(u) + d(v) \leq \Delta(G) + 2$. It means that there are at most $\Delta(G)$ edges of G adjacent to e^* in G. By the induction hypothesis, $G - e^*$ has an L-coloring φ. Since $|L(e^*)| = \Delta(G) + 1$, there must be a color not used in φ on those edges adjacent to e^*. Thus we obtain an L-coloring of G.

Now we suppose $d(x) + d(y) \geq \Delta(G) + 3$ for any edge $e = xy$. If $\Delta(G) = 5$, by Lemma 1, G contains an edge $e' = uv$ such that $d(u) + d(v) \leq 8$. Then

$G - e'$ has an L'-coloring ϕ', where L' is the restriction of L on $E(G - e)$. Since $d(u) + d(v) \leq 8$, again there will be a color which is available for e, which is not assigned by ϕ' to those edges adjacent to e' . This proves that G is edge-7-choosable. If $\Delta(G) \geq 6$, then $d(x) + d(y) \geq 9$ for any edge $e = xy$. By Lemma 1, $\Delta(G) = 6$ and G contains a 4-cycle C with two 3-vertices and two 6-vertices. By induction hypothesis, $G - E(C)$ has an L-coloring φ for the list assignment L restricted on $E(G) \setminus E(C)$. Let $L'(e)$ be the set of colors not used in φ on those edges adjacent to e in G. It is obvious that $|L'(e)| \geq 2$ for any $e \in E(C)$. Since C is a 4-cycle, it is edge-L'-colorable. This completes the proof. □

3 Some Related Results on Degeneracy of Planar Graphs

Theorem 2. *If a planar graph G does not contain a subgraph isomorphic to H, then $\delta(G) \leq 4$.*

Proof. Suppose the theorem is false and let G be a counterexample. Then $\delta(G) \geq 4$ and we may assume G is a connected plane graph. We assign a weight function for the elements of $V(G) \cup F(G)$ by $w(x) = d(x) - 6$ if $x \in V(G)$ and $w(x) = 2d(x) - 6$ if $x \in F(G)$. Then by Euler's formula and the hand shaking lemma,

$$\sum_{v \in V(G)} (d(v) - 6) + \sum_{f \in F(G)} (2d(f) - 6) = -12.$$

So, there must be some 5-vertices in G, which are only elements with negative weight. We redistribute the weight in the following rules.

Rule: every 4^+-faces give $\frac{w(f)}{d(f)}$ for its each incident 5-vertices.

Let $w'(x)$ be the new weight after the transfer of weight is finished. Since G does not contain H, there are at least two 4^+-faces incident to a 5-vertex v. Thus

$$w'(v) \geq w(v) + 2 \times \frac{w(f)}{d(f)} = -1 + 2 \times \frac{2d(f) - 6}{d(f)} \geq -1 + 2 \times \frac{1}{2} = 0.$$

The theorem is proved. □

A graph is said to be a k-degenerate if every induced subgraph contains a vertex of degree at most k. It is an easy consequence of Euler's formula on planar graphs that every planar graph without 3-cycles contains a vertex of degree at most 3. Wang and Lih [12], independently Lam et al. [9] proved that planar graph without 5-cycles are 3-degenerate. Fijavz et al. [4] proved that planar graph without 6-cycles are 3-degenerate. Note that icosidodecahedorm (i.e., the line graph of dodecahedron) is not 3-degenerate [9]. Hence the lack of 4-cycles cannot guarantee the 3-degeneracy of G. For each $k \geq 7$, Choudum [3] constructed 4-regular planar graphs without k-cycles. Since H is a minimum supergraph of C_3, C_5 and C_6, it is natural to ask that is every planar graph without H 3-degenerate ? One can easily find that the dual of graphs G_m constructed in the

following theorem are 4-regular and do not contain H. Moreover, the theorem answers a question posed by Raspaud and Wang [10] in negative, which states that is there a constant c such that every planar graph G without triangles at distance less than c has $\delta(G) \leq 3$?

Theorem 3. *For every positive integer n there is a 4-regular planar graph in which every two triangles are at a distance greater than n.*

Proof. Let G_m denote the planar graph obtained from the three-dimensional cube by tiling each of its six faces with an $m \times m$ square grid. More rigorously, the vertices of G_m are all vectors (x_1, x_2, x_3) such that each of x_1, x_2, x_3 is one of the integers $0, 1, \ldots m$ and at least one of of x_1, x_2, x_3 is 0 or m; two such vertices are adjacent if and only if their Euclidean distance is 1. Every face of G_m is a 4-face; G_m has precisely eight vertices of degree three (these are the eight corners of the cube) and every two of them are at a distance m or more. The dual of G_m is 4-regular; G_m has precisely eight triangles and every two of them are at a distance $m-1$ or more. □

Note that there are exactly eight 3-cycles in the dual of G_m constructed in Theorem 3.

Theorem 4. *Let G be a planar graph with at most seven 3-cycles. Then G is 3-degenerate.*

Proof. Suppose the theorem is false, and G is a minimal counterexample. Then $\delta(G) \geq 4$ and G is connected. We assume G is embedded on the plane. Rewrite Euler's formula as

$$\sum_{v \in V(G)} (d(v) - 4) + \sum_{f \in F(G)} (d(f) - 4) = -8.$$

Then $0 - |F_3| \leq -8$, and thus $|F_3| \geq 8$, where F_3 is the set of 3-faces in G. Since for a connected plane graph, the number of 3-cycles is not less than that of 3-faces unless $G \cong C_3$, the result follows. □

Acknowledgments. The authors are grateful to referee's for their helpful comments. This work was done when the first author was visiting Concordia Computational Combinatorial Optimization Laboratory. The authors are greatly indebted to Professor Vasek Chvatal, Neil Olver of McGill University for their help to formulate Theorem 3. The financial support from China Scholarship Council is also appreciated.

References

1. Borodin, O.V.: A Generalization of Kotzig's Theorem and Prescribed Edge Coloring of Planar Graphs. Mat. Zametki 48, 22–28 (1990)
2. Chen, Y., Zhu, W., Wang, W.: Edge Choosability of Planar Graphs without 5-Cycles with a Chord. Discrete Math. (to appear)

3. Choudum, S.A.: Some 4-Valent, 3-Connected, Planar, Almost Pancyclic Graphs. Discrete Math. 18, 125–129 (1977)
4. Fijavz, G., Juvan, M., Mohar, B., Skrekovski, R.: Planar Graphs without Cycles of Specific Lengths. Europ. J. Combin. 23, 377–388 (2002)
5. Häggkvist, R., Janssen, J.: New Bounds on the List-Chromatic Index of the Complete Graph and other Simple Graphs. Combin. Probab. Comput. 6(3), 295–313 (1997)
6. Harris, A.J.: Problems and Conjectures in Extremal Graph Theory. Ph.D. dissertation, Cambridge University, UK (1984)
7. Juvan, M., Mohar, B., Skrekovski, R.: Graphs of Degree 4 are 5-Edge-Choosable. J. Graph Theory 32, 250–262 (1999)
8. Kostochka, A.V.: List Edge Chromatic Number of Graphs with Large Girth. Discrete Math. 101, 189–201 (1992)
9. Lam, P., Shiu, W., Xu, B.: On Structure of Some Plane Graphs with Application to Choosability. J. Combin. Theory Ser. B 82, 285–296 (2001)
10. Raspaud, A., Wang, W.: On the Vertex-Arboricity of Planar Graphs. Europ. J. Combin. 29, 1064–1075 (2008)
11. Wang, W., Lih, K.: Structural Properties and Edge Choosability of Planar Graphs without 6-Cycle. Combin. Prob. Comput. 10, 267–276 (2001)
12. Wang, W., Lih, K.: Choosability and Edge Choosability of Planar Graphs without Five Cycles. Appl. Math. Lett. 15, 561–565 (2002)
13. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice-Hall, Upper Saddle River (2001)
14. Zhang, L., Wu, B.: Edge Choosability of Planar Graphs without Small Cycles. Discrete Math. 283, 289–293 (2004)

A Sufficient and Necessary Condition for the Forcing Number of a Bipartite Graph Being Equal to the Minimum Number of Trailing Vertices

Hongwei Wang

Department of Mathematics, Linyi Normal University
Linyi, Shandong 276005, China
wanghw0539@sina.com

Abstract. Riddle [15] showed that the forcing number of a bipartite graph is bounded blow by the minimum number of trailing vertices of the ordering of a color set. In the present work, we improve the trailing-vertex method by Riddle and obtain a necessary condition for the matching forcing number of a bipartite graph being equal to a given natural number k; furthermore, we give a sufficient and necessary condition for the minimum forcing number of bipartite graph being equal to the minimum number of trailing vertices of all standard orderings of a color set.

1 Introduction

Let G be a graph that admits a perfect matching. A forcing set for a perfect matching M of G is a subset S of M, such that S is contained in no other perfect matchings of G. The cardinality of a forcing set of M with the smallest size is called the forcing number of M, and is denoted by $f(G, M)$. The minimum (resp. maximum) forcing number of G, denoted by $f(G)$ (resp. $F(G)$), is the minimum (resp. maximum) value of forcing numbers of all perfecting matchings of G. The spectrum of forcing numbers of G is the set of forcing numbers of all perfect matchings, denoted by $Spec(G)$. Adams et al. [2] have proved that the problem of determining the minimum forcing set of a bipartite graph, in which the maximum degree is 3, is NP-complete .

The concept of perfect matching is corresponding to Kekulé structure in chemistry, and the (matching) forcing number is originated from the degree of freedom of Kekulé structure [6,7,8] which has relation to the resonance of graphs intimately [16,17]. The idea of "forcing" appears in many research fields in both graph theory and combinatorics, such as graph coloring, geodetic set, Latin square and block design. Recently, the matching forcing number has been extensively studied, and new concepts, such as global forcing number [18], anti-Kekulé number [19], anti-forcing number [5,20] and forcing hexagon [4], have been proposed. The authors has investigated the forcing number of toroidal polydex intensively [22]. From chemistry perspective, the freedom degree of Kekulé structure has been studied [13,14,21].

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 258–268, 2009.

On the basis of Hall Theorem [3], by establishing ordering of a color set, Riddle [15] obtained a lower bound for the forcing number of a bipartite graph. In this paper, we improve the concept of ordering of a color set and the trailing-vertex method and obtain a necessary condition for the matching forcing number of a bipartite graph being equal to a given natural number k; furthermore, we give a sufficient and necessary condition for the minimum forcing number of a bipartite graph being equal to the minimum number of trailing vertices of all standard orderings of a color set.

2 Basic Concepts and Some Lemmas

Throughout the paper, we only consider bipartite graphs with a perfect matching. Let G be a bipartite graph with vertex set $V(G)$ and edge set $E(G)$. Suppose that G has bipartition (B, W) such that vertices in B (resp. W) is black (resp. white).

If $T \subseteq V(G)$, then $G[T]$ denotes the subgraph induced by T. Let $H \subseteq G$ be a subgraph of G, then $G - H$ denote the subgraph obtained from G by deleting vertices in H as well as edges incident to vertices in H. Let A be a finite set, we use $|A|$ to denote the cardinality of A. If $b \in B$, then $N(b)$ denote the neighborhood of b, that is, the set of vertices adjacent to b. Define $N[b] := N(b) \cup \{b\}$. If $T \subseteq B$, define $N(T) := \bigcup_{b \in T} N(b)$ and $N[T] := N(T) \cup T$. Define functions α and β on $E(G)$ as follows: for any edge $e = wb \in E(G)$ with $w \in W$ and $b \in B$, $\alpha(e) = w$ and $\beta(e) = b$. For $S \subseteq E(G)$, define $\alpha(S) := \{\alpha(e) | e \in S\}$ and $\beta(S) := \{\beta(e) | e \in S\}$.

A set M of edges is called a matching of G if M consists of independent edges. If a vertex v is incident to an edge in M, then v is called saturated by M, otherwise, v is unsaturated by M. If all the vertices are saturated by M, then M is called a perfect matching (or Kekulé structure in chemistry) of G. A cycle C of G is called M-alternating if the edges of C appear alternately in M and $E(G) \backslash M$. We denote by $c(M)$ the maximum number of disjoint M-alternating cycles in the perfect matching M of graph G. The set of all perfect matchings is denoted by $\mathcal{M}$.

For more concepts and notations, the readers are referred to [3,11].

Definition 1. *Let G be a bipartite graph possessing a perfect matching M with bipartition (B, W), $S \subseteq M \in \mathcal{M}$. For $u \in V(G) \setminus V(S)$, we say S forces u if $|N(u) \setminus V(S)| = 1$. In particular, S W-forces (resp. B-forces) an edge e if $\alpha(e)$ (resp. $\beta(e)$) is forced by S. If there exists a sequence of edges $e_1, e_2, \ldots, e_k$ and a sequence of edge sets $S = S_0, S_1, S_2, \ldots, S_k$ such that $S_i = S_{i-1} \cup \{e_i\}$ and S_{i-1} W-forces (resp. B-forces) $e_i (i = 1, 2, \ldots, k)$, then we say S W-forces (resp. B-forces) the set S_k.*

Lemma 1. [15] *Let $S \subseteq M \in \mathcal{M}$. Then S forces M if and only if S W-forces M (or B-forces M).*

Assign an ordering $e_n > e_{n-1} > \ldots > e_1$ to the edges of M. Then $b_n > b_{n-1} > \ldots > b_1$ is the unique corresponding ordering on the vertices in B such that

$b_i = \beta(e_i)(i = 1, 2, \ldots, n)$. Given an ordering of the vertices in B, we have the following definitions:

Definition 2. [15] *For a vertex $b \in B$, b leads $N(w)$ if b is the largest vertex in $N(w)$. A vertex b is called a leading vertex if it leads at least one set $N(w)$ for some $w \in W$, otherwise, it is called a trailing vertex.*

If $E_i = \{e_1, e_2, \ldots, e_i\}$, $B_i = \beta(E_i) = \{b_1, b_2, \ldots, b_i\}$, denote by $\bar{B}_i$ the set $\{b_n, b_{n-1}, \ldots, b_{i+1}\}$.

Lemma 2. *If E_i W-forces e_{i+1}, then b_{i+1} leads the set $N(w_{i+1})$, where $\beta(e_{i+1}) = b_{i+1}$ and $\alpha(e_{i+1}) = w_{i+1}$.*

Proof. By the definition of E_i W-forces e_{i+1}, all vertices in $N(w_{i+1})$ belong to $\beta(E_i) = \beta_i = \{b_1, b_2, \ldots, b_i\}$ except b_{i+1}, so b_{i+1} is the largest vertex in $N(w_{i+1})$, that is, b_{i+1} leads $N(w_{i+1})$. □

In matching theory of bipartite graphs, Hall Theorem [3] is well known.

Theorem 1. [3] *Let G be a bipartite graph with bipartition (B, W). Then G contains a matching that saturates every vertex in B if and only if*

$$|N(T)| \geq |T| \quad \textit{for all} \quad T \subseteq B.$$

By Hall Theorem, it is natural for us to consider the excess of the subset of B as defined in the following.

Definition 3. [15,22] *Let G be a bipartite graph with color sets B and W. For $T \subseteq B$, the excess of T is defined as $\epsilon(T) := |N(T)| - |T|$. The maximum excess of an ordering $b_n > b_{n-1} > \ldots > b_i > \ldots > b_1$ of B is the maximum value of $\epsilon(\bar{B}_i)$ for all i, that is, $max\{\epsilon(\bar{B}_i)|i = 0, 1, \ldots, n-1\}$. The excess of b_i is defined to be $\epsilon(b_i) = \epsilon(\bar{B}_{i-1}) - \epsilon(\bar{B}_i)$. We call b_i an m-excess vertex, simple m-ex vertex, if $\epsilon(b_i) = m$.*

Lemma 3. [15] *Let G be a bipartite graph possessing a perfect matching M with bipartition B and W. Then $f(G)$ is bounded below by the smallest possible maximum excess or by the minimum number of trailing vertices for all orderings of B.*

We proved the following proposition on trailing vertex, leading vertex and excess of an ordering.

Proposition 1. [22] *Let G be a bipartite graph with color sets B and W, and $b_n > b_{n-1} > \ldots > b_i > \ldots > b_1$ be an ordering of B. Then*

1. *the following statements are equivalent:*
   - *(a) b_i is a trailing vertex;*
   - *(b) $|N(\bar{B}_{i-1})| = |N(\bar{B}_i)|$;*
   - *(c) $\epsilon(\bar{B}_{i-1}) = \epsilon(\bar{B}_i) - 1$;*
   - *(d) b_i is a (-1)-ex vertex.*

2. *the following statements are equivalent:*

   (a) b_i *is a leading vertex;*
   (b) $|N(\bar{B}_{i-1})| - |N(\bar{B}_i)| \geq 1$*;*
   (c) $\epsilon(\bar{B}_{i-1}) \geq \epsilon(\bar{B}_i)$*;*
   (d) b_i *is a non-negative excess vertex, that is,* $\epsilon(b_i) \geq 0$.

For an ordering of B, this proposition gives the geometrical significations of the leading vertex, trailing vertex and 0-ex vertex. In fact, given the ordering of B, it is natural to obtain the ordering of W: vertices which are adjacent to vertices with larger labelings in B have larger labelings. According to the labelings from large to small, if we call vertices in $N(b_i)$ which are not labeled as simultaneously labeled vertices with b_i, which are not labeled, as the vertices which get labels at the same then the leading vertices are those vertices in $N(b_i)$ have more than one simultaneously labeled vertex with b_i. The trailing vertices are those whose neighbors are all labeled.

Definition 4. *An ordering* $b_n > b_{n-1} > \ldots > b_1$ *of* B *is standard if its smallest leading vertex is larger than the largest trailing vertex; non-standard, otherwise.*

In a standard ordering of B, denote the set of all the positive-excess vertices by B_+, the set of 0-ex vertices by B_0, and B_- the set of trailing vertices, then $B = B_+ \cup B_0 \cup B_-$. Denote the set of vertices which are simultaneously labeled with B_0 by W_0. Let $\bar{W}_0 = W \backslash W_0$.

Lemma 4. *Let* G *be a bipartite graph possessing a perfect matching with bipartition* B *and* W*. Then for any non-standard ordering of* B*, there exists a standard ordering such that they have the same number of trailing vertices.*

Proof. Suppose that $|V(G)| = 2n$. By the definition, the largest vertex of any ordering must be leading vertex, and the smallest vertex must be trailing vertex.

Let

$$b_{n-k} > \ldots > b_{j_1} > \ldots > b_i > \ldots > b_{j_r} > \ldots > b_{j_k}$$

be a non-standard ordering of B with trailing vertex set $B_- = \{b_{j_1}, b_{j_2}, \ldots, b_{j_k}\}$ and leading vertex set $\{b_{n-k}, b_{n-k-1}, \ldots, b_1\}$. In what follows, we prove that the ordering

$$b_{n-k} > b_{n-k-1} > \ldots > b_1 > b_{j_1} > b_{j_2} > \ldots > b_{j_k}$$

is a standard ordering.

For convenience, in the ordering

$$b_{n-k} > b_{n-k-1} > \ldots > b_1 > b_{j_1} > b_{j_2} > \ldots > b_{j_k},$$

we denote b_{j_r} by $b'_{j_r}(r = 1, \ldots, k)$, and denote b_i by $b'_i(i = n-k, \ldots, 1)$, then $\bar{B'}_{j_r}$ and $\bar{B'}_i$ have the corresponding meaning.

Let us first prove b_{j_r} $(r = 1, \ldots, k)$ is a trailing vertex of the latter ordering. By Proposition 1, $|N(\bar{B}_{j_r-1})| = |N(\bar{B}_{j_r})|$, hence we have

$$N(b'_{j_r}) = N(b_{j_r}) \subseteq N(\bar{B}_{j_r}) \subseteq N(\bar{B'}_{j_r}).$$

So $|N(\bar{B}'_{j_r-1})| = |N(\bar{B'}_{j_r})|$ and b'_{j_r} (coinciding with b_{j_r}) is a trailing vertex of the latter ordering $(r = 1, \ldots, k)$.

In the following, we will show that b_i $(i = n-k, \ldots, 1)$ is a leading vertex of the latter ordering.

By Proposition 1, we know $|N(\bar{B}_{i-1})| \geq |N(\bar{B}_i)| + 1$, that is,

$$N(b_i) \backslash N(\bar{B}_i) \neq \emptyset.$$

However,

$$N(\bar{B}_i) \supseteq N(\bar{B'}_i), N(b'_i) = N(b_i),$$

then we have

$$N(b'_i) \backslash N(\bar{B'}_i) \neq \emptyset,$$

that is,

$$|N(B'_{i-1})| \geq |N(B'_i)| + 1.$$

By Proposition 1, $b'_i (i = n-k, \ldots, 1)$ is a leading vertex of the latter ordering.

By Definition 4, we obtain that

$$b_{n-k} > b_{n-k-1} > \ldots > b_1 > b_{j_1} > \ldots > b_{j_k}$$

is a standard ordering which has the same number of trailing vertex with non-standard ordering

$$b_{n-k} > \ldots > b_{j_1} > \ldots > b_i > \ldots > b_{j_r} > \ldots > b_{j_k}.$$

The proof is then finished. □

For short, we call the maximum excess of a standard ordering of B excess of the standard ordering. By proposition 1, we easily obtain the following proposition. The proof is omitted.

Proposition 2. *For any standard ordering, the excess is equal to the number of trailing vertices.*

Basing on Lemma 4 and Proposition 2, instead of considering the excess and number of trailing vertices of all orderings, we only need to consider excess and number of trailing vertices of standard orderings. It not only reduces the range of orderings, but also has a stronger geometrical property intuitionally. From now on, if there is no special explanation, the orderings in consideration are standard orderings. Denote by σ_G the smallest value of excess among all the standard orderings.

By Lemma 3, Lemma 4 and Proposition 2, we can obtain the following conclusion:

Theorem 2. *Let G be a bipartite graph possessing a perfect matching M with bipartition W and B. Then the forcing number $f(G, M)$ is bounded below by the smallest value of trailing vertices in B (the minimum excess of standard orderings) over all standard orderings of B, i.e. $f(G, M) \geq \sigma_G$.*

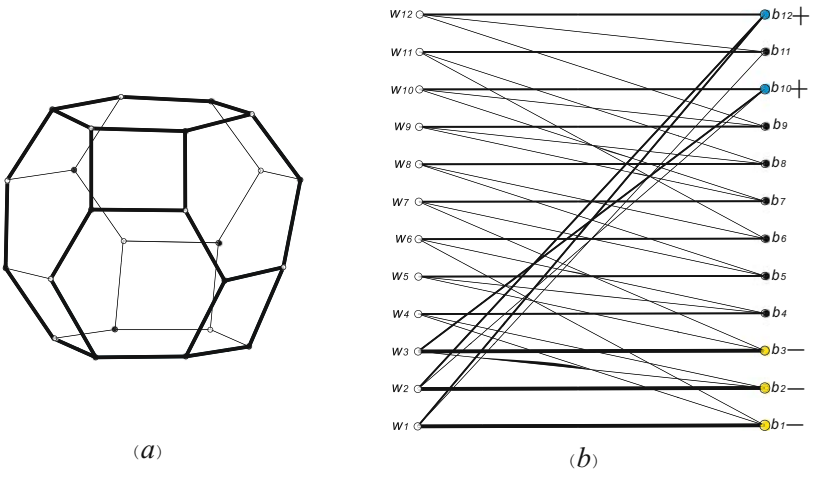


Fig. 1. The molecular graph of $B_{12}N_{12}$ (left) and another drawing of it (right)

Now we give an intuitional explanation to above concepts and results by Fig. 1. A standard ordering of the molecular graph of Boron-Nitrogen Fullerenes $B_{12}N_{12}$ is shown in Fig. 1(*b*): $b_{12} > b_{11} > \ldots > b_4 > b_3 > b_2 > b_1$, where $B_+ = \{b_{12}, b_{10}\}$, $B_0 = \{b_{11}, b_9, \ldots, b_4\}$ and $B_- = \{b_3, b_2, b_1\}$.

According to Proposition 1, the leading vertex set is $B_+ \cup B_0$, and the trailing vertex set is the isolated vertex set $G - G[N[B_+ \cup B_0]]$ which is obtained by deleting $B_+ \cup B_0$ and its neighbors. It is easy to see that $M = \{b_i w_i | 1 \leq i \leq 12\} \in \mathcal{M}$ and $S = \{b_1 w_1, b_2 w_2, b_3 w_3\}$ is a forcing set of M.

By Lemma 1, see Fig. 1, from the bottom up on the left side, we have S W-forces M, while from top to bottom on the right side, we can see that S B-forces M.

Let G be a bipartite graph possessing a perfect matching with a bipartition B and W, $b_n > \ldots > b_{k+1} > b_k > \ldots > b_1$ is a standard ordering of B with the number of trailing vertices k. Define

$$N^*(b_i) \equiv N(b_i) \setminus N(\bar{B}_i) = N(\bar{B}_{i-1}) \setminus N(\bar{B}_i)$$

as the simultaneously labelled set of b_i $(i > k)$, which indicates that the vertices in W get their labelings according to the ordering of B. Vertices in $N^*(b_i)$ can be labeled arbitrarily and it is easy to know $|N^*(b_i)| = \epsilon(b_i) + 1$.

Lemma 5. *Let G be a bipartite graph possessing a perfect matching with bipartition B and W. If S is a minimum forcing set of M, then there exists a standard ordering of B such that $\beta(S)$ is the set of trailing vertices and $B \backslash \beta(S)$ is the set of leading vertices.*

Proof. Omitted. □

Lemma 6. [12] *Let G be a planar bipartite graph with a perfect matching M. Then the forcing number of M is equal to the maximum number of disjoint M-alternating cycles, i.e. $f(G, M) = c(M)$.*

3 Main Results

In this section we give a necessary condition for a natural number k belonging to the spectrum of forcing number in G and a sufficient and necessary condition for the minimum number of trailing vertices being equal to the forcing number $f(G)$.

Lemma 7. *Let G be a bipartite graph possessing a perfect matching with a bipartition B and W. If there is a standard ordering of B such that the induced subgraph $G[(B_+ \cup B_-) \cup (\bigcup_{b_i \in B_+} N^*(b_i))]$ has a perfect matching, then the number of trailing vertices B_- (excess) of the standard ordering is no less than the forcing number $f(G, M)$ of some perfect matching M.*

Proof. By Proposition 2, for any standard ordering, the number of trailing vertices is equal to the excess, that is,

$$|B_-| = \sum_{b_i \in B_+} \epsilon(b_i).$$

Since $|N^*(b_i)| = \epsilon(b_i) + 1$, then we have

$$\Big|B_+ \cup B_-\Big| = \left| \bigcup_{b_i \in B_+} N^*(b_i) \right|.$$

Denote the perfect matching in

$$G\left[\left(B_+ \cup B_-\right) \cup \Big(\bigcup_{b_i \in B_+} N^*(b_i) \Big) \right]$$

by S, then $|S| = |B_+| + |B_-|$.

In the following, we prove $S' = \{e | e \in S, \beta(e) \in B_-\} \subset S$ is a forcing set of some perfect matching M in G.

In fact, $G - S$ is a bipartite graph with unique perfect matching M'. In M', the double edges can be given one by one according to the order obtained by restricting the standard ordering of G on $G - S$. Since for every 0-ex vertex, there exists a unique vertex in $G - S$ which is simultaneously labeled and the edge connecting them is a double edge.

In addition, for $G[(B_+ \cup B_-) \cup (\bigcup_{b_i \in B_+} N^*(b_i))]$, since there is only one vertex which is not saturated by S in the simultaneously labelled set of every positive-excess vertex, so S' W-forces S. Then S' is the forcing set of the perfect matching $M = S \cup M'$ in G, and $|S'| = |B_-|$. By the definition of $f(G, M)$, we have $|B_-| \geq f(G, M)$. □

In Lemma 7, the property of a standard ordering can be suberized as "there is a uniform matching between the trailing vertices and positive-excess vertices". The characterization of a standard ordering is that for every trailing vertex, there exists at least one positive excess vertex such that they are at distance two.

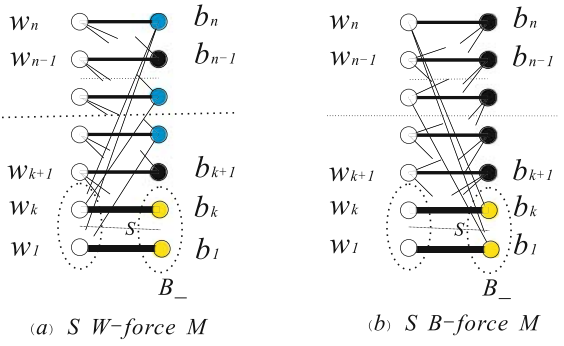


Fig. 2. Illustration for the proof of Theorem 3

Theorem 3. *Let G be a bipartite graph possessing a perfect matching with a bipartition B and W. If $k \in Spec(G)$, then there exists a standard ordering with the number of trailing vertices k such that the induced subgraph $G[(B_+ \cup B_-) \cup (\bigcup_{b_i \in B_+} N^*(b_i))]$ has a perfect matching.*

Proof. Let $M \in \mathcal{M}$, $f(G, M) = k$ and $S = \{w_i b_i | i = 1, 2, \ldots, k\} \subseteq M$ be a minimum forcing set of M.

By Theorem 1, S W-forces M, that is, there exists an edge sequence $e_{k+1} = w_{k+1}b_{k+1}, \ldots, e_n = w_n b_n$ in G. Denote $S_k = S$, $S_{j+1} = S_j \cup \{e_{j+1}\}$, $j = k, k+1, \ldots, n-1$, then S_j W-forces e_{j+1}. Obviously, $M = S \cup \{e_i = w_i b_i | i = k+1, \ldots, n\}$.

By Lemma 2 and Proposition 1, we know $b_j \in B_+ \cup B_0$, $j = k+1, \ldots, n$. There is not edge between b_j and $w_{j'}$ $(j > j' \geq k+1)$ (see Fig. 2(a)). Otherwise, it contradicts S W-forces edge sequence $e_{k+1}, \ldots, e_n$. Therefore, positive excess vertex $b_j \in B_+$ is only adjacent to vertices in $\alpha(S)$ besides w_j.

Thus, we obtain a standard ordering

$$b_n > b_{n-1} > \ldots > b_{k+1} > b_k > \ldots > b_1,$$

such that $\{b_j | j > k\} \subseteq B_+ \cup B_0$ and $B_- \subseteq \{b_k, b_{k-1}, \ldots, b_1\}$. Moreover, $\{w_i b_i | 1 \leq i \leq n\}$ is a perfect matching of G.

By the minimum of the forcing set S, we have $B_+ \cup B_0 = \{b_j | j > k\}$, $B_- = \{b_k, b_{k-1}, \ldots, b_1\}$ and every vertex in $\{w_k, w_{k-1}, \ldots, w_1\}$ belongs to the simultaneously labelled set of B_+.

Otherwise, without loss of generality, suppose that w_1 does not belongs to the simultaneously labeled set of B_+. Since w_1 is not adjacent to vertices in B_0, it is only adjacent to vertices in B_-. Suppose that w_1 is adjacent to $b_{k'}$. Then $k' \leq k$, which contradicts that $b_{k'}$ is a trailing vertex. So $\{w_i b_i | i \leq k\}$ is a matching between the trailing vertex set and the simultaneously labelled set of positive-excess vertices. The induced subgraph $G[B_- \cup (\bigcup_{b_i \in B_+} N^*(b_i))]$ has

a matching saturating B_-, then $G[(B_+ \cup B_-) \cup (\bigcup_{b_i \in B_+} N^*(b_i))]$ has a perfect matching. The proof is completed. □

Theorem 4. *Let G be a bipartite graph possessing a perfect matching with a bipartition B and W. Then $f(G) = \delta_G$ if and only if there exists a standard ordering with the number of trailing vertices δ_G such that the induced subgraph $G[(B_+ \cup B_-) \cup (\bigcup_{b_i \in B_+} N^*(b_i))]$ has a perfect matching.*

Proof. The sufficiency can be obtained by Theorem 2 and Lemma 7, and the necessity can be obtained by Theorem 3. □

4 Concluding Remarks

The new method to determine the forcing number of a bipartite graph by the necessary condition given in Theorem 3 can be named "improved trailing-vertex method" for short. The technique in this method lies in that there exists a matching saturating B_- in the induced subgraph $G[B_- \cup (\bigcup_{b_i \in B_+} N^*(b_i))]$. By the "improved trailing-vertex method", we can give the forcing number of some classes bipartite graphs, see [10,12,22,24]; Specially, for toroidal polyhexes, we have designed a linear algorithm to compute the forcing number [22]. In addition, by constructing a standard ordering with as few trailing vertices as possible, we can obtain an upper bound for the forcing number of bipartite graphs, see [23].

There are some graphs which do not satisfy the condition in Theorem 4, such as $P_6 \times P_4$ and $P_8 \times P_4$. For $P_6 \times P_4$ and $P_8 \times P_4$, it is easy to see that the minimum number σ_G of trailing vertices is two among all orderings. The former can achieve the lower bound since $P_6 \times P_4 = 2$. However, the latter can not achieve the lower bound since P. Afshani et al. [2] have proved that $P_8 \times P_4 = 3$.

Though Theorem 4 provides a sufficient and necessary condition for the smallest number of trailing vertices among all standard orderings of color set B being equal to the forcing number in a bipartite graph, there are two difficulties: one is that the smallest number of trailing vertices is difficult to find out and the other is that even if there exists a standard ordering satisfying the conditions, we may not find a good algorithm to search for it. We need some new techniques to conquer these difficulties. Different techniques and methods are required for different classes of graphs, such as stop signs [10], square grids [12], and toroidal polyhexes [22] and so on. In fact, determining the forcing number of a bipartite graphs with the minimum degree 3 is a $NP-$complete problem [1,2].

By Theorem 3, it is easy to obtain the following result.

Corollary 1. [2] *The forcing number of $P_{2n} \times P_{2n}$ is n.*

Moreover, it should be mentioned that Kleinerman [9] discussed bipartite graphs of which the forcing numbers achieve the lower bound in another way by the edge packing.

Acknowledgement

The author was supported partially by the Priority Academic Discipline Foundation of Linyi Normal University and Shandong province, and the Startup Scientific Research Fund for doctors of Linyi Normal University. Many thanks to my advisor H. Zhang, to Y. Yang and D. Ye for their supports.

References

1. Adams, P., Mahdian, M., Mahmoodian, E.S.: On the Forced Mathching Numbers of Bipartite Graphs. Discrete Math. 281, 1–12 (2004)
2. Afshani, P., Hatami, H., Mahmoodian, E.S.: On the Spectrum of the Forced Matching Number of Graphs. Australasian Journal of Combinatorics 30, 147–160 (2004)
3. Bondy, J.A., Murty, U.S.R.: Graph Theory. Graduate Texts in Mathematics, vol. 244. Springer, Heidelberg (2008)
4. Che, Z., Chen, Z.: Forcing Hexagons in Hexagonal Systems. MATCH Commun. Math. Comput. Chem. 56, 649–668 (2006)
5. Deng, H.: The Anti-Forcing Number of Double Hexagonal Chains. MATCH Commun. Math. Comput. Chem. 60, 183–192 (2008)
6. Harary, F., Klein, D.J., Živković, T.P.: Graphical Properties of Polyhexes: Perfect Matching Vector and Forcing. J. Math. Chem. 6, 295–306 (1991)
7. Harary, F., Slany, W., Verbitsky, O.: On the Computational Complexity of the Forcing Chromatic Number. SIAM Journal on Computing 37(1), 1–19 (2007)
8. Klein, D.J., Randić, M.: Innate Degree of Freedom of a Graph. J. Comput. Chem. 8, 516–521 (1987)
9. Kleinerman, S.: Bounds on the Forcing Numbers of Bipartite Graphs. Discrete Math. 306, 66–73 (2006)
10. Lam, F., Pachter, L.: Forcing Numbers of Stop Signs. Theoretical Computer Sci. 303, 409–416 (2003)
11. Lovász, L., Plummer, M.D.: Matching Theory. Annals of Discrete Math., vol. 29. North-Holland, Amsterdam (1986)
12. Pechter, L., Kim, P.: Forcing Matchings on Square Grids. Discrete Math. 190, 287–294 (1998)
13. Randić, M.: Aromaticity of Polycyclic Conjugated Hydrocarbons. Chem. Revi. 103, 3449–3605 (2003)
14. Randić, M., Klein, D.J.: Kekulé Evalence Structures Revisited, Innate Degrees of Freedom of π-Electron Couplings. In: Trinajstić, N. (ed.) Mathematics and Computational Concepts in Chemistry, pp. 274–282. Horwood/Wiley, NewYork (1985)
15. Riddle, M.E.: The Minimum Forcing Number for the Torus and Hypercube. Discrete Math 245, 283–292 (2002)
16. Shiu, W.C., Lam, P.C.B., Zhang, H.: k-resonance in toroidal polyhexes. J. Math. Chem. 38, 451–466 (2005)
17. Shiu, W.C., Zhang, H.: Acomplete Characterization for k-Resonant Klein-Bottle Polyhexes. J. Math. Chem. 43, 45–59 (2008)
18. Vukičević, D., Sedlar, J.: Total Forcing Number of the Triangular Grid. Mathematical Communications 9, 169–179 (2004)
19. Vukičević, D., Trinajstić, N.: On the Anti-Kekulé Number and Anti-Forcing Number of Cata-Condensed Benzenoids. J. Math. Chem. 43, 719–726 (2008)

20. Vukičević, D., Trinajstić, N.: On the Anti-Forcing Number of Benzenoids. J. Math. Chem. 42, 575–583 (2007)
21. Vukičevič, D., Kroto, H.W., Randić, M.: Atlas of Kekule Valence Structures of Buckminster-fullerene. Croat. Chem. Acta. 78(2), 223–234 (2005)
22. Wang, H., Ye, D., Zhang, H.: The Forcing Number of Toroidalpolyhexes. J. Math. Chem. 43, 457–475 (2008)
23. Wang, H.: The Maximum Forcing Number and the Face Independence Number of a Boron-Nitrogen Fullerene. Technical Report, Linyi Normal University, Shandong, China (2009)
24. Wang, H.: The Forcing Number of Bipartite Klein Bottle Polyhexs. Technical Report, Linyi Normal University, Shandong, China (2008)

On Integrity of Harary Graphs

Fengwei Li, Qingfang Ye, and Baohuai Sheng

Department of mathematics, Shaoxing University
Zhejiang, Shaoxing 312000, China
fengwei.li@eyou.com, fengwei.li@hotmail.com

Abstract. The integrity of a graph $G = (V, E)$ is defined as $I(G) = min\{|S| + m(G - S) : S \subseteq V(G)\}$, where $m(G - X)$ denotes the order of the largest component in the graph $G - X$. This is a better parameter to measure the stability of a network G, as it takes into account both the amount of work done to damage the network and how badly the network is damaged. In this paper, we give the exact values or bounds for the integrity of Harary graphs.

Keywords: Networks, Integrity, Harary Graph, Independence Number.

1 Introduction

In an analysis of the vulnerability of a communication network to disruption, Two qualities that come to mind are the number of elements that are not functioning and the size of the largest remaining subnetwork within which mutual communication can still occur. In particular, in an adversarial relationship, it would be desirable for an opponent's network to be such that the two qualities can be made to be simultaneously small.

Thus, communication networks must be constructed to be as stable as possible, not only with respect to the initial disruption, but also with respect to the possible reconstruction of the network. Many graph theoretical parameters have been used in the past to describe the stability of communication networks. Most notably, the vertex-connectivity and the edge-connectivity have been frequently used. The difficulty with these parameters is that they do not take into account what remains after the graph is disconnected. Consequently, a number of other parameters have been introduced in an attempt to overcome this difficulty, including toughness [5] and edge-toughness [11], integrity [3] and edge-integrity [2], tenacity [8] and edge-tenacity [12], rupture-degree [9]. Unlike connectivity measures, each of these parameters shows not only the difficulty to break down the network but also the damage that has been caused.

Before we formally give the concept of integrity of a graph, we recall some parameters. Let G be a finite simple graph with vertex set $V(G)$ and edge set $E(G)$. For $S \subseteq V(G)$, let $\omega(G - S)$ and $m(G - S)$, respectively, denote the number of components and the order of a largest component in $G - S$. A set $S \subseteq V(G)$ is a cut set of G, if either $G - S$ is disconnected or $G - S$ has only one vertex. For comparing, the following graph parameters are listed.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 269–278, 2009.

The Vertex-Connectivity of G:

$$\kappa(G) = min\{|S| : S \subseteq V(G) \text{ is a cut set of } G\}.$$

The Vertex-Toughness of G (Chvátal (1973) [5]):

$$t(G) = \min\left\{\frac{|S|}{\omega(G-S)} : S \subseteq V(G) \text{ is a cut set of } G\right\}.$$

The Vertex-Tenacity of G (Cozzens et al (1995) [8]):

$$T(G) = \min\left\{\frac{|S| + m(G-S)}{\omega(G-S)} : S \subseteq V(G) \text{ is a cut set of } G\right\}.$$

The Rupture-degree of G (Li et al (2005) [9]):

$$r(G) = \max\{\omega(G-S) - m(G-S) - |S| : S \subseteq V(G) \text{ is a cut set of } G\}.$$

The corresponding edge analogues of these concepts are defined similarly, see [11,12].

The *integrity* of a graph $G = (V, E)$, which was introduced in [3] as a useful measure of the vulnerability of the graph, is defined as follows:

$$I(G) = \min\{|S| + m(G-S) : S \subseteq V(G)\},$$

where $m(G-S)$ denotes the order of the largest component in $G-S$.

A vertex set S of graph $G = (V, E)$ is called an *I-set* of G if it satisfies that $I(G) = |S| + m(G-S)$.

Unlike the connectivity measures, integrity shows not only the difficulty to break down the network but also the damage that has been caused. In [3], Barefoot *et al.* give some basic results on integrity and Clark *et al.* proved that the determination of integrity is NP-complete [6]. For knowing more about integrity, one can see a survey of integrity in [1].

Throughout this paper, a graph $G = (V, E)$ always means a finite simple connected graph with vertex set V and edge set E. We shall use $\lfloor x \rfloor$ to denote the largest integer not larger than x, and $\lceil x \rceil$ the smallest integer not smaller than x. $\alpha(G)$ denotes the independence number of the graph G. We use Bondy and Murty [4] for terminology and notations not defined here.

Since computing the integrity of a graph is NP-complete in general, it becomes an interesting question to calculate the integrity for some special classes of interesting or practically useful graphs. In this paper, we consider the problem of computing the integrity of Harary graphs. We give the exact values or bounds for the integrity of Harary graphs. We will deal with this in the sequel.

2 Integrity of Harary Graphs

In 1962, Harary investigated a problem on reliable communication networks: For given order n and a nonnegative integer l ($l < n$), how to construct a simple

graph G of order n such that $\kappa(G) = l$, and G has as few edges as possible. For any fixed integers n and p such that $p \geq n + 1$, Harary constructed the class of graphs $H_{n,p}$ which are n-connected with the minimum number of edges on p vertices. Thus Harary graphs are examples of graphs which in some sense have the maximum possible connectivity and hence are of interests as possibly having good stability properties. $H_{n,p}$ is constructed as follows:

Case 1: If n is even, let $n = 2r$, then $H_{n,p}$ has vertices $0, 1, 2, \cdots, p-1$, and two vertices i and j are adjacent if and only if $|i - j| \leq r$, where the addition is taken modulo p.

Case 2: If n is odd $(n > 1)$ and p is even, let $n = 2r + 1$ $(r > 0)$, then $H_{n,p}$ is constructed by first drawing $H_{2r,p}$, and then adding edges joining vertex i to vertex $i + p/2$ for $1 \leq i \leq p/2$.

Case 3: If n is odd $(n > 1)$ and p is odd, let $n = 2r + 1(r > 0)$, then $H_{2r+1,p}$ is constructed by first drawing $H_{2r,p}$, and then adding edges joining the vertex i to $i + (p+1)/2$ for $0 \leq i \leq (p-1)/2$. Note that under this definition, the vertex 0 is adjacent to both the vertices $(p + 1)/2$ and $(p - 1)/2$. Again note that all vertices of $H_{n,p}$ have degree n except for the vertex 0, which has degree $n + 1$.

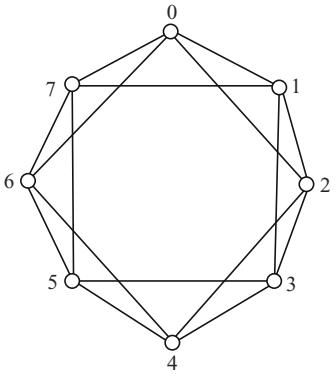


Fig. 1. Graph $H_{4,8}$

As a useful reliable network, Harary graphs have arouse interests for many network designers. Harary [4] proved that the Harary graphs $H_{n,p}$ is n-connected. Ouyang et al [10] gave the scattering number of the Harary graphs. In [7] Cozzens et al gave their exact values or good bounds for the tenacity. In this section, we compute their integrity. Throughout this section, we set the connectivity $n = 2r$ or $n = 2r + 1$ and the number of vertices $p = k(r + 1) + s$ for $0 \leq s \leq r + 1$. So we can see that $p \equiv s \mod (r + 1)$ and $k = \lfloor \frac{p}{r+1} \rfloor$. We assume that the graph $H_{n,p}$ is not complete, and so $n + 1 < p$, which implies that $k \geq 2$.

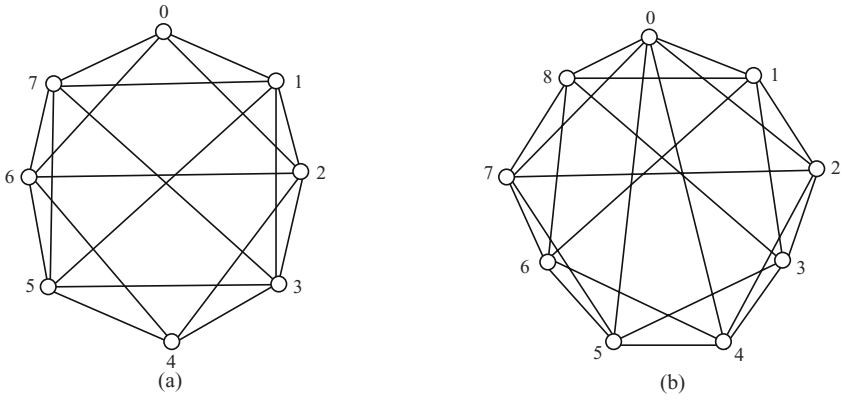


Fig. 2. (a) Graph $H_{5,8}$; (b) Graph $H_{5,9}$

Lemma 2.1. *If S is a minimal I-set for the graph $H_{n,p}$, $n = 2r$, then S consists of the union of sets of r consecutive vertices such that there exists at least one vertex not in S between any two sets of consecutive vertices in S.*

Proof. We assume that the vertices of $H_{n,p}$ are labelled by $0, 1, 2, \cdots, p-1$. Let S be a minimal I-set of $H_{n,p}$ and j be the smallest integer such that $T = \{j, j+1, \cdots, j+t-1\}$ is a maximum set of consecutive vertices such that $T \subseteq S$. Relabel the vertices of $H_{n,p}$ as $v_1 = j, v_2 = j+1, \cdots, v_t = j+t-1$, $\cdots, v_p = j-1$. Since $S \neq V(H_{n,p})$ and $T \neq V(H_{n,p})$, v_p does not belong to S. Since S must leave at least two components of $G - S$, we have $t \neq p-1$, and so $v_{t+1} \neq v_p$. Therefore, $\{v_{t+1}, v_p) \cap S = \emptyset$. Choose v_i such that $1 \leq i \leq t$, and delete v_i from S yielding a new set $S^{'} = S - \{v_i\}$ with $|S^{'}| = |S| - 1$. Now suppose $t < r$. By the definition of $H_{n,p}$ $(n = 2r)$ we know that the edges $v_i v_p$ and $v_i v_{t+1}$ are in $H_{n,p} - S^{'}$. Consider a vertex v_k adjacent to v_i in $H_{n,p} - S^{'}$. If $k \geq t+1$, then $k < t+r$. So, v_k is also adjacent to v_{t+1} in $H_{n,p} - S^{'}$. If $k < p$, then $k \geq p - r + 1$ and v_k is also adjacent to v_p in $H_{n,p} - S^{'}$. Since $t < k$, then v_p and v_{t+1} are adjacent in $H_{n,p} - S^{'}$. Therefore, we can conclude that deleting the vertex v_i from S does not change the number of components, and so $\omega(H_{n,p} - S^{'}) = \omega(H_{n,p} - S)$ and $m(H_{n,p} - S^{'}) \leq m(H_{n,p} - S) + 1$. Thus, we have

$$\begin{aligned} & |S^{'}| + m(H_{n,p} - S^{'}) \\ & \leq |S| + m(H_{n,p} - S) \\ & = I(H_{n,p}). \end{aligned}$$

This is contrary to our choice of S. Thus we must have $t \geq r$. Now suppose $t > r$. Delete v_t from the set S yielding a new set $S_1 = S - \{v_t\}$. Since $t > r$, the edge $v_t v_p$ is not in $H_{n,p} - S_1$. Consider a vertex v_k adjacent to v_t in $H_{n,p} - S_1$.

Then, $k \geq t+1$ and $k \leq t+r$, and so v_k is also adjacent to v_{t+1} in $H_{n,p} - S_1$. Therefore, deleting v_t from S yields $m(H_{n,p} - S_1) = m(H_{n,p} - S) + 1$. So,

$$\begin{aligned} & |S_1| + m(H_{n,p} - S_1) \\ & \leq |S| + m(H_{n,p} - S) \\ & = I(H_{n,p}), \end{aligned}$$

which is again contrary to our choice of S. Thus, $t = s$, and so S consists of the union of sets of exactly r consecutive vertices. □

Lemma 2.2. *There is an I-set S for the graph $H_{n,p}$, $n = 2r$, such that all components of $H_{n,p} - S$ have order $m(H_{n,p} - S)$ or $m(H_{n,p} - S) - 1$.*

Proof. Among all I-sets of minimum order, consider those sets with maximum number of minimum order components, and we let s denote the order of a minimum component. Among these sets, let S be one with the fewest components of order s. Suppose $s \leq m(H_{n,p} - S) - 2$. Note that all of the components must be sets of consecutive vertices. Assume that C_k is a smallest component. Then $|V(C_k)| = s$, and without loss of generality, let $C_k = \{v_1, v_2, \cdots, v_s\}$. Suppose C_e is a largest component, and so $|V(C_e)| = m(H_{n,p} - S) = m$ and let $C_e = \{v_j, v_{j+1}, \cdots, v_{j+m-1}\}$. Let $C_1, C_2, \cdots, C_a$ be the components with vertices between v_s of C_k and v_j of C_e, such that $|C_i| = p_i$ for $1 \leq i \leq a$, and let $C_i = \{v_{i_1}, v_{i_2}, \cdots, v_{i_{p_i}}\}$. Now we construct the vertex set S' as $S' = S - \{v_{s+1}, v_{1_{p_1+1}}, v_{2_{p_2+1}}, \cdots, v_{a_{p_a+1}}\} \cup \{v_{1_1}, v_{2_2}, \cdots, v_{a_1}, v_j\}$. Therefore, $|S'| = |S|$, $m(H_{n,p} - S') \leq m(H_{n,p} - S)$ and $\omega(H_{n,p} - S') = \omega(H_{n,p} - S)$. So we have

$$|S'| + m(H_{n,p} - S') \leq |S| + m(H_{n,p} - S).$$

Therefore, $m(H_{n,p} - S') = m(H_{n,p} - S)$. But, $H_{n,p} - S'$ has one less components of order s than $H_{n,p} - S$, a contradiction. Thus, all components of $H_{n,p} - S$ have order $m(H_{n,p} - S)$ or $m(H_{n,p} - S) - 1$. So, $m(H_{n,p} - S) = \lceil \frac{p - r\omega}{\omega} \rceil$. □

By the above two lemmas we give the exact values of integrity of the Harary graphs for $n = 2r$.

Theorem 2.1. *Let $H_{n,p}$ be a Harary graph with $n = 2r$ and $p = k(r+1) + s$ for $0 \leq s < r+1$. Then*

$$I(H_{n,p}) = \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p\ > 4(r-1), \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

Proof. Let S be a minimum I-set of $H_{n,p}$. By Lemmas 2.1 and 2.2 we know that $|S| = r\omega$ and $m(H_{n,p} - S) = \lceil \frac{p - r\omega}{\omega} \rceil$. Thus, from the definition of integrity we have

$$I(H_{n,p}) = \min \left\{ r\omega + \lceil \frac{p - r\omega}{\omega} \rceil \;\middle|\; 2 \leq \omega \leq k \right\}.$$

Now we consider the function

$$f(\omega) = r\omega + \lceil \frac{p - r\omega}{\omega} \rceil.$$

It is easy to see that $f^{'}(\omega) = r + \lceil \frac{-p}{\omega^2} \rceil = \lceil \frac{r\omega^2 - p}{\omega^2} \rceil$. Since $\omega^2 > 0$, we have $f^{'}(\omega) \geq 0$ if and only if $g(\omega) = r\omega^2 - p \geq 0$. Since the two roots of the equation $g(\omega) = r\omega^2 - p = 0$ are $\omega_1 = -\sqrt{\frac{p}{r}}$ and $\omega_2 = \sqrt{\frac{p}{r}}$. But $\omega_1 < 0$, and so it is deleted. Then if $0 < \omega \leq \lceil \omega_2 \rceil$, we have $f^{'}(\omega) \leq 0$, and so $f(\omega)$ is an decreasing function; if $\omega \geq \lceil \omega(2) \rceil$, then $f^{'}(\omega) \leq 0$, and so $f(\omega)$ is a increasing function. Thus, we have the following cases:

Case 1: If $p \leq 4(r-1)$, then $\lfloor \omega_2 \rfloor \leq 2$. Since we know that $2 \leq \omega \leq k$, we have that $f(\omega)$ is a increasing function and the minimum value occurs at the boundary. So, $\omega = 2$ and $I(H_{n,p}) = f(2) = r + \lceil \frac{p}{2} \rceil$.

Case 2: If $p > 4(r-1)$, then $\lfloor \omega_2 \rfloor > 2$. So, we have

Subcase 2.1: If $2 \leq \omega \leq \lfloor \omega_2 \rfloor$, then $f(\omega)$ is an decreasing function.

Subcase 2.2: If $\lceil \omega_2 \rceil \leq \omega \leq k$, then $f(\omega)$ is a increasing function.

Thus the minimum value occurs when $\omega = \lfloor \omega_2 \rfloor$. Then, $I(H_{n,p}) = f(\lfloor \omega_2 \rfloor) = (m-1)r + \lceil p/m \rceil$, where $m = \lfloor \omega_2 \rfloor = \lfloor \sqrt{p/r} \rfloor$. The proof is now complete. □

The following two lemmas can be found in [7] or easily seen

Lemma 2.3 ([7]). *Let $H_{n,p}$ be a Harary graph with $n = 2r+1$ and p even. Then*

$$\alpha(H_{n,p}) = \begin{cases} k, & if\ p \neq 0\ mod(n+1); \\ k-1, & if\ p = 0\ mod(n+1). \end{cases}$$

Lemma 2.4 ([3]). *If H is a spanning subgraph of a connected graph G, then $I(G) \geq I(H)$.*

Lemma 2.5. *Let G be a noncomplete connected graph of order n. Then $I(G) \leq n - \alpha(G) + 1$.*

Proof. Let S be a maximum independence set of G, then $|S| = \alpha(G)$. Let $A = V(G) - S$, then $\omega(G) = \alpha(G)$, $m(G-A) = 1$, and $|A| = n - \alpha(G)$, so, by the definition of the integrity, we have $I(G) = min\{|S| + m(G-S)\} \leq |A| + m(G-A) = n - \alpha(G) + 1$. □

Theorem 2.2. *Let $H_{n,p}$ be a Harary graph with p even, n odd and $n = 2r+1$, then*

(1) *If $p \leq 4(r-1)$, then*

$$r + \lceil p/2 \rceil \leq I(H_{n,p}) \leq \begin{cases} kr + s + 1, if\ p \neq 0\ mod(n+1) \\ kr + s + 2,\ \ if\ p = 0\ mod(n+1). \end{cases}$$

(2) *If* $p > 4(r-1)$, *then*

$$(m-1)r + \lceil p/m \rceil \leq I(H_{n,p}) \leq \begin{cases} kr+s+1, if\ p \neq 0\ mod(n+1); \\ kr+s+3,\ \ if\ p = 0\ mod(n+1). \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$, $k = \lfloor p/(r+1) \rfloor$.

Proof. Since $V(H_{2r+1,p}) = V(H_{2r,p})$, $E(H_{2r+1,p}) \subseteq E(H_{2r,p})$, it is obvious that $H_{2r,p}$ is a connected spanning subgraph of $H_{2r+1,p}$. So, by Lemma 2.4 we have

$$I(H_{2r+1,p}) \geq I(H_{2r,p}) = \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p > 4(r-1). \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

On the other hand, by Lemmas 2.3 and 2.5 we have

$$I(H_{n,p}) \leq \begin{cases} kr+s+1, if\ p \neq 0\ mod(n+1); \\ kr+s+3,\ \ if\ p = 0\ mod(n+1). \end{cases}$$

The theorem is thus proved. □

From above theorem, the following corollary is easily obtained.

Corollary 2.1. *Let* n *be odd. If* p *is even and* $s \neq 0$, *then*

$$kr+s+1 \geq I(H_{n,p}) \geq \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p > 4(r-1), \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

Corollary 2.2. *Let* n *be odd. If* p *is even,* $s = 0$ *and* k *is odd, then*

$$kr+1 \geq I(H_{n,p}) \geq \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p > 4(r-1), \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

Corollary 2.3. *Let* n *be odd. If* p *is even,* $s = 0$ *and* k *is even, then*

$$kr+2 \geq I(H_{n,p}) \geq \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p > 4(r-1), \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

Lemma 2.6 ([7]). *Let $H_{n,p}$ be a Harary graph such that n is odd, $n = 2r+1$, p is even, $r \geq 2$, $0 < s < r+1$, $s < k$, and k is odd. Then there exists a cut set S with kr elements, such that $\omega(H_{n,p} - S) = k$ and $m(H_{n,p} - S) = 2$.*

Theorem 2.3. *Let $H_{n,p}$ be a Harary graph such that $n = 2r+1$, p is even, $r \geq 2$, $0 < s < r+1$, $s < k$, and k is odd. Then*

$$kr + 2 \geq I(H_{n,p}) \geq \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p > 4(r-1). \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

Proof. First note that if $r = 1$, then $s = 1$ and so $p = 2k+1$, a contradiction. So, $r \geq 2$. By Lemma 2.6 and the definition of integrity we have $I(H_{n,p}) \leq |S| - m(H_{n,p} - S) = kr + 2$.

On the other hand, by Theorem 2.2 we have

$$I(H_{n,p}) \geq \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p > 4(r-1). \end{cases}$$

The theorem is thus proved. □

Lemma 2.7 ([7]). *Let $H_{n,p}$ be a Harary graph such that n is odd, $n = 2r+1$ and p is even, Then $p \equiv 0 mod(n+1)$ if and only if $s = 0$ and k is even.*

Theorem 2.4. *Let $H_{n,p}$ be a Harary graph such that $n = 2r+1$ is odd and p is even, $s = 0$ and k is even. Then*

$$2 + kr \geq I(H_{n,p}) \geq \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p > 4(r-1). \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

In the following we will give some lower and upper bounds for $I(H_{n,p})$ such that both n and p are odd.

Lemma 2.8 ([7]). *Let $H_{n,p}$ be a Harary graph such that both n and p are odd, $n = 2r+1$ and $r > 0$. Then*

$$\alpha(H_{n,p}) = \begin{cases} k - kr - s - 1, & if\ p \neq 1\ mod(n+1); \\ k - kr - s - 3, & if\ p = 1\ mod(n+1). \end{cases}$$

Theorem 2.5. *Let $H_{n,p}$ be a Harary graph such that both n and p are odd, $n = 2r+1$ and $r > 0$. Then*

(1) *If* $p \leq 4(r-1)$, *then*

$$r + \lceil p/2 \rceil \leq I(H_{n,p}) \leq \begin{cases} (2+k)r - k + s + 2, if\ p \neq 1\ mod(n+1); \\ (2+k)r - k + s + 4,\ \ if\ p = 1\ mod(n+1). \end{cases}$$

(2) *If* $p > 4(r-1)$, *then*

$$(m-1)r + \lceil p/m \rceil \leq I(H_{n,p}) \leq \begin{cases} (2+k)r - k + s + 2, if\ p \neq 1\ mod(n+1); \\ (2+k)r - k + s + 4,\ \ if\ p = 1\ mod(n+1). \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

Proof. The proof is similar to that of Theorem 2.2. □

Lemma 2.9 ([7]). *Let* $H_{n,p}$ *be a Harary graph such that* n *and* p *are all odd,* $n = 2r + 1$, $r > 0$. *Then* $p \equiv 1 mod(n+1)$ *if and only if* $s = 1$ *and* k *is even.*

Theorem 2.6. *Let* $H_{n,p}$ *be a Harary graph such that* $n = 2r + 1$ *and* p *are all odd,* $r > 0$, $s = 1$, *and* k *is even. Then*

$$(2+k)r - k + 5 \geq I(H_{n,p}) \geq \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p > 4(r-1). \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

Lemma 2.10 ([7]). *Let* $H_{n,p}$ *be a Harary graph such that* $n = 2r + 1$ *and* p *are all odd,* $r \geq 2$, $1 < s < r + 1$, $s < k$, *and* k *is even. Then there exists a cut set* S *with* $kr + 1$ *elements, such that* $\omega(H_{n,p} - S) = k$ *and* $m(H_{n,p} - S) = 2$.

Theorem 2.7. *Let* $H_{n,p}$ *be a Harary graph such that* $n = 2r + 1$ *is odd and* p *is odd,* $r \geq 2$, $1 < s < r + 1$, $s < k$, *and* k *is even. Then*

$$3 + kr \geq I(H_{n,p}) \geq \begin{cases} r + \lceil p/2 \rceil, & if\ p \leq 4(r-1); \\ (m-1)r + \lceil p/m \rceil, & if\ p > 4(r-1). \end{cases}$$

where $m = \lfloor \sqrt{p/r} \rfloor$.

3 Conclusion

The robustness of a distributed system of computers can be represented by the integrity of the graph describing the network. The authors present and prove a formula to calculate the exact values or bounds for the rupture degrees of Harary graphs.

Acknowledgements

This work was supported by NSFC (No.10871226). The authors are grateful to the anonymous referees for helpful comments on an earlier version of this article.

References

1. Baggas, K.S., Beineke, L.W., Goddard, W.D., Lipman, M.J., Pippert, R.E.: A Survey of Integrity. Discrete Applied Math. 37/38, 13–28 (1992)
2. Baggas, K.S., Beineke, L.W., Lipman, M.J., Pippert, R.E.: Edge-Integrity - a Survey. Discrete math. 124, 3–12 (1994)
3. Barefoot, C.A., Entringer, R., Swart, H.: Vulnerability in Graphs - A Comparative Survey. J. Combin. Math. Combin. Comput. 1, 12–22 (1987)
4. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. Macmillan/ Elsevier, London, New York (1976)
5. Chvátal, V.: Tough Graph and Hamiltonian Circuits. Discrete Math. 5, 215–228 (1973)
6. Clark, L.H., Entringer, R.C., Fellows, M.R.: Computational Complexity of Integrity. J. Combin. Math. Combin. Comput. 2, 179–191 (1987)
7. Cozzens, M., Moazzami, D.: The Tenacity of the Harary Graphs. J. Combin. Math. Combin. Comput. 16, 33–56 (1994)
8. Cozzens, M., Moazzami, D., Stueckle, S.: The Tenacity of a Graph. In: Proc. Seventh International Conference on the Theory and Applications of Graphs, pp. 1111–1122. Wiley, New York (1995)
9. Li, X.L., Li, Y.K., Zhang, S.G.: Rupture Degree of Graphs. International Journal of Computer Mathematics 82(7), 793–803 (2005)
10. Ouyang, K.Z., Ouyang, K.Y.: Relative Breaktivity of Graphs. J. Lanzhou University 29(3), 43–49 (1993)
11. Peng, Y.H., Chen, C.C., Koh, K.M.: On the Edge-Toughness of a Graph (I). Southeast Asian Math. Bull. 12, 109–122 (1988)
12. Piazza, B.L., Roberts, F.S., Stueckle, S.K.: Edge Tenacious Networks. Networks 25, 7–17 (1995)
13. Zhang, S.G., Wang, Z.G.: Scattering Number in Graphs. Networks 37, 102–106 (2001)

A Note on n-Critical Bipartite Graphs and Its Application

Yueping Li and Zhe Nie

School of Electronics and Information Engineering, Shenzhen Polytechnic
Shenzhen 518055, China
leeyueping@gmail.com, niezhe@oa.szpt.net

Abstract. In matching theory, n-critical graphs play an important role in the decomposition of graphs with respect to perfect matchings. Since bipartite graphs cannot be n-critical when $n > 0$, we amend the classical definition of n-critical graphs and propose the concept of n-critical bipartite graphs. Let $G = (B, W; E)$ be a bipartite graph with $n = |W| - |B|$, where B and W are the bipartitions of vertex set, E is the edge set. Then, G is n-critical if when deleting any n distinct vertices of W, the remaining subgraph of G has a perfect matching. Furthermore, an algorithm for determining n-critical bipartite graphs is given which runs in $O(|W||E|)$ time, in the worst case. Our work helps to design a job assignment circuit which has high robustness.

Keywords: Matching, n-critical graph, assignment.

1 Introduction

An important application of matching theory is the job assignment problem. An instance of the problem is given in Fig. 1 by a bipartite graph $G = (B, W)$. Assume that vertex set $V(B) = \{j_1, \ldots, j_a\}$ is a set of jobs demanding simultaneous handling, and vertex set $V(W) = \{m_1, \ldots, m_b\}$ is a set of machines. The edge set is determined as follows: If one job can be assigned to a certain machine via the line of the circuit, i.e. the machine is able to accomplish the job, then there is an edge connecting them. Otherwise, no edge lies between them. The same as the classical job assignment, each job requires only a single machine for processing and every machine can serve only one job at the same time.

Matching extendability of graphs was proposed to investigate whether arbitrary matching with fix cardinality can be extended to a perfect matching. When the job assignment circuit is a k-extendable graph, any matching with k edges can be extended to a perfect matching. It indicates that the assignment circuit allows arbitrary k jobs specify the desired machines in advance. The remaining jobs can find machines to process. It is a good virtue of the assignment circuit, since it has high flexibility.

Different from matching extendability, our scenario is to evaluate the robustness of the assignment circuit. It is usual that more than a machines are deployed in order to obtain high robustness. Let the number of machines be $a + n$ where

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 279–286, 2009.

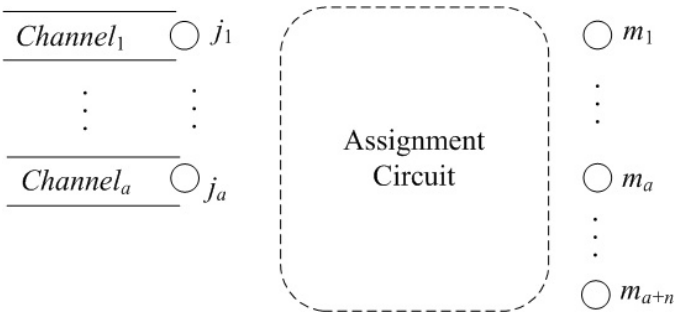


Fig. 1. Job assignment circuit

n is a positive integer, shown in Fig. 1. It is natural to ask whether the job assignment circuit still works while no more than n machines are failed. That is, each job can find a machine to process even n machines are unavailable.

We propose the concept of "n-critical bipartite graph" to represent the circuit which has this property. We reveal the relation between n-critical bipartite graphs and n-extendable connected bipartite graphs. Furthermore, an algorithm for determining n-critical bipartite graphs is developed.

2 Definition and Preliminary Result

The graphs discussed in this paper will be finite and simple. Let $G = (V, E)$ be a simple graph with vertex-set $V(G)$ and edge-set $E(G)$. We use the notation $V(X)$ to denote the set of all the vertices in X. Similarly, $E(Y)$ denotes the set of all the edges in Y.

A *matching* M is a subset of $E(G)$ of which no edges share the same vertex. A vertex is said to be *saturated* with respect to a matching if it appears in a matching edge and *unsaturated*, otherwise. A matching containing k edges is called to be a k-matching. A matching is *perfect* if it saturates all the vertices. Furthermore, a matching is called a *defect-d* matching if it covers exactly $|V(G)| - d$ vertices of G.

Let M be a matching of graph G. A path $p = e_1, e_2, \ldots, e_j$ is said to be an *M-alternating* graph if and only if $e_i \in M \Leftrightarrow e_{i+1} \notin M$ where $1 \leq i < j$. For a vertex x, let $\Gamma(x)$ denote the set of vertices that are adjacent to x.

Let G be a graph with at least $2k+2$ vertices. If G has a k-matching and every k-matching is contained in a perfect matching, then G is said to be *k-extendable*. The *extendability* of G, denoted by $\text{ext}(G)$, is defined to be the maximum integer k_0 such that graph G is k_0-extendable. A great deal of results related to extendable graphs have been obtained. For more details, we refer readers to [7,8].

A graph G is called *n-critical* if after deleting any n vertices the remaining subgraph has a perfect matching. This concept is proposed by Favaron [1] and Yu [10], independently. It is a generalization of the notions of factor-critical graphs and bicritical graphs (the cases when $n = 1$ and $n = 2$). Many investigations

of n-critical graphs have been presented in [1,2,10]. For example, degree sum, toughness, binding number, connectivity, etc.

Liu and Yu [6] proposed the concept of "(n,k,d)-*graph*" for generalization of matching extensions in graphs. Let G be a graph with vertex set $V(G)$. Let n,k,d be non-negative integers such that $n+2k+d \leq |V(G)|-2$ and $|V(G)|-n-d$ is even. If when deleting any n vertices in $V(G)$ the remaining subgraph contains a k-matching and every k-matching can be extended to a defect-d matching, then G is called an (n,k,d)-graph. It is clear that n-critical graph is equivalent to $(n,0,0)$-graph.

Since the job assignment circuit is a bipartite graph, we discuss bipartite graphs. We give some known results which will help to prove our main theorems.

Theorem 1. *(Plummer [9]). Let G be a connected bipartite graph on v vertices with bipartition (B,W). Suppose that n is a positive integer such that $n \leq (v-2)/2$. Then the following are equivalent:*

(i) *G is n-extendable;*
(ii) *For all $u_1,u_2,\ldots,u_n \in B$ and $w_1,w_2,\ldots,w_n \in W$,*
$G = G-u_1-u_2-\ldots-u_n-w_1-w_2-\ldots-w_n$ has a perfect matching.

Let G be a bipartite graph with bipartitions B and W. Suppose the vertices of W (B, respectively) are colored white (black, respectively). An orientation is defined as follows: orient all edges of M from the white vertices to the black ones and orient the other edges of G from the black vertices to the white ones. The resulting directed graph is denoted by $\overrightarrow{G}(M)$. An example is illustrated in Fig. 2.

Let D be a directed graph and u,v be a pair of distinct vertices. We call u to v *k-arc-connected* if the removal of any fewer than k arcs results in a subgraph which is connected from u to v. The *arc-connectivity* of D from u to v, denoted by $\lambda(u,v)$, defined as the maximum integer k such that u to v is k-arc-connected. Let $D=(B,W)$ be a bipartite directed graph. Some terms are defined as follows:

$$\lambda^{wb} := \min\{\lambda(u,v) \mid u \in W \text{ and } v \in B\}$$
$$\lambda^{bw} := \min\{\lambda(u,v) \mid u \in B \text{ and } v \in W\}$$

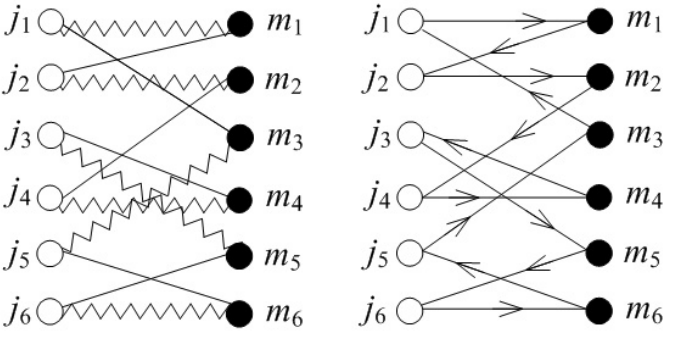


Fig. 2. An example of orientation

Theorem 2. *(Zhang and Zhang [11]) Let G be a connected bipartite graph with a perfect matching M. Then, $ext(G) = \lambda^{bw}$.*

Furthermore, let $G = (B, W; E)$ be a bipartite graph, where B and W are the bipartitions of vertex set, E is the edge set. Zhang and Zhang [11] proposed an $O(|B||E|)$ algorithm for determining whether G is k-extendable which is the fastest one by far.

It is straightforward that a bipartite graph cannot be an (n, k, d)-graph where $n > 0$. Thus, Li and Lou amended the definition of (n, k, d)-graph with respect to bipartite graphs [5]. They required that the n vertices to be deleted lie in the bipartition with more vertices. The reason is that if the vertices to be removed can be chosen in different bipartitions, the graph can not be an (n, k, d)-graph, for fixed n, k, d. They presented the relation of $(n, k, 0)$-bipartite graphs and $(n+k)$-extendable bipartite graphs, in the case $k > 0$. In addition, they developed an algorithm for determining $(n, k, 0)$-bipartite graphs, where $k > 0$.

We borrow the constrain of Li and Lou [5] to define n-critical bipartite graphs which demands the n vertices to be removed belong to the bipartition with more vertices. We show the relation of n-critical graphs (i.e., $(n, 0, 0)$-graphs) and n-extendable graphs for bipartite graphs.

3 Main Results

It is clear that Theorem 1 and Theorem 2 require that the considered graph should be connected. Thus, we firstly discuss the connectivity of n-critical graphs in the following theorem.

Theorem 3. *Let $G = (B, W)$ be a bipartite graph where $|W| > |B|$ and $n = |W| - |B|$. If G is n-critical where $n > 0$, then graph G is connected.*

Proof. Suppose graph G is not connected. Suppose C_1 and C_2 are two connected components of G. If C_1 or C_2 contains exactly one vertex, then graph G does not have a matching to saturate such vertex. Hence, G cannot be n-critical. It follows that the components of G have more than one vertex.

Case 1: $|V(C_1) \cap V(W)| < |V(C_1) \cap V(B)|$. In this case, since the n vertices to be removed lie in the bipartition W, it can be concluded that the component C_1 does not have a perfect matching no matter how to choose these n vertices. Therefore, graph G is not n-critical.

Case 2: $|V(C_1) \cap V(W)| = |V(C_1) \cap V(B)|$. In this case, we choose a vertex, denoted by u, to be one of the n vertices which are supposed to be deleted. It is clear that when these n vertices are removed, the component C_1 does not have a perfect matching. Hence, G cannot be n-critical in this case.

Case 3: $|V(C_1) \cap V(W)| > |V(C_1) \cap V(B)|$. This case is divided further into three subcases.

Case 3.1. $|V(C_1) \cap V(W)| - |V(C_1) \cap V(B)| > n$. It is trivial that C_1 does not have a perfect matching after any n vertices are removed. So does graph G.

Case 3.2. $|V(C_1) \cap V(W)| - |V(C_1) \cap V(B)| = n$. We can conclude that the component C_2 satisfies $|V(C_2) \cap V(W)| = |V(C_2) \cap V(B)|$ which is discussed in Case 2. Thus, G is not n-critical.

Case 3.3. $|V(C_1) \cap V(W)| - |V(C_1) \cap V(B)| < n$. Let $a = \min\{n, |V(C_1) \cap V(W)|\}$. We choose exactly a vertices of $V(C_1) \cap V(W)$ which are supposed to be deleted. It is trivial that the component C_1 does not have a perfect matching after deleting these n vertices. So does graph G.

Hence, graph G is connected. □

A construction is introduced, secondly. For a bipartite graph G with bipartition B and W. Without loss of generality, we suppose $|W| \geq |B|$. Let $d = |W| - |B|$. In our problem, the value of d is larger than 0. Add d new vertices, denoted by X, in bipartition B such that $\Gamma(u) = V(W)$ for every $u \in X$. The resulting graph is denoted by $\widetilde{G}$. An example of construction is shown in Fig. 3. It is straightforward that the resulting graph of this construction is a connected graph.

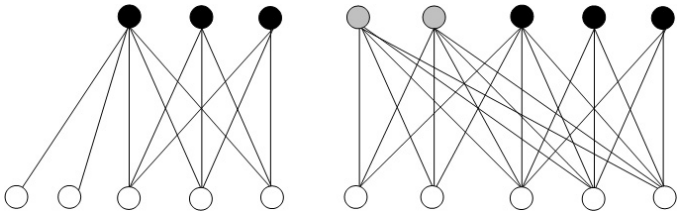


Fig. 3. An example of construction

Theorem 4. *Let $G = (B, W)$ be a bipartite graph where $|W| > |B|$. Let $n = |W| - |B|$. Then graph G is n-critical if and only if graph $\widetilde{G}$ is n-extendable.*

Proof. Let the new added vertices in bipartition B in graph $\widetilde{G}$ be the set X (i.e., $X = V(\widetilde{G}) \backslash V(G)$).

Sufficiency, first. Suppose that graph $\widetilde{G}$ is n-extendable. Let S be arbitrary n vertices of bipartition W in graph G. It is straightforward that $S \subseteq V(\widetilde{G})$ and $|S| = |X| = n$. By Theorem 1, $\widetilde{G} - S - X$ has a perfect matching, denoted by PM. Since $V(PM) \cap V(B) = \emptyset$, the matching PM is also a matching of G. In addition, PM is a perfect matching of $G - S$. It follows that graph G is n-critical.

Necessity. Suppose that graph G is n-critical. Before showing graph $\widetilde{G}$ is n-extendable, we need to point out $\widetilde{G}$ has an n-matching. Assume that $X = \{x_1, x_2, \ldots, x_n\}$. Let $Y = \{y_1, y_2, \ldots, y_n\}$ be arbitrary n distinct vertices in bipartition W of $\widetilde{G}$. Since $|V(W)| > n$, the set Y exists. Each vertex in X is adjacent to every vertex in W by the construction of $\widetilde{G}$. Therefore, the edge (x_k, y_k) belongs to $E(\widetilde{G})$ for any $k \in \{1, 2, \ldots, n\}$. It is clear that $\{(x_k, y_k) | 1 \leq k \leq n\}$ is an n-matching of $\widetilde{G}$.

Let M be any n-matching of $\widetilde{G}$. We denote the matching edges in M which have an end-vertex in X by $M_1 = \{(b'_1, w'_1), (b'_2, w'_2), \ldots, (b'_i, w'_i)\}$, where $\{b'_1,$

$b'_2, \ldots, b'_i\} \subseteq X$ and $\{w'_1, w'_2, \ldots, w'_i\} \subseteq V(W)$. Let $M_2 = \{(b_1, w_1), (b_2, w_2), \ldots, (b_j, w_j)\}$ be the matching $M - M_1$, where $\{b_1, b_2, \ldots, b_j\} \subseteq V(B)$ and $\{w_1, w_2, \ldots, w_j\} \subseteq V(W)$. It is straightforward that $i + j = n$.

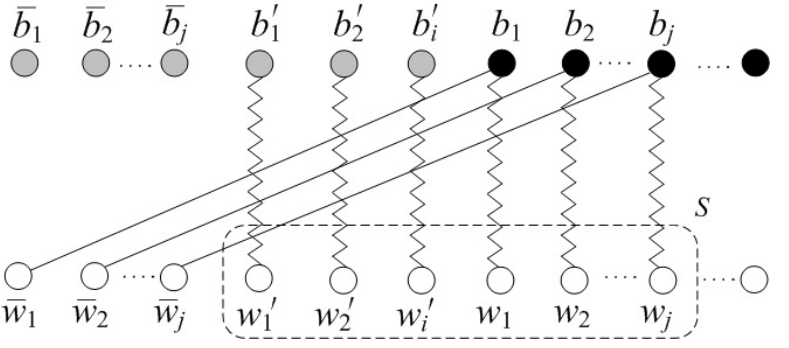


Fig. 4. The matchings M and M'_2

Let $S = V(M) \cap V(W)$ and $X' = X \setminus V(M_1)$. Since graph G is n-critical, $G - S$ has a perfect matching, denoted by M'. Assume that the matching edges of M' saturate the vertices $\{b_1, b_2, \ldots, b_j\}$ are $(b_1, \overline{w_1}), (b_2, \overline{w_2}), \ldots, (b_j, \overline{w_j})$, denoted by M'_2. Since $i + j = n$ and $|V(M_1) \cap X| = i$, we have $|X'| = j$. Suppose the vertices of X' are $\overline{b_1}, \overline{b_2}, \ldots, \overline{b_j}$. Since every vertex in X is adjacent to all vertices of bipartition W, the edges $(\overline{b_k}, \overline{w_k})$ belong to $E(\widetilde{G})$ for all $k \in \{1, 2, \ldots, j\}$. It can be verified that $\{(\overline{b_k}, \overline{w_k}) | 1 \leq k \leq j\} \cup M \cup (M' \setminus M'_2)$ is a perfect matching of graph $\widetilde{G}$. Since the matching M is contained in this perfect matching, graph $\widetilde{G}$ is n-extendable. □

Theorem 4 indicates that if the job assignment circuit is n-critical, the job can be handed out even n machines fail. It is natural to investigate the situation that fewer machines are out of work.

Corollary 1. *Let $G = (B, W)$ be a bipartite graph where $|W| > |B|$. Let $n = |W| - |B|$. If G is n-critical, then graph $G - S$ has a matching M saturating bipartition B, for any vertex set S where $S \subseteq V(W)$ and $0 \leq |S| < n$.*

Proof. Let S be any vertex subset of bipartition W such that $|S| < n$. Let S' be any subset of $V(W) \setminus S$ such that $|S'| + |S| = n$. Since G is n-critical, $G - S - S'$ has a perfect matching, denoted by M. It is straightforward that matching M saturates every vertex of bipartition B. Therefore, M is a matching of $G - S$ saturating B. □

It is interesting to discuss the situation that n' machines fail where n' is less than the number of extra machines.

Theorem 5. *Let $G = (B, W)$ be a bipartite graph where $|W| > |B|$. Let $n = |W| - |B|$. If graph $\widetilde{G}$ is n'-extendable, then graph $G - S$ has a matching saturating bipartition B, for any vertex set S where $S \subseteq V(W)$ and $0 < |S| \leq n' < n$.*

Proof. Let X be the new added vertices in graph $\widetilde{G}$ (i.e., $V(\widetilde{G})\backslash V(G)$). Choose arbitrary n' distinct vertices of X, denoted by X'. Since $|X| = n$, such vertex set X' exists.

For any vertex set S where $S \subseteq V(W)$ and $0 < |S| \leq n' < n$, let S' be any subset of $V(W)\backslash S$ such that $|S'| + |S| = n'$. Since $\widetilde{G}$ is n'-extendable, $\widetilde{G} - X' - S - S'$ has a perfect matching, denoted by M, by Theorem 1. It is clear that $V(B) \subseteq V(\widetilde{G} - X' - S - S')$. Thus, M is a matching of $\widetilde{G} - X' - S - S'$ saturating bipartition B. Since $(X' \cup S') \cap V(B) = \emptyset$ and $X \cap V(B) = \emptyset$, matching M saturates B in graph $\widetilde{G} - X - S$ (i.e., $G - S$). □

Algorithms. The algorithm for determining n-critical bipartite graphs can be derived from Theorem 4 and the algorithm for searching the extendability in [11]. Furthermore, the algorithms for some various versions mentioned in Corollary 1 and Theorem 5 can also be developed.

It is straightforward that the time complexity of these algorithms is bounded by Zhang and Zhang's algorithm [11], which is $O(|V||E|)$ where V is the vertex set and E is the edge set. Thus, our algorithm runs in $O(|W||E|)$ time for bipartite graph $G = (B, W; E)$, where $|W| \geq |B|$.

4 Conclusion

We propose the concept of n-critical bipartite graphs motivated by the application of job assignment problem. A relation between n-critical graphs and n-extendable graphs for bipartite graphs is presented. Based on our result, an algorithm for determining n-critical graphs is developed. Furthermore, algorithms for some various application are also discussed.

Zhang and Zhang [11] stated that a graph is 1-extendable if and only if it is strongly connected. Gabow and Jordan [3] developed an algorithm for augmenting a graph to be strongly connected by adding fewest edges. In the light of this, Li and Lou [4] presented an algorithm for augmenting a graph to be 1-extendable. It is clear that Li and Lou's algorithm is applicable for 1-critical graphs. Furthermore, since Zhang and Zhang [11] revealed the connection of the extendability and arc-connectivity by Theorem 2, we believe our results help to investigate the problem of augmenting a graph to be n-critical. It is an important issue in the design of job assignment circuit.

Acknowledgement. This work is supported by the Science and Technology Project of Shenzhen City, Grant 07KJCE140.

References

1. Favaron, O.: On k-Critical Graphs. Discuss. Math. Graph Theory 16, 41–51 (1996)
2. Favaron, O.: Extendability and Factor-Criticality. Discrete Math. 213, 115–122 (2000)

3. Gabow, H., Jordan, T.: How to Make a Square Grid Framework with Cables Rigid. SIAM Journal of Computing 30, 649–680 (2000)
4. Li, Y., Lou, D.: Matching Extendability Augmentation in Bipartite Graphs. In: International Multiconference of Engineers and Computer Scientists (IMECS 2007), IAENG, Hongkong, pp. 2291–2296 (2007)
5. Li, Y., Lou, D.: Finding the $(n, k, 0)$-Extendability in Bipartite Graphs and its Application. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4489, pp. 401–409. Springer, Heidelberg (2007)
6. Liu, G., Yu, Q.: Generlization of Matching Extensions in Graphs. Discrete Math. 231, 311–320 (2001)
7. Plummer, M.: Extending Matchings in Graphs: a Survey. Discrete Math. 127, 277–292 (1994)
8. Plummer, M.: Extending Matchings in Graphs: an Update. Congr. Numer. 116, 3–32 (1996)
9. Plummer, M.: Matching Extension in Bipartite Graphs. Congr. Numer. 54, 245–258 (1986)
10. Yu, Q.: Characterizations of Various Matching Extensions in Graphs. Australas. J. Combin. 7, 55–64 (1993)
11. Zhang, F., Zhang, H.: Construction for Bicritical Graphs and k-Extendable Bipartite Graphs. Discrete Math. 306, 1415–1423 (2006)

Real-Time Algorithm Scheme for n-Vehicle Exploration Problem*

Xiaoya Li and Jinchuan Cui

Institute of Applied Mathematics, Academy of Mathematics and Systems Science
Chinese Academy of Sciences, Beijing 100190, China
{xyli,cjc}@amss.ac.cn

Abstract. We consider a new kind of n-vehicle exploration problem. Given n vehicles, respectively denoted by $v_1, v_2, \cdots, v_n$, the oil capacity of v_i is a_i, and the oil consumption amount of v_i is b_i. The n vehicles depart from the same place, and steer straight to the same direction. Without oil supply in the middle, but some vehicles can stop wherever and give its oil to any others. The question is how to arrange the order of n vehicles to make one of the vehicles go the farthest. In this paper, we propose a couple of heuristic algorithms and construct the real-time algorithm scheme. We can find a solution in a reasonable time complexity and approximation ratio under the real-time scheme. We simulate the proposed algorithms, and the results show that the approximate ratio of heuristic algorithm is as well as be above 98%.

Keywords: n-vehicle exploration problem, real-time algorithm scheme, heuristic algorithm, efficient algorithm.

1 Introduction

Many very different-looking problems with direct or at least indirect connections to real applications prove to be NP-hard or NP-complete. We cannot get their exact solutions easily [1,2]. Actually, our research goal is to make better decision, so a better response is to find an optimal or sub-optimal polynomial time algorithm under some circumstances. So we have to construct various models to represent the original problem firstly and then to explore 'good' algorithms to get the right answers on decision maker's demand.

Here, 'real-time' characteristic is always ignored by algorithm designers. We emphasize the real-time characteristic because we think it is very popular and enjoyed by many NP-hard problems. As we all known, uncertainty is very pervasive in applications, and it makes a lot of problems complicated. When we try to solve a problem, we have at least two choices: one is getting an optimal solution by consuming a great amount of computational resources, the other is getting a sub-optimal solution quickly but the solution might be far away from

* This research is supported by 973 Program (2006CB701306), and Key Laboratory of Management, Decision and Information Systems, CAS.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 287–300, 2009.

the exact answer. It seems like that we cannot make the best of both worlds. but by considering real-time characteristic, maybe we can get a better strategy to solve the problem, even in a compromised way. So although both the optimal and approximate algorithms are fixed methods, by using real-time concept, we can construct an algorithm scheme to make the algorithms dynamic [3].

The strategy we solve a problem is according to the external and subjective requirements, including the structural complexity of the problem, the computer's capability, the computational time requirement, and the approximation ratio target. For instance, given an n-vehicle exploration problem, we want to find the distribution of its complexity by exploring the structure of the problem. Then we can build an algorithm scheme to choose the most appropriate algorithm according to the complexity degree of different instances, the requirements of computational time, and the tolerance of approximation degree. All of these work are based on the analysis of complexity and not done in a static way, which means, by providing executive operations we try to get the most satisfied strategy to choose the proper algorithm and to support decision.

Here, two fundamental goals in our real-time algorithm scheme are finding algorithms with provably good run times and with provably good or optimal solution quality. A heuristic is an algorithm that gives up one or both of these goals. For example, it usually finds pretty good solutions, but there is no proof the solutions could not get arbitrarily bad; or it usually runs reasonably quickly, but there is no argument that this will always be the case [4]. Even though with these pitfalls, heuristic algorithm is usually adopted in practice and engineering. The other reason for us to adopt heuristic is that for many practical problems, a heuristic algorithm may be the only way to get good solutions in a reasonable amount of time [1,5].

In this paper, we will provide a heuristic and its improved algorithm, which will solve the n-vehicle exploration problem in an efficient way. To make up the pitfalls of heuristic, we pose a lot of numerical examples to testify our heuristic algorithms and fail to find even one counterexample. The numerical experiment results are very good no matter the scale of n is big or small.

The rest of this paper is organized as follows. Section 2 describes the modeling of n-vehicle exploration problem and transforms it into a scheduling one. Section 3 explores a special case and proposes an efficient algorithm and Section 4 investigates the time complexity of n-vehicle exploration problem. Section 5 proposes a heuristic algorithm with time complexity of $O(n^2)$, an improved heuristic algorithms with time complexity of $O(n^3)$, and the real-time algorithm scheme. Section 6 reports the results of the simulated algorithms. In the end, Section 7 provides conclusions.

2 Modeling of the n-Vehicle Exploration Problem

The n-vehicle exploration problem is stated as follows [6]:

Given n vehicles, respectively denoted by $v_1, v_2, \cdots, v_n$, the oil capacity of v_i is a_i, and the oil consumption amount of v_i is b_i. The n vehicles depart from the

same place, and steer straight to the same direction. Without oil supply in the middle, but some vehicles can stop wherever and give its oil to any others. The question is how to schedule the order to make one of the vehicles go farthest, and at last all the vehicles can return to the start point?

Let $v_{i_1} \Rightarrow v_{i_2} \Rightarrow \cdots\cdots \Rightarrow v_{i_{n-1}} \Rightarrow v_{i_n}$ denotes the order, where vehicles in arrow's back provides oil to the vehicles in arrow's head. which can be seen from Figure 1.

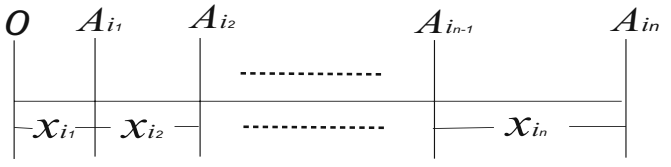


Fig. 1. Order related to distance for n-vehicle

In Fig. 1, A_{i_j} denotes the farthest distance v_{i_j} can arrive. x_{i_j} denotes the additional distance v_{i_j} runs than $v_{i_{j-1}}$. Let S_i denote the farthest distance of the n vehicles, there is:

$$S_i = \sum_{j=1}^{n} x_{i_j}$$

$$s.t. \begin{cases} 2x_{i_n} \times b_{i_n} = a_{i_n} \\ 2x_{i_{n-1}} \times (b_{i_{n-1}} + b_{i_n}) + 2x_{i_n} \times b_{i_n} = a_{i_{n-1}} + a_{i_n} \\ \cdots\cdots \\ 2x_{i_1} \times (b_{i_1} + b_{i_2} + \cdots + b_{i_n}) + \cdots + 2x_{i_n} \times b_{i_n} = a_{i_1} + a_{i_2} + \cdots + a_{i_n} \end{cases} \tag{1}$$

The above Equations (1) can be solved and changed into the following format:

$$S_i = \sum_{j=1}^{n} x_{i_j} = \frac{1}{2}\Big(\frac{a_{i_1}}{b_{i_1} + b_{i_2} + \cdots + b_{i_n}} + \frac{a_{i_2}}{b_{i_2} + \cdots + b_{i_n}} + \cdots + \frac{a_{i_n}}{b_{i_n}}\Big) \tag{2}$$

So the original n-vehicle exploration problem is transformed into a scheduling problem as follows:

Given $a_1, a_2, \cdots, a_n; b_1, b_2, \cdots, b_n$, an order $i_1, i_2, \cdots, i_n$ is being required to maximize the value

$$S_i = \frac{1}{2}\Big(\frac{a_{i_1}}{b_{i_1} + b_{i_2} + \cdots + b_{i_n}} + \frac{a_{i_2}}{b_{i_2} + \cdots + b_{i_n}} + \cdots + \frac{a_{i_n}}{b_{i_n}}\Big).$$

Because each S_i corresponds to a permutation of n vehicles, so there are $n!$ choices of S, then we can get $S_{\max} = \max\{S_i | i = 1, 2, \ldots, n!\}$.

3 Design and Analysis of Efficient Algorithm

By modeling, we transform the original n-vehicle exploration problem to a scheduling problem whose time complexity reaches to $O(n!)$. It's an exponential computational time complexity, and we need to investigate deeply the structure of the problem to get some clues and to make the problem simpler. Before designing the algorithm scheme, we will firstly analyze the 2-vehicle case, and find the determinative parameter for obtaining the optimal solution. Then we will extend the 2-vehicle to n-vehicle case, and obtain a special case that proved to be efficiently solvable.

3.1 2-Vehicle Exploration Problem

Li and Cui [7] have investigated the 2-vehicle exploration problem, and proved the following lemma.

Lemma 1. *In an n-vehicle exploration problem, when $n = 2$, we can compare the value a_1/b_1^2 and a_2/b_2^2, the smaller one related vehicle supports oil to the other vehicle, can make the two vehicles go farthest.*

According to Lemma 1, a/b^2 is the unique parameter to determine the order. Actually, this parameter is also quite useful in the heuristic algorithm.

3.2 A Special Case of n-Vehicle Model

If we extend the 2-vehicle to n-vehicle, we can find a special case which can be solved efficiently (i.e. the time complexity is no greater than $O(n^3)$). This special instance was firstly found by Li and Cui, and the following theorem and polynomial algorithm were also proved [7].

Theorem 1. *In an n-vehicle exploration problem, if*

$$\frac{a_1}{b_1^2} \leq \frac{a_2}{b_2^2} \leq \cdots \leq \frac{a_{n-1}}{b_{n-1}^2} \leq \frac{a_n}{b_n^2}$$

and

$$\frac{a_1}{b_1} \leq \frac{a_2}{b_2} \leq \cdots \leq \frac{a_{n-1}}{b_{n-1}} \leq \frac{a_n}{b_n}$$

are both satisfied, then we can get an order $v_1 \Rightarrow v_2 \Rightarrow \cdots \Rightarrow v_n$ in which the n vehicles can get the farthest distance and return to the start point at last.

Algorithm 1. (For Special Case of n-Vehicle Exploration Problem):

Step 1. Order n vehicles in terms with the value of a/b;
Step 2. Compute S by using the order calculated by Step 1;
Step 3. Output S and the order.

There are $n!$ counts of S need to be computed in the original n-vehicle exploration problem, but in special case, we may only compute one S to get the answer. It can be verified that the complexity of Algorithm 1 is $O(n^2)$.

4 Complexity of n-Vehicle Exploration Problem

The special case introduced by Theorem 1 only takes up a small part of the whole set of n-vehicle exploration problem. In general case, the condition of the consistency of a/b and a/b^2 is not satisfied. Is there an efficient algorithm can solve the more complex instances? Before answering the question, it is necessary to study the structure of the problem and analyze the complexity's changing pattern.

4.1 Primary Analysis of the Complexity

In an n-vehicle exploration problem, suppose that i_m, i_{m+1} are respectively denoted by i, j, which are two adjacent vehicles in the order $v_{i_1} \cdots \Rightarrow v_{i_m} \Rightarrow v_{i_{m+1}} \cdots \Rightarrow v_{i_n}$, then the following theorem can be obtained by adopting Equations (2).

Theorem 2. *[8] Two kinds of situations need to be considered as follows:*

(i) *If* $a_i/[b_i \times (b_i + b_{i_{m+2}} + \cdots + b_{i_n})] \leq a_j/[b_j \times (b_j + b_{i_{m+2}} + \cdots + b_{i_n})]$, *then keeping the order of* v_i *and* v_j *will get the farther distance;*

(ii) *If* $a_i/[b_i \times (b_i + b_{i_{m+2}} + \cdots + b_{i_n})] > a_j/[b_j \times (b_j + b_{i_{m+2}} + \cdots + b_{i_n})]$, *then changing the order of* v_i *and* v_j *will get the farther distance.*

4.2 General Analysis of the Complexity

The special case introduced by Theorem 1 shows a strictly consistency of the variable a/b and a/b^2. The following theorem will extend the special case to more disorder cases, and explore the complexity structure.

Theorem 3. *In an n-vehicle exploration problem, if there are m vehicles not satisfying the following consistency:*

$$\frac{a_1}{b_1^2} \leq \frac{a_2}{b_2^2} \leq \cdots \leq \frac{a_{n-1}}{b_{n-1}^2} \leq \frac{a_n}{b_n^2} \quad \text{and} \quad \frac{a_1}{b_1} \leq \frac{a_2}{b_2} \leq \cdots \leq \frac{a_{n-1}}{b_{n-1}} \leq \frac{a_n}{b_n},$$

then the upper bound of the count of S is $m! \times (m+1)^{n-m}$.

Proof. To get through the proof, we use inductive technique and divided the proof into three parts.

Part 1: $m = 0$**.** In this part, according to Theorem 1, only 1 kinds of order need to be considered. As $1 = 1 \times 1^n$, the theorem follows in this case.

Part 2: $m = 1$**.** In this part, suppose that v_1 does not satisfy the consistency, since the rest $(n-1)$ vehicles can be considered as the special case of Theorem 1. Then we don't have to compute so many cases to get the optimal solution. For example, if v_i is adjacent to v_j, and we also know that $i > j > 1$, then let v_j provide oil to v_i will run farther. Similarly, if the other $(n-1)$ vehicles are adjacent, then the order must be $v_2 \Rightarrow v_3 \Rightarrow \cdots\cdots \Rightarrow v_{n-1} \Rightarrow v_n$. Next, we will go into the detail about this case and list all the possibilities as follows.

1) Assume that v_1 runs farthest, then the optimal order in this case is $v_2 \Rightarrow \cdots\cdots \Rightarrow v_{n-1} \Rightarrow v_n \Rightarrow v_1$. Only one kind of order need to be computed in this situation.

2) Respectively assume that $v_2, v_3, \ldots, v_n$ runs farthest, and v_1 the second farthest. Same with the above situation, the optimal order of the rest vehicles can be easily got. $(n-1)$ kinds of order need to be computed and compared to get the optimal order respectively.

3) Choose two vehicles from the rest $(n-1)$ vehicles and assume that only these two vehicles go farther than v_1. In this situation, C_{n-1}^2 kinds of order need to be computed and compared to get the corresponding optimal order.

.....

n) Assume that v_1 runs the nearest, and the corresponding optimal order is $v_1 \Rightarrow v_2 \Rightarrow \cdots\cdots \Rightarrow v_{n-1} \Rightarrow v_n$.

By summing up all the situations above, the counts of order need to computed equals to $C_{n-1}^0 + C_{n-1}^1 + \cdots + C_{n-1}^{n-1} = 2^{n-1}$. The optimal order must be in one of these 2^{n-1} kinds, So when $m = 1$, the number of S needs to be computed is $1! \times (1+1)^{n-1}$, which equals to $m! \times (m+1)^{n-m}$.

Part 3: from $m = k$ to $m = k+1$. In this part, suppose that the theorem follows when $m \leq k$, which means that at most $k! \times (k+1)^{n-k}$ kinds of order need to be considered when there are m vehicles don't satisfy the consistency of a/b and a/b^2. Next, we will prove that when $m = k+1$, the theorem also follows.

As there is one more vehicle doesn't satisfy the consistency, i.e. v_{k+1}. Then the original set of vehicles don't satisfy the consistency, marked as $\mathcal{A} = \{1, 2, \ldots, k\}$, now changes into $\mathcal{B} = \{1, 2, \ldots, k, k+1\}$.

Assume that p counts of vehicles, $0 \leq p \leq k$, are chosen from set $\mathcal{A}$, will run farther than v_{k+1}. Then fix the variable p, we will study in this case, at most how many kinds of order need to be computed. Respectively, let v_{k+1} run the 1_{st}, 2_{nd}, ..., n_{th} position in the order.

1) Let v_{k+1} run the nearest, if $p < k$, no solution is corresponding to this situation.

.....

k-p) Let v_{k+1} run the $(k-p)_{th}$ position of the order, no solution is corresponding to this situation.

.....

k-p+1) Let v_{k+1} be in the $(k-p+1)_{th}$ position of the order, and the order can be divided into two parts: left to v_{k+1}, the count of order is $C_{n-k-1}^0 \times (k-p)! \times (k-p+1)^0$; then the corresponding right part's count of order is $p! \times (p+1)^{n-k-1}$. So the whole number of order in this situation is $C_k^p \times C_{n-k-1}^0 \times (k-p)! \times (k-p+1)^0 \times p! \times (p+1)^{n-k-1}$.

.....

n-p) Let v_{k+1} run the $(n-p)_{th}$ position of the order and the order can be divided into two parts: left to v_{k+1}, the count of order is $C_{n-k-1}^{n-k-1} \times (k-p)! \times (k-p+1)^{n-k-1}$; then the corresponding right part's count of order is $p! \times (p+1)^0$.

So the whole number of order in this situation is $C_k^p \times C_{n-k-1}^{n-k-1} \times (k-p)! \times (k-p+1)^{n-k-1} \times p! \times (p+1)^0$.

. . . . . .

n-p+1) Let v_{k+1} run the $(n-p+1)_{th}$ position of the order, no solution is corresponding to this situation.

. . . . . .

n) Let v_{k+1} run the farthest, same with the above analysis, no solution is corresponding to this situation.

Summing up all the categories above, the counts of order need to be computed equals to $C_k^p \times p! \times (k-p)! \times \sum_{i=0}^{n-k-1}[C_{n-k-1}^i \times (p+1)^{n-k-1-i} \times (k-p+1)^i]$, which is equal to $k! \times (k+2)^{n-k-1}$. As can be seen that the number of order drops the variable p. As $0 \leq p \leq k$, and for different value of variable p, the number of order is $k! \times (k+2)^{n-k-1}$, so the whole number of order is $(k+1)! \times (k+2)^{n-k-1}$. The theorem follows. □

Theorem 3 provides an upper bound of the complexity for different instances. When $m = n$, the complexity turns out to be $n!$. Here, the $m! \times (m+1)^{n-m}$ is determined by two variables m, n. If the inconsistency only happens between n_s vehicles, and $n_s \ll n$, the whole complexity decreases to $m! \times (m+1)^{n_s-m}$. So to determine the whole complexity of a problem, it is necessary to make sure that the m and n_s are the right determinations.

Definition 1. *[9] In an n-vehicle exploration problem, if there are m vehicles distributed in n_s vehicles not satisfying the consistency of a/b^2 and a/b as displayed in Theorem 1, then the problem's complexity entropy is denoted by E, which is equals to* $\log_n m! \times (m+1)^{n_s-m}$.

E reflects the the problem's disorder degree, and gives the quantity reflection of the problem's complexity. By using E, we can set the benchmark for designing the real-time algorithm scheme.

4.3 Other Special Complex Instances

Theorem 3 sets a gloomy result for finding the optimal solution of general case of n-vehicle exploration problem. The upper bound of complexity means that a lot of computing and comparison work should be taken in the seeking process for optimal solution. According to Theorem 3 we can also deduct that there is no 'short cut' to confirm the optimal order and to reach the farthest distance S_{max} unless taking $n!$ times of comparisons in the worst case. Even when only one vehicle of all doesn't match the consistency, the complexity is 2^{n-1}. So the analysis of complexity of the problem seems to lead to an annoying result, which needs exponential time complexity for finding the optimal solution.

In Example 1, except v_1 doesn't satisfy the consistency, the other 8 vehicles satisfy the condition of Theorem 1. The optimal order is $v_5 \Rightarrow v_8 \Rightarrow v_7 \Rightarrow v_6 \Rightarrow v_9 \Rightarrow v_1 \Rightarrow v_3 \Rightarrow v_4 \Rightarrow v_2$, which maintains the other 8 vehicles' order of consistency. This kind of cases can be found a lot, and if we suppose that only one vehicle doesn't satisfy the consistency, that is: $a_1/b_1^2 \leq a_2/b_2^2 \leq \cdots \leq$

Table 1. Example 1

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
a	51	60	62	65	120	135	140	150	155
b	9	7	10	8	30	26	28	32	27

$a_{n-1}/b_{n-1}^2 \le a_n/b_n^2$ and $a_2/b_2 \le \cdots \le a_{n-1}/b_{n-1} \le a_n/b_n \le a_1/b_1$, can we seek some simpler solution for this situation? To answer this question, we pose the following observations, which need to be proved.

Observation 1. *Given an n-vehicle exploration problem and an order* $v_1 \Rightarrow v_2 \Rightarrow \cdots\cdots \Rightarrow v_{n-1} \Rightarrow v_n$. *If* v_k *and* v_{k+1} *satisfy the consistency of* a/b *and* a/b^2, *and* $a_{k-1}/b_{k-1} \le a_k/b_k \le a_{k+1}/b_{k+1}$, $a_k/b_k^2 \le a_{k+1}/b_{k+1}^2 \le a_{k-1}/b_{k-1}^2$, *then the order* $v_k \Rightarrow v_{k-1} \Rightarrow v_{k+1}$ *will run farther than the order* $v_{k+1} \Rightarrow v_{k-1} \Rightarrow v_k$.

Observation 2. *Given an n-vehicle exploration problem and* v_1 *doesn't satisfy the consistency of* a/b *and* a/b^2, *which means that* $a_1/b_1^2 \le a_2/b_2^2 \le \cdots \le a_{n-1}/b_{n-1}^2 \le a_n/b_n^2$ *and* $a_2/b_2 \le \cdots \le a_{n-1}/b_{n-1} \le a_n/b_n \le a_1/b_1$. *In this case, only n kinds of order need to be considered to get the optimal distance* S_{max}.

Observation 3. *Given an n-vehicle exploration problem:*

(i) *If 2 vehicles don't satisfy the consistency of* a/b *and* a/b^2, *then* $2^{n-1} + (n-1)(n-2)$ *kinds of permutation need to be considered to get the optimal distance* S_{max}.

(ii) *If 3 vehicles don't satisfy the consistency of* a/b *and* a/b^2, *then* $3! \times (C_{n-2}^1 + 2C_{n-2}^2 + C_{n-2}^3 + 2^{n-1})$ *kinds of permutation need to be considered to get the optimal distance* S_{max}.

According to the above observations, we can assume that the lower bound of the complexity is at least greater than $m!$, which means that when $m = n$, the complexity is around $O(n!)$. So maybe there exists some sort of "short cut" (especially according to Observation 2), the whole complexity for the general n-vehicle exploration problem is exponential type.

5 Real-Time Algorithm Scheme

To construct algorithm scheme, we need to delineate the boundary between cases that can be efficiently solved and those that can only be solved with unreasonable amount of computer resources. It's quite an important job, which can help us to know the structural complicatedness of the problem.

5.1 Basic Enumerate Algorithm

Before proposing the database of our algorithms to support the real-time scheme, we provide the following exact algorithm by using enumerate technique. The exact algorithm is realized by designing the following recursion program 'recursion (a, b, n)'. The initial data is $a = (a_1, a_2, \cdots, a_n)$, $b = (b_1, b_2, \cdots, b_n)$.

Algorithm 2. (For General Case of n-Vehicle Exploration Problem)

Step 1. If $n = 2$, according to Lemma 1, let the vehicle which has the greater a/b^2 run farther. Calculate S according to formula (2).
Step 2. If $n > 2$, let $v_1, v_2, \ldots, v_n$ respectively run the nearest position of the order, use program "recursion()" to compute the farthest distance the remaining $(n-1)$ vehicles can reach.
Step 3. Iteratively run Step 2 until $(n-1)$ decreases to 2, then according to Step 1, calculate the S. Finally obtain n groups of distances. Compare them to get the maximal distance S.
Step 4. Output the S.

It can be verified that the complexity of Algorithm 2 is $O(n!)$.

5.2 Improved Enumerate Algorithm

According to Theorem 1, if one vehicle's a/b and a/b^2 are both the smallest of all the vehicles, then the vehicle should run the nearest position. So we improve Algorithm 2 to the following algorithm, which will save much more time to get the optimal solution. The initial data is still $a = (a_1, a_2, \cdots, a_n)$, $b = (b_1, b_2, \cdots, b_n)$.

Algorithm 3. (For General Case of n-Vehicle Exploration Problem)

Step 1. If $n = 2$, according to Lemma 1, then let the vehicle which has the greater a/b^2 run farther. Calculate S according to Equation (2).
Step 2. If $n > 2$, find $a_j/b_j = \min\{a_i/b_i, i = 1, 2, \ldots, n\}$. From v_1 to v_n, if v_i satisfies $a_i/b_i^2 \le a_j/b_j^2$, then let the vehicle run the nearest position, use program "recursion()" to compute the farthest distance the remaining $(n-1)$ vehicles can reach.
Step 3. Iteratively run Step 2 until $(n-1)$ decreases to 2, then according to Step 1, calculate S. Finally get n groups of distances. Compare them to get the maximal distance S.
Step 4. Output the S.

5.3 Heuristic Algorithm for General Large Scale Instance

We pick up two variables a/b and a/b^2 from above analysis, and use them to construct better algorithm. By considering Lemma 1, we construct the following heuristic algorithm. (According to the numerical experimental, we drop the variable a/b, and only keep a/b^2 in heuristic algorithms.)

Algorithm 4. (For General Large Scale Instance)

Step 1. Let $p_i = i, a_{p_i} = a_i, b_{p_i} = b_i$, $i = 1, 2, \ldots, n$;
Step 2. If $a_j/b_j^2 = \max\{a_i/b_i^2, i = 1, 2, \ldots, n\}$, then let $i_n = j$, $p_j = n$, $a_{p_j} = a_n, b_{p_j} = b_n, x_n = a_j/b_j$, and $\sum b = b_j$;
Step 3. For $q = 2$, $q < n$, $q++$, find

$$\frac{a_j}{(b_j + \sum b)^2} = \max\left\{\frac{a_{p_1}}{(b_{p_1} + \sum b)^2}, \frac{a_{p_2}}{(b_{p_2} + \sum b)^2}, \cdots, \frac{a_{p_{n-q+1}}}{(b_{p_{n-q+1}} + \sum b)^2}\right\}.$$

Let $i_{n-q+1} = j$, $p_j = n - q + 1, a_{p_j} = a_{n-q+1}, b_{p_j} = b_{n-q+1}$,
$x_{n-q+1} = a_j/(b_j + \sum b)$, $\sum b = \sum b + b_j$;
Step 4. Output $x_1, x_2, \cdots, x_n, i_1, i_2, \cdots, i_n$.
($S = x_1 + x_2 + \cdots + x_n$ is the optimal solution.)

5.4 Improved Heuristic Algorithm

In Example 2, the order of a/b is (increasing order) $v_3 \to v_8 \to v_{10} \to v_1 \to v_2 \to v_6 \to v_7 \to v_9 \to v_5 \to v_4$; and the order of a/b^2 is (increasing order) $v_3 \to v_8 \to v_{10} \to v_6 \to v_2 \to v_1 \to v_7 \to v_9 \to v_5 \to v_4$. We can see from the two arrays and find that only the middle part ($v_1 \to v_2 \to v_6$ Vs. $v_6 \to v_2 \to v_1$) don't satisfy the consistency.

Table 2. Example 2

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	20	30	40	89	78	67	46	38	90	35
b	7	9	27	8	11	16	10	20	13	15

The optimal order of Example 2 is $v_3 \Rightarrow v_8 \Rightarrow v_{10} \Rightarrow v_1 \Rightarrow v_2 \Rightarrow v_6 \Rightarrow v_7 \Rightarrow v_9 \Rightarrow v_5 \Rightarrow v_4$, and the order calculated by Algorithm 4 is $v_1 \Rightarrow v_3 \Rightarrow v_8 \Rightarrow v_{10} \Rightarrow v_2 \Rightarrow v_7 \Rightarrow v_6 \Rightarrow v_9 \Rightarrow v_5 \Rightarrow v_4$.

Similar with Algorithm 3, according to Theorem 1, if one vehicle's a/b and a/b^2 are both the smallest of all the vehicles, then the vehicle should run the nearest position. By using this rule can improve the solution of Example 2. Moreover, because the position i_n and x_{i_n} play major contributions to the value of S_i, putting much more considerations in choosing the farthest vehicle will help to improve the Algorithm 4.

Algorithm 5. (Improved Heuristic Algorithm):

Step 1. Let $p_i = i, a_{p_i} = a_i, b_{p_i} = b_i$, $i = 1, 2, \ldots, n$, $\mathcal{C} = \emptyset$, $\mathcal{I} = \emptyset$;
Step 2. Find $a_j/b_j^2 = \max\{a_i/b_i^2, i = 1, 2, \ldots, n\}$,for $i = 1, 2, \ldots, n$, if
$a_i/b_i \geq a_j/b_j$, then add i in the set $\mathcal{C}$.
Step 3. For each index j in the set $\mathcal{C}$, let $i_n = j$, $p_j = n, a_{p_j} = a_n, b_{p_j} = b_n$,
$x_n = a_j/b_j$, and $\sum b = b_j$. Run Step 4 to get the distance S_j;
Step 4. For $q = 2$, $q < n$, $q{+}{+}$, find $a_j/b_j^2 = \max\{a_{p_i}/b_{p_i}^2, i = 1, 2, \ldots, n-q+1\}$;
For $i = 1, 2, \ldots, n$, if $a_i/b_i \geq a_j/b_j$, then add i in the set $\mathcal{I}$.
Then find the $a_j/(b_j + \sum b)^2 = \max\{a_i/(b_i + \sum b)^2, i \in \mathcal{I}\}$;
Let $i_{n-q+1} = j$, $p_j = n - q + 1, a_{p_j} = a_{n-q+1}, b_{p_j} = b_{n-q+1}$,
$x_{n-q+1} = a_j/(b_j + \sum b)$, $\sum b = \sum b + b_j$, set $\mathcal{I} = \emptyset$;
Step 5. Find $S_{max} = max\{S_j, j \in \mathcal{C}\}$;
Step 6. Output $x_1, x_2, \cdots, x_n, i_1, i_2, \cdots, i_n$.
($S = x_1 + x_2 + \cdots + x_n$ is the optimal solution.)

It can be verified that the complexity of Algorithm 5 is $O(n^3)$.

5.5 Algorithm Scheme

Given an n-vehicle exploration problem, we can compute its entropy E. Suppose that the up limit of the complexity we can bear is equal to entropy Q:

1) If $E = 0$, choose efficient Algorithm 1;
2) If $0 < E \leq Q$, choose Algorithm 3;
3) If $E > Q$, choose approximation or heuristic algorithm.

Here, the real-time characteristic means that after determining the complexity degree of the problem, decision maker can choose the proper algorithm by considering requirements such as computer's capability, requirement of compute time and space, approximation ratio's tolerance.

To determine the variable E, two variables m and n_s need to be picked out. This is a difficult procedure especially when the n_s turns out to be very big, and the following algorithm can be used to provide a rough method to estimate the entropy E.

Algorithm 6. (For Determining the Entropy):

Step 1. Order the n vehicles according to their a/b value (increasing order), get an array $\mathbb{c}$;
Step 2. Order the n vehicles according to their a/b^2 value (decreasing order), get an array $\mathbb{d}$;
Step 3. Set $m = 0$, $n_s = 0$, a null set $\mathcal{I}$ to record the inconsistency vehicles;
Step 4. Let i respectively be $v_1, v_2, \ldots, v_n$ of array $\mathbb{c}$,
then let j respectively be $v_1, v_2, \ldots, v_n$ of array $\mathbb{d}$.
If v_i of array $\mathbb{c}$ is the same vehicle with v_j of array $\mathbb{d}$,
then put v_i into set $\mathcal{I}$. Let $n_s = max\{|n - j - i + 1|, n_s\}$.
If $n - j - i > m$, set $m = m + 1$;
Step 5. Output m and n_s.
(The approximate complexity entropy is $\log_n m! \times (m+1)^{n_s - m}$.)

6 Numerical Experiment and Results

Until now, we have given the model of n-vehicle exploration problem, and analyzed its complexity. According to the complexity analysis, it is not easy to get the optimal solution unless taking an exponential time complexity. So we design a heuristic algorithm and its improved edition to get near optimal solutions. All of the heuristic algorithms we proposed are efficient, which means within time complexity $O(n^3)$. Many numerical examples are provided as follows and results' comparison is presented at last. The numerical experiment shows that the improved heuristic algorithm can get a very satisfied solution for each case of complexity. Moreover when we increase the scale n into 14, see Example 12, and the improved heuristic algorithm can still get an about 99.6% approximation ratio. (Actually we have tried some examples when n equals to 10, 14, 20 30, and run the heuristic algorithms. Because it needs enormous deal of computation

time to find the optimal solution for large scale problems, in this case, to testify the approximation degree of heuristic, we could only guess the optimal solution by using local search method.) All the numerical experiments are programmed in the Matlab plat, and the version 7.0.1 [10].

Table 3. Example 3

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	130	60	62	65	120	135	140	150	155	68
b	30	7	10	8	30	26	28	32	27	8

Table 4. Example 4

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	51	60	62	65	120	135	140	150	155	68
b	9	7	10	8	30	26	28	32	27	8

The optimal order of Example 3 is $v_5 \Rightarrow v_1 \Rightarrow v_8 \Rightarrow v_7 \Rightarrow v_6 \Rightarrow v_9 \Rightarrow v_3 \Rightarrow v_4 \Rightarrow v_{10} \Rightarrow v_2$. Example 3 satisfies the special case condition, which means that the entropy of Example 3 is 0. So we can directly use Algorithm 1 to get its optimal solution. The optimal order of Example 4 is $v_5 \Rightarrow v_8 \Rightarrow v_7 \Rightarrow v_6 \Rightarrow v_9 \Rightarrow v_1 \Rightarrow v_3 \Rightarrow v_4 \Rightarrow v_{10} \Rightarrow v_2$.

Table 5. Example 5

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	51	60	62	65	120	135	140	150	155	55
b	9	7	10	8	30	26	28	32	27	12

Table 6. Example 6

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	6	9	12	15	18	21	24	32	38	45
b	4	5	6	7	8	9	10	13	15	17

The optimal order of Example 5 is $v_5 \Rightarrow v_8 \Rightarrow v_7 \Rightarrow v_6 \Rightarrow v_9 \Rightarrow v_{10} \Rightarrow v_1 \Rightarrow v_3 \Rightarrow v_4 \Rightarrow v_2$. The optimal order of Example 6 is $v_1 \Rightarrow v_3 \Rightarrow v_9 \Rightarrow v_{10} \Rightarrow v_8 \Rightarrow v_7 \Rightarrow v_6 \Rightarrow v_5 \Rightarrow v_4 \Rightarrow v_2$.

Table 7. Example 7

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	12	23	11	26	16	31	19	21	44	25
b	2	4	3	5	3.4	5	4.3	7	7.8	4.5

Table 8. Example 8

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	65	120	135	140	150	155	55	100	122	60
b	8	30	26	28	32	27	12	10	11	4

The optimal order of Example 7 is $v_8 \Rightarrow v_3 \Rightarrow v_7 \Rightarrow v_5 \Rightarrow v_9 \Rightarrow v_4 \Rightarrow v_{10} \Rightarrow v_6 \Rightarrow v_2 \Rightarrow v_1$. The optimal order of Example 8 is $v_2 \Rightarrow v_5 \Rightarrow v_4 \Rightarrow v_3 \Rightarrow v_6 \Rightarrow v_7 \Rightarrow v_1 \Rightarrow v_8 \Rightarrow v_9 \Rightarrow v_{10}$.

Table 9. Example 9

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	32	42	55	65	72	80	83	84	95	100
b	4	4.8	5.5	7.8	11	9.6	6.7	15	8.6	16

Table 10. Example 10

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	70	76	84	70	70	79	87	71	74	72
b	7.5	8.6	9.6	6.4	7.2	9	10	7.8	7	8

The optimal order of Example 9 is $v_8 \Rightarrow v_{10} \Rightarrow v_5 \Rightarrow v_6 \Rightarrow v_4 \Rightarrow v_1 \Rightarrow v_2 \Rightarrow v_9 \Rightarrow v_3 \Rightarrow v_7$. The optimal order of Example 10 is $v_7 \Rightarrow v_3 \Rightarrow v_6 \Rightarrow v_2 \Rightarrow v_{10} \Rightarrow v_8 \Rightarrow v_1 \Rightarrow v_5 \Rightarrow v_9 \Rightarrow v_4$.

Table 11. Example 11

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
a	56	73	89	68	76	82	85	87	66	51
b	2	3	3.3	2.7	3.1	4	3.8	4.2	2.5	2.3

The optimal order of Example 11 is $v_6 \Rightarrow v_8 \Rightarrow v_7 \Rightarrow v_{10} \Rightarrow v_2 \Rightarrow v_5 \Rightarrow v_3 \Rightarrow v_4 \Rightarrow v_9 \Rightarrow v_1$.

Table 12. Example 12

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}
a	30	32	40	42	45	50	53	70	72	72	74	76	78	80
b	4	5.2	4.8	5.1	6	6.2	6.4	7.5	11	8	7	8.6	12	13

The optimal order of Example 12 is $v_{14} \Rightarrow v_{13} \Rightarrow v_9 \Rightarrow v_2 \Rightarrow v_5 \Rightarrow v_6 \Rightarrow v_7 \Rightarrow v_{12} \Rightarrow v_{10} \Rightarrow v_4 \Rightarrow v_8 \Rightarrow v_3 \Rightarrow v_{11} \Rightarrow v_1$.

Table 13 lists the simulated results of different examples. In Table 13, the entropy corresponding to different examples are also provided, and according to the entropy value, an algorithm is suggested for each example. This is a simple reflection of the our motivation to construct the real-time algorithm scheme.

Table 13. Results of simulated algorithms

	Exact algorithm $O(n!)$	Heuristic algorithm $O(n^2)$		Improved heuristic $O(n^3)$		Algorithm Scheme($Q = 3$)	
	S_{max}	S_{max}	ratio	S_{max}	ratio	entropy	suggest
Example 1	11.9466	11.7532	98.4%	11.8345	99.1%	0.6	Algorithm 3
Example 2	11.0236	10.9680	99.5%	11.0236	100%	0.8	Algorithm 3
Example 3	12.7697	12.4763	97.7%	12.7697	100%	0	Algorithm 1
Example 4	12.7230	12.3035	96.7%	12.4895	98.2%	0.6	Algorithm 3
Example 5	12.1240	11.8963	98.1%	11.9735	98.8%	2.7	Algorithm 3
Example 6	3.6033	3.5113	97.5%	3.5387	98.2%	6.6	Algorithm 5
Example 7	9.4293	9.3540	99.2%	9.3774	99.5%	4.1	Algorithm 5
Example 8	18.2772	18.2303	99.7%	18.2363	99.8%	1.3	Algorithm 3
Example 9	15.5472	14.7718	95.0%	15.4277	99.2%	3.2	Algorithm 5
Example 10	15.4766	15.3166	99.0%	15.4766	100%	0	Algorithm 1
Example 11	41.4520	40.9312	98.7%	41.2120	99.4%	4.9	Algorithm 5
Example 12	15.2481	14.8997	97.7%	15.1848	99.6%	8.7	Algorithm 5

(As has explained earlier, by guessing the optimal solution using local search algorithm, we also testify the heuristic when $n = 20, 30$. The approximation ratios of heuristic Algorithm 4 for $n = 20$ and $n = 30$ are both greater than 97%, and the approximation ratios of improved Algorithm 5 are both greater than 99%.)

7 Conclusion

Given an n-vehicle exploration problem, we can find the distribution of its complexity. Then we choose a polynomial time algorithm, which can get optimal or sub-optimal solution. Our choice mainly depends on the requirements of computational time and approximation ratio based on the work of complexity analysis. By using the structure of n-vehicle exploration problem, we provide a real-time algorithm scheme.

A convenient way to accomplish the real-time algorithm scheme when the problem's scale is enormous is adopting the heuristic algorithm. Although there are no obvious evidence to show that the heuristic algorithm will go well in any case, but at least it is an efficient algorithm and runs very well in the numerical experiment. Sometimes one can find specially crafted problem instances where the heuristic will in fact produce very bad results; however, these instances might never occur in practice because of their special structure. Besides according to our many numerical experiments, the heuristic algorithm performs very well.

References

1. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization Algorithms and Complexity. Prentice Hall Inc., Englewood Cliffs (1982)
2. Du, D.Z., Ko, K.-I., Wang, J.: Introduction to Computational. Higher Education Press (2002) (in Chinese)
3. Yu, G., Qi, X.T.: Disruption Management: Framework, Models and Applications. World Scientific Publishing, Co., Singapore (2004)
4. Heuristic Algorithm, http://en.wikipedia.org/wiki/Heuristic_(computer_science)
5. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2001)
6. Rui, Y.: A Travelling Problem and its Extended Research. Bulletin of Maths. 9, 44–45 (1999)
7. Li, X.Y., Cui, J.C.: Efficient Algorithm for Kind of Exploration Problem with N Vehicles. Journal of Systems Engineering 23(4), 444–448 (2008) (in Chinese)
8. Liang, L.L.: A Problem of Permutation Optimization. Journal of Guangxi University for Nationalities 12(4), 72–76 (2006) (in Chinese)
9. Wu, T.: Koimogorov Complexity Research Historical Review. Journal of Systemic Dialectics 12(1), 24–29 (2004) (in Chinese)
10. Wang, Z.L., Liu, M.: Master in Matlab 7. Publishing House of Electronics Industry (2006) (in Chinese)

Deterministically Estimating Data Stream Frequencies

Sumit Ganguly

Indian Institute of Technology, Kanpur, India
sganguly@cse.iitk.ac.in

Abstract. We consider updates to an n-dimensional frequency vector of a data stream, that is, the vector f is updated coordinate-wise by means of insertions or deletions in any arbitrary order. A fundamental problem in this model is to recall the vector approximately, that is to return an estimate $\hat{f}$ of f such that

$$|\hat{f}_i - f_i| < \epsilon \|f\|_p, \text{ for every } i = 1, 2, \ldots, n,$$

where ϵ is an accuracy parameter and p is the index of the ℓ_p norm used to calculate the norm $\|f\|_p$. This problem, denoted by $\text{ApproxFreq}_p(\epsilon)$, is fundamental in data stream processing and is used to solve a number of other problems, such as heavy hitters, approximating range queries and quantiles, approximate histograms, etc..

Suppressing poly-logarithmic factors in n and $\|f\|_1$, for $p = 1$ the problem is known to have $\tilde{\Theta}(1/\epsilon)$ randomized space complexity [2,4] and $\tilde{\Theta}(1/\epsilon^2)$ deterministic space complexity [6,7]. However, the deterministic space complexity of this problem for any value of $p > 1$ is not known. In this paper, we show that the deterministic space complexity of the problem $\text{ApproxFreq}_p(\epsilon)$ is $\tilde{\Theta}(n^{2-2/p}/\epsilon^2)$ for $1 < p < 2$, and $\Theta(n)$ for $p \geq 2$.

1 Introduction

In the data streaming model, computation is performed over a sequence of rapidly and continuously arriving data in an online fashion by maintaining a sub-linear space summary of the data. A data stream may be modeled as a sequence σ of updates of the form $(index, i, v)$, where, $index$ is the position of the update in the sequence, $i \in [n] = \{1, 2, \ldots, n\}$ and v is the update indicated by this record to the frequency f_i of i. The frequency vector $f(\sigma)$ of the stream σ is defined as:

$$f(\sigma) = \sum_{(index,i,v) \in \sigma} v \cdot e_i$$

where, $e_1, \ldots, e_n$ are the elementary n-dimensional unit vectors (i.e., e_i has 1 in position i and 0's elsewhere).

The problem of estimating the item frequencies of a data stream is to approximately recall the frequency vector of the stream. More precisely, the problem, denoted by $\text{ApproxFreq}_p(\epsilon)$, is to design a data stream processing algorithm

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 301–312, 2009.

that can return an n-dimensional vector f' satisfying $err_p(f', f(\sigma)) \leq \epsilon$, for $p \geq 1$, where,

$$err_p(f', f) = \frac{\|f' - f\|_\infty}{\|f\|_p} \ .$$

This problem is fundamental in data stream processing. Solutions to this problem are used to find approximate frequent items (also called heavy hitters) [4,5,12,13,15], approximate range queries and quantiles [9,4], and approximately v-optimal histograms [8,10].

Review of algorithms for ApproxFreq$_p(\epsilon)$. The problem ApproxFreq$_p(\epsilon)$ is widely studied for $p = 1$ and for $p = 2$. For $p = 1$ and for insert-only streams, the algorithm of [15,5,12] uses space $\Theta((1/\epsilon)\log m)$, where $m = \max_i f_i$. The algorithm works only for insert-only streams (i.e., no decrement updates) and has optimal $O(1)$ time complexity for processing each stream update. Other algorithms presented for this problem include the sticky sampling technique [14] that uses space $O((1/\epsilon)(\log n)(\log m))$.

For general streams allowing arbitrary insertions and deletions, the randomized algorithms Count-Min [4] and Countsketch [3] are applicable for solving the problems ApproxFreq$_1(\epsilon)$ and ApproxFreq$_2(\epsilon)$ respectively. These algorithms are randomized. The Count-Min uses space $O((1/\epsilon)(\log mn)(\log 1/\delta))$, where $1 - \delta$ is the confidence parameter of the randomized algorithm. The Countsketch solves ApproxFreq$_2(\epsilon)$ using space $O((1/\epsilon^2)(\log mn)(\log(1/\delta)))$. Both algorithms are space-optimal up to poly-logarithmic factors.

We now consider deterministic solutions to the problem ApproxFreq$_p(\epsilon)$ for general streams. Deterministic algorithms have certain advantages, as is exemplified by the following scenario. Consider a service provider that wishes to give a discount to all its customers whose business with the company is a certain significant fraction (say 0.01%) of its revenue. The scheme is supposedly continuous, namely, that if a customer becomes a highly-valued customer then s/he gets the benefit immediately and vice-versa. For economy of space and time, the decision about whether a customer should be given a discount is done by a stream processing algorithm of the kind discussed earlier. If the algorithm is randomized, there is a chance, albeit small, that a highly valued customer is misclassified, resulting in an unhappy customer. Deterministic algorithms do not use random coin tosses and cannot lead to such grievances.

The algorithms in [5,12,15] are deterministic, however, these algorithms are applicable only for insert-only streams. The CR-precis algorithm [7] is a deterministic algorithm for ApproxFreq$_1(\epsilon)$ for general streams with insertions and deletions and uses space $O(\epsilon^{-2}(\log m \log n)^2)$ bits, where, $m = \|f(\sigma)\|_\infty$. The work in [6] shows that any total, deterministic algorithm for solving the ApproxFreq$_1(\epsilon)$ problem requires $\Omega\left((\log m)/\epsilon^2\right)$ bits. Thus, the deterministic space complexity of ApproxFreq$_p(\epsilon)$ is resolved to $\tilde{\Theta}(\epsilon^{-2})$ for $p = 1$, where, $\tilde{\Theta}$ notation suppresses poly-logarithmic factors in n and m.

No results are known so far for space bounds for deterministic algorithms for ApproxFreq$_p(\epsilon)$, for $p > 1$. The problem is fundamental, for instance, in the

randomized case, the CountSketch algorithm solves ApproxFreq$_2(\epsilon)$ using space $\tilde{\Theta}(\epsilon^{-2})$, and this important result is the basis for a number of space-optimal algorithms for estimating frequency moments [11,1], approximate histograms [8], etc.. Therefore, understanding the space complexity of a deterministic solution to the problem ApproxFreq$_p(\epsilon)$ is of basic importance.

Contributions. We present space lower and upper bounds for deterministic algorithms for ApproxFreq$_p(\epsilon)$ for $p \geq 1$. We show that for $p \geq 2$, solving ApproxFreq$_p(\epsilon)$ requires $\Omega(n)$ space. For $p \in [1, 2)$, the space requirement is $\Omega(n^{2-2/p}(\log m)/\epsilon^2)$. Finally, we show that the upper bounds are matched by suitably modifying the CR-precis algorithm. The formal statement of our result is as follows.

Theorem 1. *For $\epsilon \leq 1/8$ and $p \geq 2$, any deterministic algorithm that solves* ApproxFreq$_p(\epsilon)$ *over general data streams requires space $\Omega(n \log m)$. For $1 \leq p < 2$ and $\epsilon \geq 0.5n^{1/p-1/2}$, any deterministic algorithm that solves* ApproxFreq$_p(\epsilon)$ *over general data streams requires space $\Omega(\epsilon^{-2}\, n^{2-2/p} \log m)$. Further, these lower bounds can be matched by algorithms up to poly-logarithmic factors.*

Organization. The remainder of the paper is organized as follows. Section 2 reviews work on stream automaton, which is is used to prove the lower bounds. Sections 3 and 4 presents lower and upper bounds respectively, for the space complexity of streaming algorithms for ApproxFreq$_p(\epsilon)$.

2 Review: Stream Automaton

We model a general stream over the domain $[n] = \{1, 2, \ldots, n\}$ as a sequence of individual records of the form $(index, a)$, where, *index* represents the position of this record in the sequence and a belongs to the set $\Sigma = \Sigma_n = \{e_1, -e_1, \ldots, e_n, -e_n\}$. Here, e_i refers to the n-dimensional elementary vector $(0, \ldots, 0, 1\ (i\text{th position}), 0 \ldots, 0)$. The *frequency* of a data stream σ, denoted by $f(\sigma)$ is defined as the sum of the elementary vectors in the sequence. That is,

$$f(\sigma) = \sum_{(index,v)\in\sigma} v \ .$$

The concatenation of two streams σ and τ is denoted by $\sigma \circ \tau$. The size of a data stream σ is defined as follows.

$$|\sigma| = \max_{\sigma' \text{ sub-sequence of } \sigma} \|f(\sigma')\|_\infty \ .$$

A deterministic **stream automaton** [6] is an abstraction for deterministic algorithms for processing data streams. It is defined as a two tape Turing machine, where the first tape is a one-way (unidirectional) input tape that contains the sequence σ of updates that constitutes the stream. Each update is a member of Σ, that is, it is an elementary vector or its inverse, e_i or $-e_i$. The second tape

is a (bidirectional) two way work-tape. A configuration of a stream automaton is modeled as a triple (q, h, w), where, q is a state of the finite control, h is the current head position of the work-tape and w is the content of the work-tape. The set of configurations of a stream automaton A that are reachable from the initial configuration o on some input stream is denoted as $C(A)$. The set of configurations of an automaton A that is reachable from the origin o for some input stream σ with $|\sigma| \leq m$ is denoted by $C_m(A)$. A stream automaton may be viewed as a tuple $(n, C, o, \oplus, \psi)$, where, $\oplus : C \times \Sigma \rightarrow C$ is the configuration transition function and $\psi : C \rightarrow O$ is the output function. The transition function, written as $s \oplus t$, where, $s \in C$ and t is a stream update, denotes the configuration of the algorithm after it starts from configuration s and processes the stream record t. We generally write the transition function in infix notation. The notation is generalized so that $a \oplus \sigma$ denotes the current configuration of the automaton starting from configuration and processing the records of the stream σ in a left to right sequence, that is,

$$s \oplus (\sigma \circ \tau) \stackrel{\text{def}}{=} (s \oplus \sigma) \oplus \tau \ .$$

After processing the input stream σ, the stream automaton prints the output

$$\text{output}_A(\sigma) = \psi(o \oplus \sigma) \ .$$

The automaton A is said to have space function $\text{Space}(A, m)$, provided, for all input streams σ such that $|\sigma| \leq m$, the number of cells used on the work-tape during the processing of input is bounded above by $\text{Space}(A, m)$. It is said to have communication function $\text{Comm}(A, m) = \log|C_m(A)|$. The communication function can be viewed as a lower bound of the *effective* space usage of an automaton. The space or communication function does not include the space used by the automaton A to print its output. This allows the automaton to print outputs of size $\Omega(\text{Space}(A, m))$.

The approximate computation of a function $g : \mathbb{Z}^n \rightarrow O$ of the frequency vector $g(f(\sigma))$ is specified by a binary approximation predicate $\textsc{Approx} : E \times E \rightarrow \{\textsc{true}, \textsc{false}\}$ such that an estimate $\hat{a} \in O$ is considered an acceptable approximation to the true value $a \in O$ provided $\textsc{Approx}(\hat{a}, a) = \textsc{true}$ and is not considered to be an acceptable approximation if $\textsc{Approx}(\hat{a}, a) = \textsc{false}$. A stream automaton A is said to compute a function $g : \mathbb{Z}^n \rightarrow O$ of the frequency vector $f(\sigma)$ of its input stream σ with respect to the approximation predicate Approx, provided

$$\textsc{Approx}(\psi(\sigma), g(f(\sigma))) = \textsc{true}$$

for all feasible input streams σ. A stream automaton is said to be *total* if the feasible input set is the set of all input streams over the domain $[n]$ and is said to be *partial* otherwise. The class STRfreq represents data streaming algorithms for computing approximation of (partial or total) functions of the frequency vector of the input stream. The notation $\mathbb{Z}_{2m+1}$ denotes the set of integers $\{-m, \ldots, 0, \ldots, m\}$.

A stream automaton is said to be *path independent* if for any reachable configuration $a \in C(A)$, the configuration obtained by starting from a and processing

any input stream σ is dependent only on a and $f(\sigma)$. That is, $a \oplus \sigma$ depends only on a and $f(\sigma)$. The *kernel* of a path independent automaton is defined as

$$K(A) = \{a \in C(A) \mid \exists \sigma \text{ s.t. } o \oplus \sigma = o \text{ and } f(\sigma) = 0\} \ .$$

It is shown in [6] that the kernel of a path independent automaton is a sub-module of $\mathbb{Z}^n$. A stream automaton is said to be *free* if it is path-independent and its kernel is a free module. We present the basic theorem of stream automaton.

Theorem 2 ([6]). *For every stream automaton $A = (n, C_A, o_A, \oplus_A, \psi_A)$, there exists a path-independent stream automaton $B = (n, C_B, o_B, \oplus_B, \psi_B)$ such that the following holds.*

(1) *For any* Approx *predicate and any total function $g : \mathbb{Z}^n \to O$,* $\textsc{Approx}(\psi_B(\sigma), g(\sigma))$ *holds if* $\textsc{Approx}(\psi_A(\sigma), g(\sigma))$ *holds.*
(2) $Comm(B, m) \leq Comm(A, m)$.
(3) *There exists a sub-module $M \subset \mathbb{Z}^n$ and an isomorphic map $\varphi : C_B \to \mathbb{Z}^n/M$ where, $(\mathbb{Z}^n/M, \bigoplus)$ is viewed as a module with binary addition operation $\bigoplus$, such that for any stream σ,*

$$\varphi(a \oplus \sigma) = \varphi(a) \bigoplus [f(\sigma)]$$

where, $x \mapsto [x]$ is the canonical homomorphism from $\mathbb{Z}^n$ to $\mathbb{Z}^n/M$ (that is, $[x]$ is the unique coset of M to which x belongs).
(4) $Comm(B, m) = O((n - \dim M) \log m)$, *where,* $\dim M$ *is the dimension of* M.

Conversely, given any sub-module $M \subset \mathbb{Z}^n$, a stream automaton $A = (n, C_A, o_a, \oplus_A, \psi_A)$ can be constructed such that there is an isomorphic map $\varphi : C_A \to \mathbb{Z}^n/M$ such that for any stream σ,

$$\varphi(a \oplus \sigma) = \varphi(a) \bigoplus [f(\sigma)] \ .$$

where, $\bigoplus$ is the addition operation of $\mathbb{Z}^n/M$, and

$$\begin{aligned} Comm(A, m) &= \log \left\lceil \left| \{[x] : x \in \mathbb{Z}^n_{2m+1}\} \right| \right\rceil \\ &= \Theta((n - \dim M) \log m) \end{aligned}$$

□

3 Lower Bounds for $\textsc{ApproxFreq}_p$

In this section, we establish deterministic space lower bounds for $\textsc{ApproxFreq}_p(\epsilon)$.

Theorem 2 enables us to restrict attention to path independent automata in general, for all frequency-dependent computation. Lemma 1 further allows us to restrict our attention to free automata, for the problem of $\textsc{ApproxFreq}_p(\epsilon)$, while incurring a factor of 4 relaxation.

Lemma 1. *Suppose that A is a path independent stream automaton for solving* $\textsc{ApproxFreq}_p(\epsilon)$ *over domain $[n]$ and has kernel M. Then, there exists a free automaton B with kernel M' such that $M' \supset M$, $\mathbb{Z}^n/M'$ is free, and $err_p(\min_p(x + M'), x) \leq 4\epsilon$.*

The proof is similar in spirit to a corresponding Lemma in [6] and is given in the Appendix for completeness.

Consider a free automaton A over domain $[n]$ with kernel M that is a free module and let M^e denote the unique smallest dimension subspace of $\mathbb{R}^n$ that contains M. Let V be a $n \times k$ matrix whose columns are orthonormal and form a basis of $\mathbb{R}^n/M^e$. Let U denote an orthonormal basis of M^e, so that $[V\ U]$ forms an orthonormal basis of $\mathbb{R}^n$. For $x \in \mathbb{R}^n$, the coset $x + M^e = \{y : V^T y = V^T x\}$. For a given coset $x + M^e$, let $\bar{x}$ denote the element $y \in x + M^e$ with the smallest value of $\|y\|_2$. Clearly, $\bar{x}$ is the element in $x + M^e$ whose coordinates along U are all 0. Therefore,

$$\bar{x} = [V\ U] \begin{bmatrix} V^T x \\ 0 \end{bmatrix} = VV^T x \ . \tag{1}$$

Lemma 2. *If* $err_2(\bar{x}, x) \leq \epsilon$ *for all* x, *then,* $\text{rank}(V) \geq n(1 - \epsilon)$.

Proof. Let $\text{rank}(V) = k$. The condition $err_2(\bar{x}, x) \leq \epsilon$ is equivalent to

$$\|(VV^T - I)x\|_\infty \leq \epsilon \|x\|_2 \ .$$

In particular, this condition holds for the standard unit vectors $x = e_1, e_2, \ldots, e_n$ respectively. Thus, $\|VV^T e_i - e_i\|_\infty \leq \epsilon$, for $i = 1, 2, \ldots, n$. This implies that $|(VV^T)_{ii} - 1| \leq \epsilon$. Thus,

$$\text{trace}(VV^T) \geq n(1 - \epsilon) \ .$$

Since V has rank k and has k orthonormal columns, the eigenvalues of VV^T are 1 with multiplicity k and 0 with multiplicity $n - k$. Thus, $\text{trace}(VV^T) = k$. Therefore, $n(1 - \epsilon) \leq \text{trace}(VV^T) = k$. □

The lower bound proof for $1 \leq p < 2$ is slightly more complicated. We first prove the following lemma.

Lemma 3. *For any orthonormal basis* $[VU]$ *of* $\mathbb{R}^n$ *such that* $\text{rank}(V) = k$ *and for any* $1 < p < 2$, *there exists* $i \in [n]$ *such that* $\|VV^T e_i\|_2 \leq 2k/n$ *and* $\|VV^T e_i\|_p \leq 2n^{1/p-1}\sqrt{k}$.

Proof. Since, V has orthonormal columns

$$\|VV^T e_i\|_2^2 = \|V^T e_i\|_2^2 = (VV^T e_i)_i \ . \tag{2}$$

Therefore,

$$\text{trace}(VV^T) = \sum_{i=1}^{n} (VV^T e_i)_i = \sum_{i=1}^{n} \|VV^T e_i\|_2^2 \tag{3}$$

The trace of VV^T is the sum of the eigenvalues of VV^T. Suppose $\text{rank}(V) = k$. Since, V has orthonormal columns and has rank k, VV^T has eigenvalue 1 with

multiplicity k and eigenvalue 0 with multiplicity $n-k$. Thus, $\text{trace}(VV^T) = k$. By (3)

$$k = \text{trace}(VV^T) = \sum_{i=1}^{n} \|VV^T e_i\|_2^2 \ . \tag{4}$$

Further, since, $\|x\|_p \leq \|x\|_2 \cdot n^{1/p-1/2}$

$$\begin{aligned}\sum_{i=1}^{n} \|VV^T e_i\|_p &\leq \sum_{i=1}^{n} \|VV^T e_i\|_2 (n^{1/p-1/2}) \\ &\leq \sqrt{n} \left(\sum_{i=1}^{n} \|VV^T e_i\|_2^2 \right)^{1/2} n^{1/p-1/2} \\ &\quad \{\text{by Cauchy-Schwartz inequality }\} \\ &= n^{1/p}\sqrt{k} \qquad \text{by (4)} \ . \end{aligned} \tag{5}$$

Let

$$\begin{aligned} J &= \{i : \|VV^T e_i\|_2^2 \leq 2k/n\}, \text{ and} \\ K &= \{i : \|VV^T e_i\|_p \leq 2n^{1/p-1}\sqrt{k}\} \ . \end{aligned}$$

Therefore, by (4) and (5), $|J| > \frac{n}{2}$ and $|K| > \frac{n}{2}$. Hence, $J \cap K \neq \phi$, that is, there exists i such that

$$\|VV^T e_i\|_2 \leq (2k/n)^{1/2} \text{ and } \|VV^T e_i\|_p \leq 2n^{1/p-1}\sqrt{k} \ .$$

The lemma is then proved. □

Lemma 4. *Let A be a free automaton that solves the problem* ApproxFreq$_p(\epsilon)$ *over the domain $[n]$ for some $1 \leq p < 2$ and has kernel M. Let M^e be the smallest dimension subspace of $\mathbb{R}^n$ containing M. Let V, U be a collection of vectors that forms an orthonormal basis for $\mathbb{R}^n$ such that U spans M^e and V spans $\mathbb{R}^n/M^e$. Then, for $\epsilon \geq 2n^{1/2-1/p}$,* $\text{rank}(V) \geq \frac{n^{2-2/p}}{16\epsilon^2}$.

Proof. By Lemma 3, there exists i such that

$$\begin{aligned} \|VV^T e_i\|_2^2 &\leq \frac{2k}{n} \text{ and} \\ \|VV^T e_i\|_p &\leq 2n^{1/p-1}\sqrt{k} \ . \end{aligned} \tag{6}$$

Since, $e_i - VV^T e_i = UU^T e_i \in M^e$, therefore,

$$\begin{aligned} \epsilon &\geq err_p(e_i - VV^T e_i, 0) \\ &= \frac{\|e_i - VV^T e_i\|_\infty}{\|e_i - VV^T e_i\|_p} \ . \end{aligned}$$

Therefore,

$$\|e_i - VV^T e_i\|_\infty \leq \epsilon \|VV^T e_i - e_i\|_p \ . \tag{7}$$

By (2),

$$(VV^T e_i)_i = \|VV^T e_i\|_2^2 \leq \frac{2k}{n} \ .$$

Therefore,

$$\begin{aligned}\|e_i - VV^T e_i\|_\infty &\geq |(e_i - VV^T e_i)_i| = 1 - \|VV^T e_i\|_2^2 \\ &\geq 1 - \frac{2k}{n}, \quad \text{by (6)}.\end{aligned}$$

Substituting in (7),

$$\begin{aligned}1 - \frac{2k}{n} &\leq \|e_i - VV^T e_i\|_\infty \\ &\leq \epsilon \|VV^T e_i - e_i\|_p \\ &\leq \epsilon\big(\|VV^T e_i\|_p + 1\big) \\ &\leq \epsilon(2n^{1/p-1}\sqrt{k} + 1)\end{aligned}$$

where, the second to last inequality follows from using triangle inequality over pth norms and the last inequality follows from (6). Simplifying, we obtain that

$$k \geq \frac{n^{2-2/p}}{16\epsilon^2}, \text{ provided, } \epsilon \geq 2n^{1/2-1/p} \ .$$

The lemma is then proved. □

We recall that as shown in [6], $\text{Comm}(A, m) \geq \text{rank}(V)\log(2m+1)$.

Proof (Of Theorem 1). We first consider the case $p = 2$ and $p > 2$. By Theorem 2, it follows that corresponding to any stream automaton A_n, there exists a path independent stream automaton B_n that is an output restriction of A_n and such that $\text{Comm}(B_n, m) \leq \text{Comm}(A_n, m)$. By Lemma 1, it follows that if B_n solves $\text{ApproxFreq}_p(\epsilon)$, then, there exists a free automaton C_n that solves $\text{ApproxFreq}_p(4\epsilon)$. Thus, by Theorem 2, it follows that if B_n solves $\text{ApproxFreq}_2(\epsilon)$ for $4\epsilon \leq 1$, then,

$$\text{Comm}(A_n, m) \geq \text{Comm}(B_n, m) \geq \text{Comm}(C_n, m) \geq \text{rank}(V_{C_n} \log m$$

and

$$\text{rank}(V_{C_n}) \geq n(1 - 4\epsilon)\log(2m+1), \text{ by Lemma 2} \ .$$

Here V_{C_n} is the vector space $\mathbb{R}^n / M^e(C_n)$, where, $M^e(C_n)$ is the kernel of C_n.

Further, for $p > 2$, $\|f\|_p \leq \|f\|_2$, for any $f \in \mathbb{R}^n$. Therefore, $err_p(\hat{f}, f) \leq \epsilon$ implies that $err_2(\hat{f}, f) \leq \epsilon$. Thus, the space lower bound for err_2 as given by Lemma 2 holds for err_p, for any $p > 2$.

By Lemma 4, it follows that if B_n solves $\text{ApproxFreq}_p(\epsilon)$, for $4\epsilon \geq 2n^{1/2-1/p}$, then,

$$\begin{aligned}\text{Comm}(A_n, m) &\geq \text{Comm}(B_n, m) \geq \text{Comm}(C_n, m) \geq \text{rank}(V_{C_n}) \log m \\ &\geq \frac{n^{2-2/p}}{64\epsilon^2} \log m \ .\end{aligned}$$

Finally, we note that for any stream automaton A_n, $\mathrm{Comm}(A_n, m)$ is a lower bound on the effective space usage $\mathrm{Space}(A_n, m)$.

This proves the lower bound assertion of Theorem 1. □

4 Upper Bound

Lemma 5 presents a (nearly) matching upper bound for the $\textsc{ApproxFreq}_p(\epsilon)$ problem, for $1 \leq p < 2$.

Lemma 5. *For any* $1 < p < 2$ *and* $1 > \epsilon > \frac{1}{\sqrt{n}}$, *there exists a total stream algorithm for solving* $\textsc{ApproxFreq}_p(\epsilon)$ *using space* $O(\epsilon^{-2} n^{2-2/p} (\log \|f(\sigma)_1\|) (p/(p-1+p(\log(1/\epsilon)/\log n)))^2)$.

Proof. By a standard identity between norms, for any vector $f \in \mathbb{R}^n$, $\|f\|_1 \leq n^{1-1/p} \|f\|_p$. Therefore,

$$err_1(\hat{f}, f) \leq \frac{\epsilon}{n^{1-1/p}} \text{ implies } err_p(\hat{f}, f) \leq \epsilon \ .$$

So let $\epsilon' = \epsilon / n^{1-1/p}$, and use the CR-precis algorithm with accuracy parameter ϵ'. This requires space

$$O((\epsilon')^{-2} (\log \|f(\sigma)\|_1)(\log^2 n)/(\log^2(1/\epsilon')) \ .$$

Substituting the value of ϵ', we obtain the statement of the lemma. □

The statement of the lemma is equivalent to the assertion of Theorem 1 for upper bounds. This completes the proof of Theorem 1.

References

1. Bhuvanagiri, L., Ganguly, S., Kesh, D., Saha, C.: Simpler Algorithm for Estimating Frequency Moments of Data Streams. In: Proceedings of ACM Symposium on Discrete Algorithms (SODA), pp. 708–713 (2006)
2. Bose, P., Kranakis, E., Morin, P., Tang, Y.: Bounds for Frequency Estimation of Packet Streams. In: Sibeyn, J.F. (ed.) Proceedings of the 10th Internaltional Colloquium on Structural Information Complexity, Informatics 17 Carleton Scientific, pp. 33–42 (2003)
3. Charikar, M., Chen, K., Farach-Colton, M.: Finding Frequent Items in Data Streams. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002)
4. Cormode, G., Muthukrishnan, S.: An Improved Data Stream Summary: The Count-Min Sketch and its Applications. J. Algorithms 55(1), 58–75 (2005)
5. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency Estimation of Internet Packet Streams with Limited Space. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 348–360. Springer, Heidelberg (2002)

6. Ganguly, S.: Lower bounds for Frequency Estimation over Data Streams. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) Computer Science – Theory and Applications. LNCS, vol. 5010, pp. 204–215. Springer, Heidelberg (2008)
7. Ganguly, S., Majumder, A.: CR-precis: A Deterministic Summary Structure for Update Streams. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 48–59. Springer, Heidelberg (2007)
8. Gilbert, A., Guha, S., Indyk, P., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Fast Small-space Algorithms for Approximate Histogram Maintenance. In: Proceedings of ACM STOC, pp. 152–161 (2002)
9. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.J.: Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries. In: Proceedings of VLDB, pp. 79–88 (2001)
10. Guha, S., Indyk, P., Muthukrishnan, S., Strauss, M.: Histogramming Data Streams with Fast Per-Item Processing. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 681–692. Springer, Heidelberg (2002)
11. Indyk, P., Woodruff, D.: Optimal Approximations of the Frequency Moments. In: Proceedings of ACM Symposium on Theory of Computing (STOC), Baltimore, Maryland, USA, pp. 202–298 (2005)
12. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A Simple Algorithm for Finding Frequent Elements in Streams and Bags. ACM Trans. Data. Syst. 28(1), 51–55 (2003)
13. Lee, L.K., Ting, H.F.: A Simpler and More Efficient Deterministic Scheme for Finding Frequent Items over Sliding Windows. In: Proceedings of ACM International Symposium on Principles of Database Systems (PODS), pp. 263–272 (2006)
14. Manku, G., Motwani, R.: Approximate Frequency Counts over Data Streams. In: Proceedings of VLDB, pp. 346–357 (2002)
15. Misra, J., Gries, D.: Finding Repeated Elements. Sci. Comput. Programm. 2, 143–152 (1982)

A Proofs

Let M be the kernel of A_n and let M' be defined as follows.

$$M' = \{x \mid \exists a \in \mathbb{Z}, ax \in M\} \tag{8}$$

It follows that M' is torsion-free.

Fact 3. *Let $b_1, b_2, \ldots, b_r$ be a basis of M'. Then, $\exists\ \alpha_1, \ldots, \alpha_r \in \mathbb{Z} - \{0\}$ such that $\alpha_1 b_1, \ldots, \alpha_r b_r$ is a basis for M. Hence, $M^e = (M')^e$.*

Proof (Of Fact 3). It follows from standard algebra that the basis of M is of the form $\alpha_1 b_1, \ldots, \alpha_r b_r$. It remains to be shown that the α_i's are non-zero. Suppose that $\alpha_1 = 0$. For any $a \in \mathbb{Z}$, $a \neq 0$, suppose $ax \in M$ and $x \in M'$. Then, x has a unique representation as $x = \sum_{j=1}^{r} x_j b_j$. Thus, $ax = \sum_{j=1}^{r} (ax_j) b_j \in M$ and has the same representation in the basis $\{\alpha_j b_j\}_{j=1,\ldots,n}$. Therefore, $ax_1 = 0$ or $x_1 = 0$ for all $x \in M'$, which is a contradiction.

Let $\{b_1, b_2, \ldots, b_r\}$ be a basis for M'. Then, by the above paragraph, there exist non-zero elements $\alpha_1, \ldots, \alpha_r$ such that $\{\alpha_1 b_1, \alpha_2 b_2, \ldots, \alpha_r b_r\}$ is a basis for M. Therefore, over reals, $(b_1, \ldots, b_r) = (\alpha_1 b_1, \ldots, \alpha_r b_r)$. Thus, $M^e = (M')^e$. □

Lemma 6. *Let M be a sub-module of $\mathbb{Z}^n$. (1) if there exists h_p such that $err_p(h_p, M) \leq \epsilon$, then, $err_p(0, M) \leq \epsilon$, and, (2) if $err_p(0, M) \leq \epsilon$ then $err_p(0, M^e) \leq \epsilon$.*

Proof. Part (1). For any $y_i \in \mathbb{Z}$,

$$\max(|(h_p)_i - y_i|, |(h_p)_i + y_i|) \geq |y_i|.$$

Therefore,

$$\max(\|h_p - y\|_\infty, \|h_p + y\|_\infty) \geq \|y\|_\infty \ .$$

Let $y \in M$. Since, M is a module, $-y \in M$. Thus,

$$\begin{aligned} err_p(0, y) = err_p(0, -y) &= \frac{\|y\|_\infty}{\|y\|_p} \\ &\leq \frac{1}{\|y\|_p} \max(\|h_p - y\|_\infty, \|h_p + y\|_\infty) \\ &= \max(err_p(h_p, y), err_p(h_p, -y)) \leq \epsilon \end{aligned}$$

Part 2. Let $z \in M^e$. Let $b_1, b_2, \ldots, b_r$ be a basis of the free module M. For $t > 0$, let tz be expressed uniquely as $tz = \alpha_1 b_1 + \ldots + \alpha_r b_r$, where, α_i's belong to $\mathbb{R}$. Consider the vertices of the parallelopiped P_{tz} whose sides are $b_1, b_2, \ldots, b_r$ and that encloses tz.

$$\begin{aligned} P_{tz} = [\alpha_1]b_1 + [\alpha_2]b_2 + \ldots + [\alpha_n]b_n \\ + \{\beta_1 b_1 + \beta_2 b_2 + \ldots + \beta_r b_r \mid \beta_j \in \{0, 1\}, j = 1, 2, \ldots, r\} \end{aligned}$$

where, $[\alpha]$ denotes the largest integer smaller than or equal to α. Since, ℓ_∞ is a convex function $\|tz\|_\infty \leq \|y\|_\infty$ for some $y \in P_{tz}$. Let $y = \sum_{j=1}^{r} \beta_j b_j$, for $\beta_j \in \{0, 1\}$, $j = 1, 2, \ldots, r$.

$$\|y - tz\|_1 = \|\sum_{j=1}^{r} (\beta_j - [\alpha_j])b_j\|_1 \leq \sum_{j=1}^{r} \|(\beta_j - [\alpha_j])b_j\|_1 \leq \sum_{j=1}^{r} \|b_j\|_1$$

$$\text{or,} \quad \|tz\|_1 \geq \|y\|_1 - \sum_{j=1}^{r} \|b_j\|_1$$

Therefore,

$$\begin{aligned} err_p(0, tz) = \frac{\|tz\|_\infty}{\|tz\|_1} &\leq \frac{\|y\|_\infty}{\|y\|_1 - \sum_{j=1}^{r} \|b_j\|_1} \\ &\leq \left(\frac{\|y\|_1}{\|y\|_\infty} - \frac{\sum_{j=1}^{r} \|b_j\|_1}{\|y\|_\infty} \right)^{-1} \leq \left(\frac{1}{\epsilon} - \frac{\sum_{j=1}^{r} \|b_j\|_1}{\|y\|_\infty} \right)^{-1} \end{aligned}$$

where, the last step follows from the assumption that $y \in M$ and therefore, $err_p(0, y) = \frac{\|y\|_\infty}{\|y\|_1} \leq \epsilon$. The ratio $\frac{\sum_{j=1}^{r} \|b_j\|_1}{\|y\|_\infty}$ can be made arbitrarily small by choosing t to be arbitrarily large. Thus, $\lim_{t \to \infty} err_p(0, tz) \leq \epsilon$. Since, $err_p(0, tz) = \frac{\|tz\|_\infty}{\|tz\|_1} = \frac{\|z\|_\infty}{\|z\|_1} = err_p(0, z)$, for all t, we have, $err_p(0, z) \leq \epsilon$. □

Proof (Of Lemma 7.). By construction, M' is the smallest module that contains M as a sub-module and M' is free. This also implies that $\mathbb{Z}^n/M'$ is free. For $x \in \mathbb{Z}^n$, define

$$h_p(x + M') = \min_{\ell_p}(x + M') \ .$$

That is, $h_p(x + M')$ is the element with the smallest ℓ_p norm among all vectors in $x + M'$.

Let $y \in x + M'$. Then, $y \in x_p + M$ for some x_p. Let $\hat{y} = \text{output}_A(x_p + M)$ denote the output of A for an input stream with frequency in $x_p + M$ (they all return the same value, since, A is path independent and has kernel M) and let $y'_p = \min_{\ell_p}(x_p + M)$. Let h_p denote $h_p(x + M')$ and let $\hat{h} = \text{output}_A(h_p + M)$. Therefore,

$$\begin{aligned} err(h_p, y) &= \frac{\|y - h_p\|_\infty}{\|y\|_p} \\ &\leq \frac{\|y - \hat{y}\|_\infty}{\|y\|_p} + \frac{\|\hat{y} - y'_p\|_\infty}{\|y\|_p} + \frac{\|y'_p - h_p\|_\infty}{\|y\|_p} \end{aligned} \tag{9}$$

The first and the second terms above are bounded by ϵ as follows. The first term $\frac{\|y-\hat{y}\|_\infty}{\|y\|_p} = err_p(\hat{y}, y) \leq \epsilon$, since, $y \in x_p + M$ and $\hat{y}$ is the estimate returned by A_n for this coset. The second term

$$\frac{\|\hat{y} - y'_p\|_\infty}{\|y\|_p} \leq \frac{\|\hat{y} - y'_p\|_\infty}{\|y'_p\|_p} = err(\hat{y}, y'_p) \leq \epsilon$$

since, $\|y'_p\|_p \leq \|y\|_p$ and y'_p lies in the coset $x_p + M$. The third term in (9) can be rewritten as follows. Since, M' is a free module, $y'_p - h_p \in M'$ and $M' \subset M^e$. Therefore,

$$\begin{aligned} &\frac{\|y'_p - h_p\|_\infty}{\|y\|_p} \\ &\leq \frac{\|y'_p - h_p\|_\infty}{\|y'_p - h\|_p} \cdot \frac{\|y'_p - h_p\|_p}{\|y'_p\|_p}, \qquad \text{since, } \|y'_p\|_p \leq \|y\|_p \\ &\leq \epsilon \cdot \frac{\|y'_p\|_p + \|h_p\|_p}{\|y'_p\|_p} \quad \text{by Lemma 6 and by triangle inequality} \\ &\leq 2\epsilon, \qquad \text{since, } \|h_p\|_p \leq \|y'_p\|_p \end{aligned}$$

By (9), $err(h, y) \leq \epsilon + \epsilon + 2\epsilon = 4\epsilon$. The automaton B_n with kernel M' is constructed as in Theorem 2. □

Positive Influence Dominating Set in Online Social Networks

Feng Wang, Erika Camacho, and Kuai Xu

Mathematical and Natural Sciences, Arizona State University
P.O. Box 37100, Phoenix, AZ 85069, USA
{fwang25,erika.camacho,kxu01}@asu.edu

Abstract. Online social network has developed significantly in recent years as a medium of communicating, sharing and disseminating information and spreading influence. Most of current research has been on understanding the property of online social network and utilizing it to spread information and ideas. In this paper, we explored the problem of how to utilize online social networks to help alleviate social problems in the physical world, for example, the drinking, smoking, and drug related problems. We proposed a *Positive Influence Dominating Set (PIDS)* selection algorithm and analyzed its effect on a real online social network data set through simulations. By comparing the size and the average positive degree of PIDS with those of a 1-dominating set, we found that by strategically choosing 26% more people into the PIDS to participate in the intervention program, the average positive degree increases by approximately 3.3 times. In terms of the application, this result implies that by moderately increasing the participation related cost, the probability of positive influencing the whole community through the intervention program is significantly higher. We also discovered that a power law graph has empirically larger dominating sets (both the PIDS and 1-dominating set) than a random graph does.

1 Introduction

Online social network is a network composed of individuals who share the same interest and purpose which provides a powerful medium of communicating, sharing and disseminating information, and spreading influence beyond the traditional social interactions within a traditional social network setting. Online social network has developed significantly in recent years. For example, online social network sites like Facebook, MySpace are among the most popular sites on the Internet; online social networks have also raise special interest among commercial businesses, medical and pharmaceutical companies as a channel to influence the opinion of their customers; even police has utilize the information in online social network sites to track down crimes.

Some research has been done to understand the properties of online social networks [10,11,2] and how to effectively utilize social networks to spread ideas and information within a group [8]. In this paper, we explore the problem of

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 313–321, 2009.

how to utilize online social networks to help alleviate social problems in the physical world. Some examples of these type of problems include the drinking, smoking, and drug related problems. These social issues are very intricate and complex problems that require a system-level approach where the dynamics of positive and negative influence resulting from individual-to-individual and from individual-to-group interactions as well as the evolving status of individuals can be fully captured. In a social setting, people can have both positive and negative impact on each other and a person can take and move among different roles since they are affected by their peers. For example, within the context of drinking problem, a person can be an abstainer, or a binge drinker. An abstainer has positive impact on his direct friends (called neighbors) but he might turn into a binge drinker and have negative impact on his neighbors if many of his friends are binge drinkers.

Alcohol intervention strategies and programs that consist of disseminated education and therapy via mail, Internet, or face-to-face interviews such as *motivational feedback* [13] are important tools to help combat some of the social problems within today's society. In an ideal world to truly alleviate the main source of the drinking problem, one must educate as many binge drinkers as possible. This will prevent an abstainer who might adopt the negative influence of his close binge drinker friends from eventually turning into a binge drinker. If too few people are selected to participate in an intervention program, there is a high likelihood that the positive effect of the intervention will be overrun by the negative effect exerted by the binge drinkers, not included in the intervention, on those who are vulnerable. On the other hand, due to the financial limitations in budget, it is impossible to include all the binge drinkers in the intervention program. Therefore, how to choose a subset of individuals to be part of the intervention program (or to educate) so that the effect of the intervention program can spread through the whole group under consideration becomes the key item of inquiry. In an effort to address this question, the specific problem we study in this paper is the following: given an online social network of a community and a specific social problem, how to identify a subset of the individuals within the online social network to participate in an intervention program such that the intervention/education can result in a globally positive impact on the entire social network. We assume that 1) if more than half neighbors of an individual have positive impact on him, then the probability that this individual positively impact others in the network is high. 2) intervention program can convert a negative influential individual to a positive influential person. Our first assumption comes from an extensive body of evidence suggesting that one of the most powerful predictors of negative/risky behavior in individuals is whether an individual has friends who also engage in that behavior. In fact research [7] has shown that about 50% of the variance in adolescent personality primarily reflects the influence of peers. Due to Outside competition in terms of personality traits attained from peer influence, the more neighbors/ friends exerting positive influence, an individual has, the more likely he is to impact others in a positive way. The overall average effect of his negative neighbors will be overpowered by

the contributions of his positive neighbors. Our second assumption comes from the work in [9] [12], where nearly every individual in the feedback intervention program showed a reduction in drinking. With the above two assumptions, the problem is equivalent to selecting a subset of the individuals to participate in the intervention program such that each individual in the social network has more positive neighbors than negative ones.

Online social network can be represented as a graph of relationships and social interactions (edges) between individuals (nodes). We use the following network model to illustrate the online social network in context of the social problem: A undirected graph $G = (V, E, C)$ is used to represent the online social network. We use an undirected graph because friendship in an online social network are mostly bi-directional. V is the set of nodes in which each node is an individual in the online social network. E is the set of edges in which each edge represents the existence of a social connection between the two endpoints. C is the compartment vector that saves the compartment of each node. The compartment of a node decides whether it has positive or negative impact on its neighbors. For example, for the drinking problem [1], the compartment of each node is one of the followings: abstainer, problem drinker, social drinker or binge drinker. A node in the abstainer compartment has positive impact and all nodes in any of the other three compartments have negative impact. With the above network model, we define the problem of selecting a *positive influence dominating set (PIDS)* in the online social network G as finding a subset P of V such that any node u in V are dominated by at least $\lceil \frac{d(u)}{2} \rceil$ nodes in P where $d(u)$ is the degree of node u. We propose a PIDS selection algorithm and evaluate its effect on a real online social network data set through simulations by comparing the size and the average positive degree of PIDS with that of the traditional 1-dominating set. The results illustrate that by strategically choosing 26% more people in PIDS than in the 1-dominating set to participate in the intervention program, the average positive degree increases approximately by 3.3 times. This is a considerable increase in overall average positive influence emitted in an individual in comparison to the nominal increase in number of participants. Thus by moderately increasing the participation related cost, the probability of positive influencing the whole community through the intervention program is significantly higher. Our simulation results also reveal that a power law graph has empirically larger dominating set size than a random graph even though the dominating set problem is a theoretically easier problem in a power law graph than in a random graph.

Our contributions in this paper include: 1) we introduce the PIDS problem which formalizes the problem of utilizing online social network to help solving social problems; 2) we propose and evaluate the PIDS selection algorithm with a real online social network data set and compare the size of PIDS and traditional 1-dominating set; 3) we study the size of PIDS and 1-dominating set in both a power law graph and a random graph.

The rest of this paper is organized as follows. Section 2 describes the related work. In section 3, we present the PIDS selection algorithm. Section 4 shows the

simulation results of PIDS on a real online social network topology. Section 5 concludes this paper and discusses our future plan.

2 Related Work

Most of the current research in online social network fall in two categories: one is to understand the properties and characteristics of online social networks, such as the work in [10,11,2]. The other is to study how to utilize social network to spread ideas and information as presented in [8,4]. Our work focuses on exploring how to utilize online social networks to help alleviate social problems in the physical world. The social problem is different from spreading ideas and information. The spread of ideas and information is one direction in that once a person in influence to adopt an idea or learn some information, he cannot revert to his original state by means of future influence. The positive or negative influence in social problems can flow in two directions, that is a positive individual can convert to a negative individual then can move back and forth between these two states multiple times. Another different between our work and that of [8,4] is that we find a set of individuals that guarantees the positive effect of education will be injected into the entire group. They focus on finding a subset of a pre-established fix size that maximize the spread of information, but not the subset regardless of size that infuses information to the whole group.

3 Positive Influence Dominating Set Selection Algorithm

In this section, we present a PIDS selection algorithm for the positive dominating set problem formalized in the earlier section. First we define and explain a few terms and definitions used in the description of our algorithm. Each node can have either *positive* or *negative* impact on its neighbor nodes. We call a node with positive impact a *positive node* and a node with negative impact a negative node. The *positive degree* of a node is the number of its positive neighbors. The same holds for negative degree. The *compartment* of a node decides whether the node is a positive or a negative node. For example, in the context of college drinking [1], a node in the abstainer compartment is a positive node and a node in any other compartment is a negative node. Nodes that are chosen into the PIDS are marked as positive nodes. Thus a neighbor u of v is a *positive neighbor* if u is initially a positive node or u is selected into the PIDS. A *1-dominating set* is S of a graph G is a subset of nodes in G such that every node not in S has at least one neighbor in S. A *positive influence dominating set* P of a graph G is a subset of nodes in G that any node u in G are dominated by at least $\lceil \frac{d(u)}{2} \rceil$ nodes in P where $d(u)$ is the degree of node u.

The main idea of PIDS algorithm is as follows: first prune the original graph by removing the initial positive nodes, then iteratively choose a 1-dominating set of the graph consisting of nodes with less than half neighbors as positive neighbors until all nodes in the original graph are either positive nodes or have more positive neighbors than negative ones. To choose a 1-dominating set of

a graph, we use a greedy algorithm similar to the one in [5]. This algorithm selects the node with the largest node degree into the dominating set. In our greedy algorithm, we choose the node that can dominate most negative nodes into the dominating set. [5] has proven that the simple greedy algorithm gives a $1 + o(1)$ approximation with a small constant in $o(1)$ to the 1-dominating set problem in a power law graph. Algorithm 1 gives the details of the PIDS algorithm. Compartment vector C contains the nodes that are initially in positive compartment. $C = \phi$ means every node is an negative node.

Algorithm 1. Positive Influence Dominating Set Selection Algorithm

1: INPUT: A graph $G = (V, E, C)$ where V is the set of nodes, E is the set of edges that capture the social interactions of the nodes, C is the set of nodes that are initially in positive compartment.
2: OUTPUT: A subset P of V such that any node u in V has at least $\lceil \frac{d(u)}{2} \rceil$ neighbors in $P \bigcup C$, where $d(u)$ is the degree of node u.
3: initialize the status of all nodes in V to NEGATIVE and P to empty
4: let $V' = V - C$ and set the status of nodes in C to POSITIVE
5: calculate the degree and the positive degree of each node in V
6: T is the set of nodes in V' that have more positive neighbors, set the status of nodes in T to POSITIVE
7: let $V' = V' - T$
8: while not every node in V' has more positive neighbors
9: find a 1-dominating set S that dominates all nodes in V' using greedy algorithm as in [5]
10: update $P = P \bigcup S$
11: $V' = V' - S$
12: update the positive degree of each node in V
13: T is the set of nodes in V' that have more positive neighbors, set the status of nodes in T to POSITIVE
14: $V' = V' - T$
15: end of while
16: let $U = C \bigcup P$
17: while not every $w \in U$ has more positive neighbors
18: x is a neighbor of w that has maximum negative degree among all the neighbors of w
19: let $P = x \bigcup P$
20: update the positive degree of each node in V
21: end of while
22: The nodes in set P are positive influence dominating set of G, and the positive degree of each node is also calculated.

4 Performance Evaluations

To evaluate the effect of our PIDS algorithm, we first collect the data from one of the popular gaming applications, Fighter's Club (FC) [11] on Facebook social networking site. The FC game has attracted over 3.4 million Facebook users

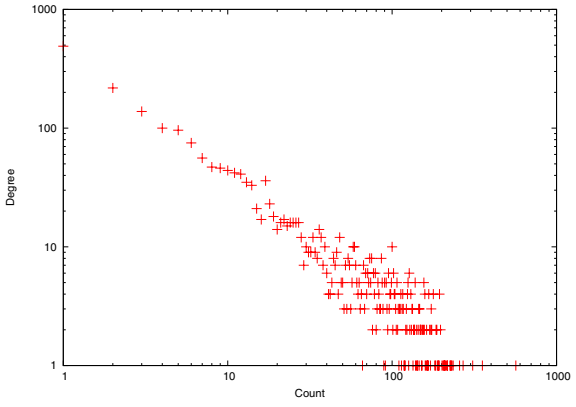


Fig. 1. Node degree distribution of an gaming application of 2334 users on facebook

since it was initially launched on June 2007. The gaming application records the players as well as their IP addresses. See [11] for a detailed descriptions of the game. In our study, we choose a subset of 2,334 Facebook users that play the online games together, and their IP addresses belong to the same IP network. Figure 1 illustrates the distribution of the node degree in this online community. Similar to the observations in prior studies [3], we find that the node degree in this Facebook application community also follows a power-law distribution.

To understand the effect of PIDS, we need to answer the following questions: 1) how many nodes need to be selected into the PIDS and how influential these nodes can be. To measure influence, we calculate the average over the number of positive neighbors of each node (called positive degree). The higher the average positive degree is, the more influential the PIDS can be. 2) what is the difference between the size of PIDS and a 1-dominating set and what is the difference between the influence of these two sets. 3) what is the difference between the size of dominating sets (both the PIDS and the 1-dominating set) of a power-law graph and that of a random graph. There are two main 1-dominating set construction algorithms, greedy algorithm (which is used in the step of generating 1-dominating set in the PIDS algorithm) or tree based algorithm as the one proposed in [14]. Tree based algorithm first builds a depth first tree then chooses a node that has the lowest level and most negative neighbors among all nodes on the same level in the dominating set. We are also interested in the performance of these two algorithms in the PIDS problem. In the remaining of this paper, we call them greedy PIDS algorithm and tree-based PIDS algorithm respectively.

Table 1 illustrates the results on the topology retrieved from the FC gaming application on the Facebook social network, whose node degree follows a power-law distribution. The size of the community is 2334, and average degree is 28.42, with 15% of all nodes initialized as positive. The result in the table is the average over 100 runs. Note that for each run, the size and average degree do not change. Since the abstainer nodes are randomly chosen, the network topology is different for each run. As we can see, the greedy algorithm has better

Table 1. Dominating set of a topology retrieved from the FC gaming application on Facebook social network

Algorithm	Size of dominating set (percentage)	Avg. positive degree (percentage)
greedy positive influence dominating set	1358 (58.2%)	22.5 (79.3%)
greedy 1-dominating set	744 (31.9%)	6.87 (24.1%)
tree-based positive influence dominating set	1478 (63.3%)	23.1 (81.4%)
tree-based 1-dominating set	851 (36.5%)	6.85 (24.1%)

performance over the tree-based algorithm in this power law graph. Applying greedy PIDS algorithm, 58% of the nodes are chosen into the dominating set and the resulting average positive degree is 22.5, which means that for a given node, on average, approximately 79% of its neighbors are positive nodes. This outcome of 79% is significant larger than our set goal, which is 50%. The high average positive degree can greatly increase the possibility of the whole community turning into a positive community. Furthermore, it indicates that we can prune the dominating set and reduce its size while keeping the average positive degree at around 50%. For greedy 1-dominating set algorithm, 31.9% nodes are chosen into the dominating set and the resulting average positive degree is only 6.85, which means that for a given node, approximately 76% of its neighbors are negative neighbors. Thus there exists a great possibility for the node of becoming a negative node and negatively impact others. The simulations reveal that by strategically choosing 26% more people into PIDS to participate in an intervention program, the average positive degree increases approximately by 3.3 times. In terms of the application, we can be confident that by moderately increasing the participation related cost, the probability of positive influencing the whole community through an intervention program where participants selection is determined by the greedy PIDS is significantly higher. Another observation is that the greedy algorithm gives smaller dominating sets than the tree-based algorithm does in an online social network, in which the node degree follows a power law distribution.

Table 2 illustrates the size of dominating sets in random graphs. We simulate a random graph by throwing a set of nodes to a square area and randomly generate the (x, y) coordinates of each node. Each node is associated with a variable range and if two nodes are within each other's range, there exists a link connecting them. We set the size of network to 2334 and initialize 15% nodes as positive, which is the same as the real online social network retrieved from the gaming application on the Facebook website, and adjust the parameters (area edge length, minimum range and maximum range) to get an average node degree close to the online social network topology (Note it is hard to generate a random graph that has exactly the same average node degree as that in the online social

Table 2. Dominating set of random networks

Algorithm	Size of dominating set (percentage)	Avg. positive degree (percentage)
greedy positive influence dominating set	1166 (50%)	19.3 (67%)
greedy 1-dominating set	233 (10%)	6.47 (22.5%)
tree-based positive influence dominating set	1196 (51.3%)	18.8 (65.2%)
tree-based 1-dominating set	253 (10.9%)	6.32 (21.9%)

network). The average of the average degree over 100 runs is 28.82, and Comparing Table I and II, we can see that under similar settings (same node size and similar density), a random graph has significantly smaller dominating set than a online social network which is a power law graph. This contradicts our conjecture. Heuristically, in a power law graph, a small set of nodes with high degree should dominate most of the nodes in the graph and thus resulting in a smaller dominating set. An explanation can be that even though the network is clustered around the most influential (or connected) nodes in the power law graph, there are more nodes that are sparse (has few neighbors) so more dominator nodes are chosen to dominate these sparse nodes. This interesting result needs further investigation.

In summary, in a typical online social network where average node degree is about 28.42 and 15% nodes as initially positive nodes, approximately 60% nodes are chosen into the PIDS, which results in an average percentage of positive degree over node degree as high as approximately 79%. Furthermore, there are 26% more people in the PIDS than those in 1-dominating set, while the average positive degree in the PIDS is approximately 3.3 times that in 1-dominating set. In the case of college drinking where participants of an intervention program are selected accord with the greedy PIDS, by moderately increasing the participation related cost, the probability of positive influencing the entire community is significantly higher. We also found that power law graphs have larger dominating set size than random graphs even though dominating set problem is theoretically an easier problem in a power law graph than in a random graph [6].

5 Conclusions

In this paper, we introduced and studied the problem of how to utilize online social network as a medium to help alleviate a certain social problem. We proposed the PIDS selection algorithm to evaluate the effect of educating a subset of the entire target group susceptible to a social problem. Our simulation results reveal that approximately 60% of the whole group under consideration needs to be chosen into the PIDS to achieve the goal that every individual in the community

has more positive neighbors than negative neighbors. We also discovered that the dominating sets of a power-law graph is larger than that of a random graph. Our future work includes applying a mathematical model to understand the influence of each individual and study the effect of PIDS over a period of time. Since the PIDS is fairly large in power law graph, we will investigate what is an empirically proper positive degree threshold that can spread the positive education influence throughout the entire community under consideration through additional modeling, experiment and data analysis.

References

1. Almada, L., Camacho, E., Rodriguez, R., Thompson, M., Voss, L.: Deterministic and Small-World Network Models of College Drinking Patterns. In: AMSSI Technical Report (2007), http://www.amssi.org/reports/alcohol2006.pdf
2. Anagnostopoulos, A., Kumar, R., Mahdian, M.: Influence and Correlation in Social Networks. In: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 7–15 (2008)
3. Barabási, A.-L., Bonabeau, E.: Scale-Free Networks. Scientific American 288(55), 60–69 (2003)
4. Bharathi, S., Kempe, D., Salek, M.: Competitive Influence Maximization in Social Networks. In: Deng, X., Graham, F.C. (eds.) WINE 2007. LNCS, vol. 4858, pp. 306–311. Springer, Heidelberg (2007)
5. Eubank, S., Anil Kumar, V.S., Marathe, M.V., Srinivasan, A., Wang, N.: Structural and Algorithmic Aspects of Massive Social Networks. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 718–727 (2004)
6. Ferrante, A., Pandurangan, G., Park, K.: On the Hardness of Optimization in Power-Law Graphs. Theoretical Computer Science 393, 220–230 (2008)
7. Jaccard, J., Blanton, H., Dodge, T.: Peer Influences on Risk Behavior: Analysis of the Effects of a Close Friend. Developmental Psychology 41, 135–147 (2005)
8. Kempe, D., Kleinberg, J., Tardos, E.: Maximizing the Spread of Influence through a Social Network. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 137–146 (2003)
9. Larimer, M.E., Cronce, J.M.: Identification, Prevention and Treatment: A Review of Individual-Focused Strategies to Reduce Problematic Alcohol Consumption by College Students. J. Studies on Alcohol 14, 148–163 (2002)
10. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and Analysis of Online Social Networks. In: Proceedings of the 7th ACM SIGCOMM conference on Internet Measurement Conference, pp. 29–42 (2007)
11. Nazir, A., Raza, S., Chuah, C.N.: Unveiling Facebook: A Measurement Study of Social Network Based Applications. In: Proceedings of the 8th ACM SIGCOMM Internet Measurement Conference, pp. 43–56 (2008)
12. Walters, S.T., Bennett, M.E.: Addressing Drinking among College Students: A Review of the Empirical Literature. Alcoholism Treatment Quarterly 18, 61–77 (2000)
13. Walters, S.T., Neighbors, C.: Feedback Interventions for College Alcohol Misuse: What, Why and for Whom? Addictive Behaviors 30, 1168–1182 (2005)
14. Wan, P.J., Alzoubi, K.M., Frieder, O.: Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks. Mobile Networks and Applications 9(2), 141–149 (2004)

Optimal Algorithms for the Online Time Series Search Problem*

Yinfeng Xu[1,2], Wenming Zhang[1,2], and Feifeng Zheng[1,2]

[1] School of Management, Xi'an Jiaotong University, Xi'an 710049, China
[2] The State Key Lab for Manufacturing Systems Engineering, Xi'an 710049, China
wenming.zhang.xjtu@gmail.com

Abstract. In the problem of online time series search introduced by El-Yaniv et al. [4], a player observes prices one by one over time and shall select exactly one of the prices on its arrival without the knowledge of future prices, aiming to maximize the selected price. In this paper, we extend the problem by introducing profit function. Considering two cases where the search duration is either known or unknown beforehand, we propose two optimal deterministic algorithms respectively. The models and results in the paper generalize those of El-Yaniv et al. [4].

Keywords: Time Series Search, Profit Function, Online Algorithm, Competitive Ratio.

1 Introduction

The problem of online time series search was introduced by El-Yaniv et al. [4], where a player observes a series of n prices sequentially in order to select the highest price in the series. On the observation of each price presented, the player has to decide immediately whether to accept the price or not without the knowledge of future prices. The profit depends on the price selected by the player. El-Yaniv et al. [4] proved that if the prices are bounded within interval $[m, M]$ $(0 < m < M)$, the optimal algorithm is to accept the first price no less than $\sqrt{Mm}$ and the competitive ratio is $\sqrt{M/m}$. Damaschke, Ha and Tsigas [3] studied another case where the upper and lower bounds of prices vary as time goes on. The approach is also adopted by Lorenz, Panagiotou and Steger [7] for the k-search problem to search for the k highest (or lowest) prices in one series. All the above work assumes that the profit to accept a price is exactly the price itself, ignoring when the price shows up in the series. In many real scenarios, however, this is not the case. For example, the player may need to pay a sampling cost at each time period to observe a price, and then the accumulated sampling cost increases as time goes on. Hence, the profit to accept a price at some period can be regarded as a function of the price, such as equaling the accepted price minus the accumulated sampling cost. In the paper we will extend the basic

* The work is supported by NSF grants of China, no. 70525004, 60736027 and 70702030.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 322–333, 2009.

model in [4] by introducing profit function, and give more general results on competitiveness.

1.1 Related Work

The problem of time series search has received considerable attention in mathematical economics and operations research since 1960's. It is quite related to the optimal stopping problem (see [8]) and the secretary problem (see [5,2]), both of which have many extensions such as secretary problem with discounts (see [1]) and with inspection costs (see [6]). Most of the previous work follows Bayesian approach, and algorithms are developed under assumption that prices are generated by some (e.g. uniform) distribution which is known beforehand (see [2]). Since the distribution of prices may not be known to the player in many situations, some research attempts to relax the assumption. Rosenfield and Shapiro [9] studied the case where the price distribution is a random variable.

1.2 Competitive Ratio

Sleator and Tarjan [10] proposed to evaluate the performance of online algorithms by competitive analysis. For an arbitrary given price sequence σ, the profit of an online algorithm ALG is compared with that of an offline player's algorithm OPT, which knows all the prices in advance. Let $ALG(\sigma)$ and $OPT(\sigma)$ denote the profits of ALG and OPT in σ respectively. The competitive ratio of ALG is then defined as

$$\alpha = \sup_{\sigma} \frac{OPT(\sigma)}{ALG(\sigma)}$$

We also say that ALG is α-competitive. Given that there are not any online algorithms with competitive ratio less than α, ALG is called an optimal online algorithm.

The rest of the paper is organized as follows. Section 2 models the time series search problem and gives some assumptions as well. In Section 3 we investigate the case with known duration, and in Section 4 we discuss the case with unknown duration. Finally, Section 5 concludes the paper.

2 Problem Statement and Assumptions

The problem: A player is searching for one price of some asset in a price quotation sequence aiming to maximize the profit. There is only one chance for the player to select a price. At each time period $i = 1, 2, \cdots, n$ where the time horizon n is a natural number, a price quotation p_i is received. The player has to decide immediately whether to accept the price. Once it is accepted at period i, the player cannot accept another price in later periods and the profit for p_i is denoted by $f_i(p_i)$, i.e., the *profit function* at period i. Otherwise p_i expires and p_{i+1} arrives in the next period. Note that the price series may end at some period $l \leq n$, i.e., the last price is p_l, and we call l the *duration* of the series.

For the online time series search problem, there are four basic assumptions in below.

(1) The values of n, m, M and the functions $f_i(p)$, $i = 1, 2, \cdots, n$ are known beforehand to the player.

(2) $n \geq 2$. Otherwise if $n = 1$, both online and offline players have to accept the only price with optimal profit.

(3) The profit function $f_i(p)$ $(i = 1, 2, \cdots, n)$ is continuous and increasing in p, which vary within interval $[m, M]$ where $0 < m < M$.

(4) For an arbitrary $p \in [m, M]$, $f_1(p) \geq f_2(p) \geq \cdots \geq f_n(p) > 0$.

The third assumption and the fourth assumption tell that at each period larger price results in larger profit and for a specific price p, the profit is larger in an earlier period than in a later one, respectively. In the following, we will divide the problem into two cases according to the knowledge of duration.

Variant 1: Known duration. The duration of the price quotation sequence is equal to n which is known to the player at the first beginning. The player can at least has a profit of $f_n(p_n)$ by accepting the last price p_n.

Variant 2: Unknown duration. The player has the only information that the duration of the price quotation sequence is at most n beforehand. At the beginning of each period the player is told whether the sequence ends at the period or not.

Note that in both variants, it is sufficient to analyze the case where $f_{i+1}(M) > f_i(m)$ for $i = 1, 2, \cdots, n-1$, otherwise if $f_{i+1}(M) \leq f_i(m)$ holds at some period i, the player will accept a price and the game ends on or before period i since he gains a profit $f_j(p_j) \geq f_i(m)$ to accept p_j $(1 \leq j \leq i)$ at period j more than that to accept p_k $(i + 1 \leq k \leq n)$ at period k with $f_k(p_k) \leq f_{i+1}(p_k) \leq f_{i+1}(M) \leq f_i(m)$. In the rest of the work, we will focus on the case such that $f_{i+1}(M) > f_i(m)$ for $1 \leq i \leq n-1$ in a price series.

3 Online Time Series Search Problem with Known Duration

In the section, we will discuss the case with known duration n, and present an optimal deterministic algorithm.

3.1 The Online Algorithm

Before describing the algorithm, we give some preliminary definitions. Let

$$\alpha = \min\Big\{\big\{\max\{\frac{f_{i+1}(M)}{f_i(m)}, \sqrt{\frac{f_2(M)}{f_i(m)}}\}, i = 1, 2, \cdots, n-1\big\}, \sqrt{\frac{f_2(M)}{f_n(m)}}\Big\} \quad (1)$$

Note that $\alpha \geq 1$ since $f_{i+1}(M) > f_i(m)$ and $f_2(M) \geq f_n(m)$. By the definition of α, there exists a natural number r such that either

$$\alpha = \frac{f_{r+1}(M)}{f_r(m)}, \quad \text{for } r \leq n-1$$

or

$$\alpha = \sqrt{\frac{f_2(M)}{f_r(m)}}, \quad \text{for } r \leq n.$$

Ties are broken by selecting the smallest r. If $\alpha = f_{r+1}(M)/f_r(m)$, let p_i^* $(1 \leq i \leq r)$ either be the solution of equation $\alpha f_i(p_i^*) = f_{i+1}(M)$ or $p_i^* = m$ in the case that there is no solution for the equation. Ties are broken by selecting the p_i^* with the smallest value. Otherwise if $\alpha = \sqrt{f_2(M)/f_r(m)}$, then let $i^* = max\{i | f_{i+1}(M) \geq \sqrt{f_2(M) f_r(m)}\}$. Let $p_i^* = m$ for $min\{i^*, r-1\} < i \leq r$, and for $1 \leq i \leq min\{i^*, r-1\}$, p_i^* either be the solution of equation $\alpha f_i(p_i^*) = f_{i+1}(M)$ or $p_i^* = m$ in the case that there is no solution for the equation. Note that for $i = 1$, $f_2(M) > f_1(m)$ and then

$$\max\left\{\frac{f_2(M)}{f_1(m)}, \sqrt{\frac{f_2(M)}{f_1(m)}}\right\} = \frac{f_2(M)}{f_1(m)}.$$

Together with formula (1), we obtain

$$\alpha \leq \frac{f_2(M)}{f_1(m)}$$

and thus

$$f_1(m) \leq \frac{f_2(M)}{\alpha} \leq f_2(M) \leq f_1(M),$$

implying that p_1^* is only defined by the solution of equation $\alpha f_1(p_1^*) = f_2(M)$. Moreover, $p_r^* = m$ by the above discussion.

Algorithm AKD **(Algorithm with Known Duration):**

Step 1. Let $i = 1$.
Step 2. At period i, if $p_i \geq p_i^*$ then accept p_i with profit $f_i(p_i)$,
otherwise if $p_i < p_i^*$, go to Step 3.
Step 3. $i = i + 1$ and go to Step 2.

Note that AKD will accept a price on or before period r since $p_r^* = m$.

3.2 Competitive Analysis

Lemma 1. *If AKD accepts p_i at some period i $(1 \leq i \leq r)$, then $m < p_j^* \leq M$ for $1 \leq j \leq i-1$.*

Proof. For the first inequality, if otherwise $p_j^* = m$, AKD will accept a previous price on or before period j since $p_j \geq m$. Moreover, $p_j^* \leq M$ due to $f_j(p_j^*) = f_{j+1}(M)/\alpha \leq f_{j+1}(M) \leq f_j(M)$ and assumption (3). The lemma follows. □

Lemma 1 implies that p_j^* shall be the solution of equation $\alpha f_j(p_j^*) = f_{j+1}(M)$ for $1 \leq j \leq i-1$.

Theorem 1. *AKD has competitive ratio of α for the online time series search problem with known duration.*

Proof. Let ε denote an arbitrarily small positive real number. We discuss two cases according to different values of α.

Case 1. $\alpha = f_{r+1}(M)/f_r(m)$. According to formula (1), we have

$$\frac{f_{r+1}(M)}{f_r(m)} \geq \sqrt{\frac{f_2(M)}{f_r(m)}},$$

which implies

$$f_{r+1}(M) \geq \frac{f_2(M)f_r(m)}{f_{r+1}(M)} = \frac{f_2(M)}{\alpha} = f_1(p_1^*).$$

As AKD accepts a price on or before period r, assume without loss of generality that it accepts p_i at period i $(1 \leq i \leq r)$. By Lemma 1, $m < p_j^* - \varepsilon < M$ for $j = 1, 2, \cdots, i-1$. So, the worst price sequence to AKD is $\sigma_1 = (p_1^* - \varepsilon, \cdots, p_{i-1}^* - \varepsilon, p_i^*, M, \cdots)$. The profit of AKD in σ_1 is

$$AKD(\sigma_1) = f_i(p_i^*) = \frac{f_{i+1}(M)}{\alpha}.$$

For OPT, combining

$$f_j(p_j^*) = \frac{f_{j+1}(M)}{\alpha}, \qquad 1 \leq j \leq i-1,$$

and assumption (4), $f_1(p_1^*) \geq \cdots \geq f_i(p_i^*)$ As $\varepsilon \to 0$, OPT's profit is as follows.

$$\begin{aligned} OPT(\sigma_1) &\approx max\{f_1(p_1^*), \cdots, f_i(p_i^*), f_{i+1}(M)\} \\ &= max\{f_1(p_1^*), f_{i+1}(M)\} \\ &= f_{i+1}(M) \end{aligned}$$

The last equation holds since $f_{i+1}(M) \geq f_{r+1}(M) \geq f_1(p_1^*)$. So in this case, we have

$$\frac{OPT(\sigma_1)}{AKD(\sigma_1)} = \alpha.$$

Case 2. $\alpha = \sqrt{f_2(M)/f_r(m)}$. According to formula (1), if $r \leq n-1$, then

$$\sqrt{\frac{f_2(M)}{f_r(m)}} \geq \frac{f_{r+1}(M)}{f_r(m)}.$$

By the definition of p_i^* in the case of $\alpha = \sqrt{f_2(M)/f_r(m)}$, $p^*_{\min\{i^*, r-1\}+1} = m$. So, AKD will accept a price on or before period $\min\{i^*, r-1\} + 1$. Assume without loss of generality that it accepts p_i at some period i $(1 \leq i \leq \min\{i^*, r-1\} + 1)$. We divide the case into two sub-cases according to different values of i.

Case 2.1. $1 \leq i \leq min\{i^*, r-1\}$. In this sub-case, noting that since $i \leq i^*$, $f_{i+1}(M) \geq \sqrt{f_2(M)f_r(m)} = \frac{f_2(M)}{\alpha} = f_1(p_1^*)$, the worst price sequence and the following discussion are the same as those in Case 1.

Case 2.2. $i = min\{i^*, r-1\} + 1$. $i = i^* + 1$ if $i^* \leq r-2$ and $i = r$ if $i^* > r-2$. We already know that $m < p_j^* - \varepsilon < M$ for $j = 1, 2, \cdots, i-1$. So, the worst price sequence to AKD is

$$\sigma_2 = \begin{cases} (p_1^* - \varepsilon, p_2^* - \varepsilon, \cdots, p_{i-1}^* - \varepsilon, m, M, \cdots) & : \quad i = i^* + 1 \\ (p_1^* - \varepsilon, p_2^* - \varepsilon, \cdots, p_{i-1}^* - \varepsilon, m, M, \cdots) & : \quad i = r \leq n-1 \\ (p_1^* - \varepsilon, p_2^* - \varepsilon, \cdots, p_{i-1}^* - \varepsilon, m) & : \quad i = r = n \end{cases}$$

We further discuss three sub-cases according to the three worst price sequences.

Case 2.2.1. $i = i^* + 1$ and $\sigma_2 = (p_1^* - \varepsilon, p_2^* - \varepsilon, \cdots, p_{i-1}^* - \varepsilon, m, M, \cdots)$. The profit of AKD is $AKD(\sigma_2) = f_i(m)$. For OPT, as $\varepsilon \to 0$,

$$\begin{aligned} OPT(\sigma_2) &\approx max\{f_1(p_1^*), \cdots, f_{i-1}(p_{i-1}^*), f_i(m), f_{i+1}(M)\} \\ &= max\{f_1(p_1^*), f_{i+1}(M)\} \\ &= f_1(p_1^*) \end{aligned}$$

where the second equation holds since $f_1(p_1^*) \geq \cdots \geq f_{i-1}(p_{i-1}^*) \geq f_i(m)$, and the third equation holds since $f_{i+1}(M) < \sqrt{f_2(M)f_r(m)} = f_1(p_1^*)$ due to $i > i^*$ and the definition of i^*. Hence in this sub-case

$$\frac{OPT(\sigma_2)}{AKD(\sigma_2)} = \frac{f_1(p_1^*)}{f_i(m)} \leq \frac{f_1(p_1^*)}{f_r(m)} = \frac{f_2(M)/\alpha}{f_r(m)} = \alpha.$$

Case 2.2.2. $i = r \leq n-1$ and $\sigma_2 = (p_1^* - \varepsilon, p_2^* - \varepsilon, \cdots, p_{i-1}^* - \varepsilon, m, M, \cdots)$. Then $AKD(\sigma_2) = f_i(m) = f_r(m)$. For OPT, $OPT(\sigma_2) \approx max\{f_1(p_1^*), f_{r+1}(M)\}$ with similar reasoning as in Case 2.2.1. In the previous sub-case, we already have

$$\frac{f_1(p_1^*)}{f_r(m)} = \alpha.$$

Moreover,

$$\frac{f_{r+1}(M)}{f_r(m)} \leq \sqrt{\frac{f_2(M)}{f_r(m)}} = \alpha,$$

where the inequality holds by the condition of Case 2 and $r \leq n-1$. So in this sub-case

$$\frac{OPT(\sigma_2)}{AKD(\sigma_2)} \leq \alpha.$$

Case 2.2.3. $i = r = n$ and $\sigma_2 = (p_1^* - \varepsilon, p_2^* - \varepsilon, \cdots, p_{i-1}^* - \varepsilon, m)$. In this sub-case, $AKD(\sigma_2) = f_i(m) = f_r(m)$. For OPT, combining $f_1(p_1^*) \geq \cdots \geq f_{i-1}(p_{i-1}^*) \geq f_i(m)$ with $\varepsilon \to 0$, $OPT(\sigma_2) \approx f_1(p_1^*)$. Hence we have

$$\frac{OPT(\sigma_2)}{AKD(\sigma_2)} = \alpha.$$

The theorem follows. □

In the following, we will show that no deterministic algorithms behave better than AKD in competitiveness for Variant 1 of the problem.

Theorem 2. *For the online time series search problem with known duration, no deterministic algorithm has competitive ratio less than α.*

Proof. Let ALG be any deterministic algorithm. We will construct a price sequence $\hat{\sigma} = (\hat{p}_1, \hat{p}_2, \cdots, \hat{p}_n)$ such that ALG cannot achieve a competitive ratio less than α.

The sequence $\hat{\sigma}$ is constructed as follows. First, similar to the discussion on the existence of p_1^* to equation $\alpha f_1(p_1^*) = f_2(M)$, we can define price $\hat{p}_1 \in [m, M]$ given by equation $\alpha f_1(\hat{p}_1) = f_2(M)$. At period 1, we present $\hat{p}_1$ to ALG, if ALG accepts the price, we further present the rest $n-1$ prices $\hat{p}_2 = \cdots = \hat{p}_n = M$, otherwise we present $\hat{p}_2 = m$ and go to the next period. Similarly, at period 2, if ALG accepts $\hat{p}_2$, then we further present the rest $n-2$ prices $\hat{p}_3 = \cdots = \hat{p}_n = M$, otherwise we present $\hat{p}_3 = m$ and go to the next period. This is repeated until either at some period i $(2 \leq i \leq n-1)$, ALG accepts $\hat{p}_i$ or ALG accepts $\hat{p}_n = m$ at period n. In the first case, we further present the rest $n-i$ prices $\hat{p}_{i+1} = \cdots = \hat{p}_n = M$.

Assume that ALG accepts $\hat{p}_i$ at period i. We discuss three cases depending on the value of i.

Case 1. $i = 1$. In this case $OPT(\hat{\sigma}) \geq f_2(M)$ and $ALG(\hat{\sigma}) = f_1(\hat{p}_1)$ implying

$$\frac{OPT(\hat{\sigma})}{ALG(\hat{\sigma})} \geq \frac{f_2(M)}{f_1(\hat{p}_1)} = \alpha.$$

Case 2. $2 \leq i \leq n-1$. In this case we further divide the case into the following two sub-cases.

Case 2.1. $f_{i+1}(M)/f_i(m) \geq \sqrt{f_2(M)/f_i(m)}$. By the definition of α,

$$\frac{f_{i+1}(M)}{f_i(m)} \geq \alpha.$$

OPT will gain a profit satisfying $OPT(\hat{\sigma}) \geq f_{i+1}(M)$ while ALG's profit $ALG(\hat{\sigma}) = f_i(m)$. Hence we have

$$\frac{OPT(\hat{\sigma})}{ALG(\hat{\sigma})} \geq \frac{f_{i+1}(M)}{f_i(m)} \geq \alpha.$$

Case 2.2. $f_{i+1}(M)/f_i(m) < \sqrt{f_2(M)/f_i(m)}$. In this case,

$$\sqrt{\frac{f_2(M)}{f_i(m)}} \geq \alpha.$$

For OPT, $OPT(\hat{\sigma}) \geq f_1(\hat{p}_1)$ while ALG's profit $ALG(\hat{\sigma}) = f_i(m)$. Hence we have

$$\frac{OPT(\hat{\sigma})}{ALG(\hat{\sigma})} \geq \frac{f_1(\hat{p}_1)}{f_i(m)} \geq \frac{f_2(M)}{\alpha f_i(m)} \geq \frac{\alpha^2}{\alpha} = \alpha.$$

Case 3. $i = n$. By the definition of α, $\sqrt{f_2(M)/f_n(m)} \geq \alpha$. OPT's profit satisfies $OPT(\hat{\sigma}) \geq f_1(\hat{p}_1)$ while ALG's profit $ALG(\hat{\sigma}) = f_n(m)$. Hence we obtain

$$\frac{OPT(\hat{\sigma})}{ALG(\hat{\sigma})} \geq \frac{f_1(\hat{p}_1)}{f_n(m)} \geq \frac{f_2(M)}{\alpha f_n(m)} \geq \frac{\alpha^2}{\alpha} = \alpha.$$

According to the above discussion, ALG cannot have a competitive ratio less than α. The theorem follows. □

4 Online Time Series Search Problem with Unknown Duration

4.1 The Online Algorithm

Before describing the algorithm, we give some preliminary definitions. For every natural number l $(2 \leq l \leq n)$, let

$$\alpha_l = \min\Big\{\big\{max\{\frac{f_{i+1}(M)}{f_i(m)}, \sqrt{\frac{f_2(M)}{f_i(m)}}\}, i = 1, 2, \cdots, l-1\big\}, \sqrt{\frac{f_2(M)}{f_l(m)}}\Big\} \quad (2)$$

Note that $\alpha_l \geq 1$ since $f_{i+1}(M) > f_i(m)$ and $f_2(M) \geq f_l(m)$. Let

$$\bar{L} = \max\big\{L | L = \arg\max_{2 \leq l \leq n} \alpha_l\big\}.$$

Obviously, $\alpha_{\bar{L}} \geq \alpha_l$ for every l $(2 \leq l \leq n)$. By the definition of $\alpha_{\bar{L}}$, there exists a natural number s such that either

$$\alpha_{\bar{L}} = \frac{f_{s+1}(M)}{f_s(m)}, \qquad \text{for } s \leq \bar{L} - 1,$$

or

$$\alpha_{\bar{L}} = \sqrt{\frac{f_2(M)}{f_s(m)}}, \qquad \text{for } s \leq \bar{L}.$$

Ties are broken by selecting the smallest s. If $\alpha_{\bar{L}} = f_{s+1}(M)/f_s(m)$, let $\bar{p}_i^*$ $(1 \leq i \leq s)$ either be the solution of equation $\alpha_{\bar{L}} f_i(\bar{p}_i^*) = f_{i+1}(M)$ or $\bar{p}_i^* = m$ in the case that there is no solution for the equation. Ties are broken by selecting the $\bar{p}_i^*$ with the smallest value. Otherwise if $\alpha_{\bar{L}} = \sqrt{f_2(M)/f_s(m)}$, then let $\bar{i}^* = max\{i | f_{i+1}(M) \geq \sqrt{f_2(M) f_s(m)}\}$. Let $\bar{p}_i^* = m$ for $min\{\bar{i}^*, s-1\} < i \leq s$, and for $1 \leq i \leq min\{\bar{i}^*, s-1\}$, $\bar{p}_i^*$ either be the solution of equation $\alpha_{\bar{L}} f_i(\bar{p}_i^*) = f_{i+1}(M)$ or $\bar{p}_i^* = m$ in the case that there is no solution for the equation. For $i = 1$, $f_2(M) > f_1(m)$ and then

$$\max\Big\{\frac{f_2(M)}{f_1(m)}, \sqrt{\frac{f_2(M)}{f_1(m)}}\Big\} = \frac{f_2(M)}{f_1(m)}.$$

Combining formula (2) and the definition of $\bar{L}$, we obtain

$$\alpha_{\bar{L}} \leq \frac{f_2(M)}{f_1(m)},$$

and thus

$$f_1(m) \leq \frac{f_2(M)}{\alpha_{\bar{L}}} \leq f_2(M) \leq f_1(M),$$

implying that $\bar{p}_1^*$ is defined by the solution of equation $\alpha_{\bar{L}} f_1(\bar{p}_1^*) = f_2(M)$. Moreover, according to the above discussion, $\bar{p}_s^* = m$.

Algorithm *AUD* **(Algorithm with Unknown Duration):**

Step 1. Let $i = 1$.
Step 2. At period i, if $p_i \geq \bar{p}_i^*$ or the duration is exactly i
then accept p_i and the game ends; otherwise go to Step 3.
Step 3. $i = i + 1$ and go to Step 2.

Note that *AUD* will accept a price on or before period s since $\bar{p}_s^* = m$.

4.2 Competitive Analysis

Let $\underline{L} = \min\{L | L = \arg\max_{2 \leq l \leq n} \alpha_l\}$. Obviously, $\alpha_{\bar{L}} = \alpha_{\underline{L}}$. In the following we will give several lemmas.

Lemma 2. *If AUD accepts p_i at some period i ($1 \leq i \leq s$), then $m < \bar{p}_j^* \leq M$ for $1 \leq j \leq i-1$.*

The proof of Lemma 2 is the same as that of Lemma 1. Lemma 2 implies that $\bar{p}_j^*$ shall be the solution of equation $\alpha_{\bar{L}} f_j(\bar{p}_j^*) = f_{j+1}(M)$ for $1 \leq j \leq i-1$.

Lemma 3. *For each natural number $l < \underline{L}$, $\alpha_l = \sqrt{f_2(M)/f_l(m)} < \alpha_{\underline{L}}$.*

Proof. By the definition of α_l, there exists a natural number $i < l$ such that either

$$\alpha_l = \max\{\frac{f_{i+1}(M)}{f_i(m)}, \sqrt{\frac{f_2(M)}{f_i(m)}}\}$$

or

$$\alpha_l = \sqrt{\frac{f_2(M)}{f_l(m)}}.$$

If $\alpha_l = \max\{f_{i+1}(M)/f_i(m), \sqrt{f_2(M)/f_i(m)}\}$ together with $i < l < \underline{L}$,

$$\alpha_{\underline{L}} \leq \max\left\{\frac{f_{i+1}(M)}{f_i(m)}, \sqrt{\frac{f_2(M)}{f_i(m)}}\right\} = \alpha_l,$$

contradicting to the definition of $\underline{L}$. So, $\alpha_l = \sqrt{f_2(M)/f_l(m)}$, and by the definition of $\underline{L}$, $\alpha_l < \alpha_{\underline{L}}$. The lemma follows. □

Lemma 4. *For each natural number $i < s$, $\sqrt{f_2(M)/f_i(m)} < \alpha_{\underline{L}}$.*

Proof. By the definition of $\alpha_{\underline{L}}$, there exists a natural number t such that either

$$\alpha_{\underline{L}} = \frac{f_{t+1}(M)}{f_t(m)} \quad \text{for } t \le \underline{L} - 1$$

or

$$\alpha_{\underline{L}} = \sqrt{\frac{f_2(M)}{f_t(m)}} \quad \text{for } t \le \underline{L}.$$

Ties are broken by selecting the smallest t. Combining the definitions of t and s and equation $\alpha_{\bar{L}} = \alpha_{\underline{L}}$, we have $t = s$ and then $i < s = t \le \underline{L}$. By Lemma 3 we obtain $\sqrt{f_2(M)/f_i(m)} = \alpha_i < \alpha_{\underline{L}}$. □

Theorem 3. *AUD has competitive ratio of $\alpha_{\bar{L}}$ for the online time series search problem with unknown duration.*

Proof. Let ε denote an arbitrarily small positive real number. Assume that *AUD* accepts p_i at some period i. We will discuss two cases according to different conditions for *AUD* to accept p_i.

Case 1. *AUD* accepts p_i due to $p_i \ge \bar{p}_i^*$. We will discuss two sub-cases according to different values of $\alpha_{\bar{L}}$.

Case 1.1. $\alpha_{\bar{L}} = f_{s+1}(M)/f_s(m)$. The discussion of this case is the same as that of Case 1 in Theorem 1, replacing p_k^* by $\bar{p}_k^*$ $(k = i, j)$.

Case 1.2. $\alpha_{\bar{L}} = \sqrt{f_2(M)/f_s(m)}$. By the definition of $\bar{p}_i^*$ in the case,

$$\bar{p}^*_{\min\{\bar{i}^*, s-1\}+1} = m.$$

So *AUD* will accept a price on or before period $\min\{\bar{i}^*, s-1\}+1$. Assume without loss of generality that it accepts p_i at period i $(1 \le i \le \min\{\bar{i}^*, s-1\}+1)$. We further discuss two sub-cases according to different values of i.

Case 1.2.1. $1 \le i \le \min\{\bar{i}^*, s-1\}$. The discussion of this case is the same as that of Case 2.1 in Theorem 1, replacing p_k^* by $\bar{p}_k^*$ $(k = i, j)$, and replacing i^* and r by $\bar{i}^*$ and s respectively.

Case 1.2.2. $i = \min\{\bar{i}^*, s-1\}+1$. There are four sub-cases in this case. For the cases where $i = \bar{i}^*+1$ and where $i = s = n$, the discussions are the same as those of Case 2.2.1 and Case 2.2.3 respectively in Theorem 1. For the case where $i = s \le n-1$ and $s \ne \bar{L}$, the discussion is the same as that of Case 2.2.2 in Theorem 1. So we will focus on the fourth case where $i = s = \bar{L} \le n-1$. By case condition, $\alpha_{\bar{L}} = \sqrt{f_2(M)/f_{\bar{L}}(m)}$. By Lemma 2, $m < \bar{p}_j^* - \varepsilon < M$ for $j = 1, 2, \cdots, i-1$. The worst price sequence to *AUD* is $\sigma_1 = (\bar{p}_1^* - \varepsilon, \bar{p}_2^* - \varepsilon, \cdots, \bar{p}^*_{\bar{L}-1} - \varepsilon, m, M, \cdots)$. The profit of *AUD* is $AUD(\sigma_1) = f_{\bar{L}}(m)$. For *OPT* we have

$$f_j(\bar{p}_j^*) = \frac{f_{j+1}(M)}{\alpha_{\bar{L}}}, \quad 1 \le j \le \bar{L},$$

and assumption (4), $f_1(\bar{p}_1^*) \geq \cdots \geq f_{\bar{L}-1}(\bar{p}_{\bar{L}-1}^*) \geq f_{\bar{L}}(m)$ implying $OPT(\sigma_1) \approx max\{f_1(\bar{p}_1^*), f_{\bar{L}+1}(M)\}$. We claim by $s = \bar{L} \leq n-1$ that

$$\frac{f_{\bar{L}+1}(M)}{f_{\bar{L}}(m)} < \sqrt{\frac{f_2(M)}{f_{\bar{L}}(m)}} = \alpha_{\bar{L}}$$

since otherwise by the definitions of $\alpha_{\bar{L}}$ and $\alpha_{\bar{L}+1}$, $\alpha_{\bar{L}+1} \geq \alpha_{\bar{L}}$ which contradicts to the definition of $\bar{L}$. Moreover,

$$\frac{f_1(\bar{p}_1^*)}{f_{\bar{L}}(m)} = \frac{f_2(M)}{\alpha_{\bar{L}} f_{\bar{L}}(m)} = \frac{\alpha_{\bar{L}}^2}{\alpha_{\bar{L}}} = \alpha_{\bar{L}}.$$

Hence we have

$$\frac{OPT(\sigma_1)}{AUD(\sigma_1)} \approx \frac{max\{f_1(\bar{p}_1^*), f_{\bar{L}+1}(M)\}}{f_{\bar{L}}(m)} \leq \alpha_{\bar{L}}.$$

Case 2. AUD accepts p_i due to the duration is met in period i ($i < s$). By Lemma 2, $m < \bar{p}_j^* - \varepsilon < M$ for $j = 1, 2, \cdots, i-1$. The worst price sequence to AUD is $\sigma_2 = (\bar{p}_1^* - \varepsilon, \bar{p}_2^* - \varepsilon, \cdots, \bar{p}_{i-1}^* - \varepsilon, m)$. In this case, $AUD(\sigma_2) = f_i(m)$. For OPT, as $\varepsilon \to 0$, $OPT(\sigma_2) \approx f_1(\bar{p}_1^*)$ due to $f_1(\bar{p}_1^*) \geq \cdots \geq f_{i-1}(\bar{p}_{i-1}^*) \geq f_i(m)$. Thus we have

$$\frac{OPT(\sigma_2)}{AUD(\sigma_2)} \approx \frac{f_1(\bar{p}_1^*)}{f_i(m)} = \frac{f_2(M)}{\alpha_{\bar{L}} f_i(m)}.$$

Combining $i < s$ and Lemma 4, we obtain $f_2(M)/f_i(m) < \alpha_{\underline{L}}^2$. Hence we have

$$\frac{OPT(\sigma_2)}{AUD(\sigma_2)} < \frac{\alpha_{\underline{L}}^2}{\alpha_{\bar{L}}} = \frac{\alpha_{\bar{L}}^2}{\alpha_{\bar{L}}} = \alpha_{\bar{L}}.$$

The theorem follows. □

In the following we will prove that no deterministic algorithm can do better than AUD in competitiveness for variant 2 of the problem.

Theorem 4. *For the online time series search problem with unknown duration, no deterministic algorithm has a competitive ratio less than $\alpha_{\bar{L}}$.*

Proof. We construct a price sequence $\hat{\sigma} = (\hat{p}_1, \hat{p}_2, \cdots, \hat{p}_{\bar{L}})$, that is, $\bar{L} = n$, and it is sufficient to prove that an arbitrary online algorithm ALG cannot have a profit larger than $1/\alpha_{\bar{L}}$ times of OPT's in $\hat{\sigma}$. The rest reasoning is the same as that in the proof of Theorem 2.

Remark. Note that in formulas (1) and (2), if the profit function satisfies $f_i(p_i) = p_i$ for $1 \leq i \leq n$, then $\alpha = \alpha_l = \sqrt{M/m}$ and thus algorithms AKD and AUD have optimal competitive ratio the same as that in El-Yaniv et al. [4].

5 Conclusion

In the paper, we extended the original online time series search problem by introducing profit function. We investigate two cases where the player knows

the duration of price series and where he has not the knowledge of duration beforehand. We propose two algorithms AKD and AUD, and prove that they are optimal in the two cases respectively. The problem with profit function is a generalization of that in El-Yaniv et al. [4]. For the problem with different profit functions, it is an interesting work to design randomized algorithms to break the lower bounds of competitive ratio.

References

1. Babaioff, M., Dinitz, M., Gupta, A., Immorlica, N., Talwar, K.: Secretary Problems: Weights and Discounts, http://zeno.siam.org/proceedings/soda/2009/SODA09_135_babaioffm.pdf
2. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: Online Auctions and Generalized Secretary Problems. ACM SIGecom Exchanges 7(2), 1–11 (2008)
3. Damaschke, P., Ha, P.H., Tsigas, P.: Online Search with Time-Varying Price Bounds. Algorithmica (to appear)
4. El-Yaniv, R., Fiat, A., Karp, R.M., Turpin, G.: Optimal Search and One-Way Trading Online Algorithms. Algorithmica 30(1), 101–139 (2001)
5. Ferguson, T.S.: Who Solved the Secretary Problem? Statistical Science 4(3), 282–296 (1989)
6. Grant, P.: Secretary Problems with Inspection Costs as a Game. Metrika 29, 87–93 (1982)
7. Lorenz, J., Panagiotou, K., Steger, A.: Optimal Algorithms for k-Search with Application in Option Pricing. Algorithmica (to appear)
8. Peskir, G., Shiryaev, A.: Optimal Stopping and Free-Boundary Problems. Birkhäuser, Basel (2006)
9. Rosenfield, D.B., Shapiro, R.D.: Optimal Adaptive Price Search. Journal of Economic Theory 25(1), 1–20 (1981)
10. Sleator, D.D., Tarjan, R.E.: Amortized Efficiency of List Update and Paging Rules. Communication of the ACM 28, 202–208 (1985)

A Risk-Reward Competitive Analysis for the Newsboy Problem with Range Information*

Guiqing Zhang[1] and Yinfeng Xu[1,2]

[1] School of Management, Xi'an Jiaotong University, Xi'an 710049, China
gq.zhang@stu.xjtu.edu.cn
[2] The State Key Lab for Manufacturing Systems Engineering, Xi'an 710049, China
yfxu@mail.xjtu.edu.cn

Abstract. Recently, the single-period, single-item newsboy problem with limited distributional information (e.g., range, mean, mode, variance, symmetry) has been widely studied. However, the existing newsboy models with partial information are only fit to risk-neutral inventory managers. This paper considers the newsboy problem with range information. Based on the competitive ratio analysis, which guarantees a certain performance level under all possible input sequences, we construct a framework to manage risk and reward of newsboy problems under different forecasts (i.e. certain forecasts; probability forecasts; probability distributions). Comparing the existing studies, this approach helps the newsboy flexibly choose the optimal reward strategies, according to his own risk tolerance levels and different forecasts.

1 Introduction

The interest in the newsboy problem and its extensions has remained high since it was first introduced by Within [12]. Newsboy problem and its extensions have been presented and studied since then. These extensions include dealing with different objectives and utility functions, different supplier pricing policies, different newsboy pricing policies and discounting structures, different states of information about demand, constrained multi-products, multiple-products with substitution, random yields, and multi-location models.

Traditional newsboy models focus on risk-neutral decision makers, i.e., optimizing the expected profit or cost. But experimental findings state that the actual quantity ordered deviates from the optimal quantity derived from the classical newsboy model. In view of this, a number of papers have been devoted to risk analysis of newsboy problems [1,3,11,13].

Moreover, traditional newsboy models assume full knowledge of the demand probability distribution. However, in reality, it is often difficult to completely characterize the demand. Therefore, some researchers recently focused on the newsboy problem with partial (or limited) information. Savage [9] presented minimax regret. The aim of this approach is to minimize the maximum opportunity

* This research is supported by NSF of China (No. 70525004, 60736027).

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 334–345, 2009.

cost from not making the optimal decision. Using minimax absolute regret, Yue et al. [15] and Perakis and Roels [8] studied the newsboy problem with partial (or limited) information. Zhu et al. [16] used minimax relative regret to study the same problem. For most of the cases, two models based on minimax regret will tend to favor similar decisions. Perakis and Roels [8] also argued that the minimax regret was analogous to the competitive ratio [2,4,5,6,10,14], popular in computer science. Following the line of competitive ratio analysis, Ma et al. [7] investigated a newsboy-type model (i.e., the number of snacks problem) with partial information.

However, the existing newsboy models with partial information are fit to risk-neutral inventory managers. Range is a familiar format of partial information. This paper uses competitive ratio analysis to investigate the newsboy problem with range information. In newsboy problems with range information, the demand is uncertain and only has a support. The newsboy needs to determine the order quantity before the selling season. The main aim of this paper is to develop a framework for managing reward and risk of the newsboy problem with range information. The remains of this paper is organized as follows. In Section 2, we generalize the risk-reward model presented in Al-Binali [2], by introducing the notion of the probabilistic forecast. In Section 3, we respectively design the optimal reward strategies for the newsboy problem under certain forecasts, probability forecasts, probability distributions. In Section 4, we provide some numerical examples to illustrate the proposed approach. Finally, concluding remarks and future research are included in Section 5.

2 Competitive Ratio Analysis and the Extended Risk-Reward Model

Karp [6] and Sleator and Tarjian [10] proposed the concept of the competitive ratio to study on-line problems, by comparing the performance of on-line strategies to a benchmark (optimal off-line) strategy. During this classical competitive analysis, there are a strategy set S for the on-line decision-maker and an uncertain information set I dominated by the off-line opponent. The on-line decision-maker's goal is to design a good strategy $q \in S$ to deal with the uncertainty input sequence $\sigma \in I$ of the off-line rival. For a known sequence σ, let $Cost_{opt}(\sigma)$ be the total cost of the optimal off-line strategy to complete σ. For an on-line strategy q, if there are constants λ_q and ζ satisfying

$$Cost_q(\sigma) \leq \lambda_q Cost_{opt}(\sigma) + \zeta. \tag{1}$$

for any $\sigma \in I$, then q is called a λ_q-competitive strategy and λ_q is called the competitive ratio of q, where $Cost_q(\sigma)$ is the total cost taken with strategy q to complete σ. That is to say,

$$\lambda_q = \sup_{\sigma \in I} \frac{Cost_q(\sigma)}{Cost_{opt}(\sigma)}.$$

We denote $\lambda^* = \inf_{q \in S}(\lambda_q)$ as the optimal competitive ratio for the on-line problem. If $\lambda_{q^*} = \lambda^*$, then q^* is called the optimal on-line strategy.

The classic competitive analysis is the most fundamental and significant approach, yet it is not very flexible, especially in the economic management issues, many investors want to manage their risk and reward. Al-Binali [2] first defined the concepts of risk and reward for on-line financial problems. Al-Binali defined the risk of a strategy q to be $t_q = \frac{\lambda_q}{\lambda^*}$. The greater the value of t_q, the higher the risk of q. Let $F \subset I$ be a forecast, then denote

$$\overline{\lambda_q} = \sup_{\sigma \in F} \frac{Cost_q(\sigma)}{Cost_{opt}(\sigma)}$$

as the restricted competitive ratio of q restricted to cases when the forecast is correct. The optimal restricted competitive ratio under the forecast F is

$$\overline{\lambda^*} = \inf_{q \in S}(\overline{\lambda_q}).$$

When the forecast is correct, Al-Binali defined the reward of the strategy q to be $R_q = \lambda^*/\overline{\lambda_q}$.

The above reward definition is based on the certain forecast that is described to be a subset of I. When the forecast selected is correct, it will bring reward; otherwise bring risk. Dong et al. [4] extended the certain forecast to the probability forecast. Let $F_1, F_2, ..., F_m$ be a group of subsets of I, where $\bigcup F_i = I$ and $F_i \bigcap F_j = \phi$ for $i \neq j$. Denote P_i as the probability that the on-line decision maker anticipates that $\sigma \in F_i$, where $\sum_{i=1}^{m} P_i = 1$. We call $\{(F_i, P_i)|i = 1, 2, ..., m\}$ a probability forecast. Let

$$\overline{\lambda_{q,i}} = \sup_{\sigma \in F_i} \frac{Cost_q(\sigma)}{Cost_{opt}(\sigma)}$$

be the restricted competitive ratio under the forecast F_i. Let $R_{q,i} = \lambda^*/\overline{\lambda_{q,i}}$ be the reward after the success of the forecast F_i. Based on this, we define $\widetilde{\lambda_q} = \sum_{i=1}^{m} P_i \overline{\lambda_{q,i}}$ as the restricted competitive ratio under the probability forecast $\{(F_i, P_i)|i = 1, 2, ..., m\}$, and define $\widetilde{R_q} = \frac{\lambda^*}{\widetilde{\lambda_q}}$ as the reward under the probability forecast.

The reward definition based on the probability forecast has some desired properties [4].

Property 1. For any $q \in S$, $\min_i\{R_{q,i}\} \leq \widetilde{R_q} \leq \max_i\{R_{q,i}\}$.

Let $\{(F_i, P_i)|i = 1, 2, ..., m\}$ be a probability forecast. We divide F_i into $F_{i,1}$ and $F_{i,2}$, where $F_{i,1} \cup F_{i,2} = F_i$ and $F_{i,1} \bigcap F_{i,2} = \phi$. We also divide P_i into $P_{i,1}$ and $P_{i,2}$, where $P_{i,1} + P_{i,2} = P_i$. In this way, we can construct a more detailed probability forecast based on $\{(F_i, P_i)|i = 1, 2, ..., m\}$, that is $\{(F_1, P_1), (F_2, P_2), ..., (F_{i-1}, P_{i-1}), (F_{i,1}, P_{i,1}), (F_{i,2}, P_{i,2}), (F_{i+1}, P_{i+1}), ..., (F_m, P_m)\}$. Denote

$$\widetilde{\widetilde{R_q}} = \frac{\lambda^*}{\widetilde{\widetilde{\lambda_q}}}$$

as the reward under the newly constructed probability forecast.

Property 2. For any $q \in S$, $\widetilde{R_q} \leq \widetilde{\widetilde{R_q}}$.

Property 2 shows that if a probability forecast can be described more detailedly, the reward under the probability forecast will be greater.

Based on these newly introduced concepts, we propose a generalized risk-reward model under the probability forecast. If t is the risk tolerance level of the on-line decision maker (where $t \geq 1$ and higher values of t denote a higher risk tolerance), then denote $S_t = \{q|\lambda_q \leq t\lambda^*\}$ by the set of all strategies with the risk level $\leq t$. Our main aim is to look for an optimal reward strategy $q^* \in S_t$ that maximizes the reward under the probability forecast $\{(F_i, P_i)|i = 1, 2, ..., m\}$, that is

$$\widetilde{R_{q^*}} = \sup_{q \in S_t} \frac{\lambda^*}{\widetilde{\lambda_q}}.$$

The mathematic model can be described as follows.

$$\begin{cases} \max\limits_{q} \ \widetilde{R_q} = \frac{\lambda^*}{\widetilde{\lambda_q}} \\ \text{s.t.} \ \ \lambda_q \leq t\lambda^* \end{cases}. \tag{2}$$

On the other hand, let R be the reward level of the on-line decision maker (where $R \geq 1$ and higher values of R denote a higher reward level), and then denote $S_R = \{q|\widetilde{\lambda_q} \leq \lambda^*/R\}$ by the set of all strategies with the reward level $\geq R$ under the probability forecast $\{(F_i, P_i)|i = 1, 2, ..., m\}$. Our main task is to look for an optimal risk strategy $q^{*'} \in S_R$ that minimizes the risk, that is

$$\widetilde{t_{q^{*'}}} = \inf_{q \in S_R} \frac{\lambda_q}{\lambda^*}.$$

The mathematic model can be described as follows.

$$\begin{cases} \min\limits_{q} \ t_q = \frac{\lambda_q}{\lambda^*} \\ \text{s.t.} \ \ \widetilde{\lambda_q} \leq \frac{1}{R}\lambda^* \end{cases}. \tag{3}$$

Remark. The proposed framework is a generalization of the risk-reward model presented in Al-Binali [2], and has two new characters: (a) instead of certain forecasts, it considers probability forecasts; (b) under probability forecast environments, it formally presents two versions (i.e., the optimal reward strategy and the minimal risk strategy) of the risk-reward model to more flexibly manage the risk and reward in the financial and investment issues.

3 The Optimal Reward Order Strategies

In this section, we propose some familiar forecast formats of the newsboy problem with range information, such as basic certain forecast, basic probability forecast and probability distribution. We use model (2) to obtain the optimal reward order strategies under these forecasts.

3.1 Certain Forecast

In newsboy problems with range information, the demand z is uncertain and only has a support $[m, M]$. We denote c as the unit ordering cost, b as the unit inventory holding cost, and l as the unit stock-out cost. The newsboy needs to determine the order quantity q before the selling season. To avoid unrealistic and trivial cases, we assume that $l \geq c > 0$ and $b > 0$. Obviously, this problem is a typical on-line problem.

For off-line newsboy problem, the newsboy chooses demand z as order quantity to minimize its cost, that is,

$$Cost_{opt}(z) = cz \tag{4}$$

For on-line newsboy problem, we have that

$$Cost_q(z) = cq + l(z-q)^+ + b(q-z)^+, \tag{5}$$

where $(z-q)^+ = \max\{z-q, 0\}$ and $(q-z)^+ = \max\{q-z, 0\}$.

Lemma 1. *For the online newsboy problem, if the newsboy knows the lower and upper bounds of the demand, namely, the two extreme demand M and m, the optimal on-line order strategy is*

$$q^* = \frac{Mm(b+l)}{M(b+c) + m(l-c)}$$

and the optimal competitive ratio equals

$$\lambda_{q^*} = \frac{Ml(b+c) - mb(l-c)}{Mc(b+c) + mc(l-c)}.$$

Proof. Let

$$f(z,q) = \frac{Cost_q(z)}{Cost_{opt}(z)} = \frac{cq + l(z-q)^+ + b(q-z)^+}{cz}, \tag{6}$$

According to the definition of the competitive ratio, we know that

$$\begin{aligned} \lambda_q &= \sup_{z \in I} f(z,q) \\ &= \max\{\sup_{z \geq q} f(z,q), \sup_{z \leq q} f(z,q)\} \\ &= \max\{f_+(q), f_-(q)\} \end{aligned} \tag{7}$$

where $f_+(q) = \sup_{z \geq q} f(z,q)$ and $f_-(q) = \sup_{z \leq q} f(z,q)$.

Based on monotony properties of $f_+(q)$ and $f_-(q)$, we get

$$f_+(q) = \frac{cq + l(M-q)}{cM}, \ f_-(q) = \frac{cq + b(q-m)}{cm}. \tag{8}$$

To minimize $\lambda_q = \max\{f_+(q), f_-(q)\}$, we need to find the intersection point of $f_+(q)$ and $f_-(q)$, which is the optimal order strategy we need.

Denote q^* satisfying $f_+(q^*) = f_-(q^*)$, we have

$$q^* = \frac{Mm(b+l)}{M(b+c)+m(l-c)}. \tag{9}$$

Applying (7), (8) and (9), we obtain the optimal competitive ratio

$$\lambda_{q^*} = \inf_{q\in[m,M]} \lambda_q = \frac{Ml(b+c)-mb(l-c)}{Mc(b+c)+mc(l-c)} \tag{10}$$

This completes the proof of Lemma 1. □

Lemma 2. *When setting the risk tolerance level $t \geq 1$, the feasible strategy set of online newsboy problem with range information is $S_t = \{q|\ q_- \leq q \leq q_+\}$, where*

$$q_- = \max\left\{\frac{Mm(tb+l)(l-c)-(t+1)M^2l(b+c)}{(l-c)[M(b+c)+m(l-c)]}, m\right\},$$

and

$$q_+ = \min\left\{\frac{Mm(tl+b)(b+c)-(t-1)m^2l(l-c)}{(b+c)[M(b+c)+m(l-c)]}, M\right\}.$$

Proof. According to the definition of the risk, the strategy set is $S_t = \{q|\ \lambda_q \leq t\lambda_{q^*}\}$. Applying (7), (8) and (10), we have

$$\lambda_q \leq t\lambda_{q^*},$$

that is

$$\max\{f_+(q), f_-(q)\} \leq t\frac{Ml(b+c)-mb(l-c)}{Mc(b+c)+mc(l-c)}.$$

Consequently,

$$\begin{cases} \frac{cq+l(M-q)}{cM} \leq t\frac{Ml(b+c)-mb(l-c)}{Mc(b+c)+mc(l-c)} \\ \frac{cq+b(q-m)}{cm} \leq t\frac{Ml(b+c)-mb(l-c)}{Mc(b+c)+mc(l-c)} \\ m \leq q \leq M \end{cases}$$

Further, we have $q_- \leq q \leq q_+$, where

$$q_- = \max\{\frac{Mm(tb+l)(l-c)-(t-1)M^2l(b+c)}{(l-c)[M(b+c)+m(l-c)]}, m\},$$

and

$$q_+ = \min\{\frac{Mm(tl+b)(b+c)-(t-1)m^2l(l-c)}{(b+c)[M(b+c)+m(l-c)]}, M\}.$$

This completes the proof of Lemma 2. □

We analyze the newsboy problem with range information under the risk-reward framework presented in Al-Binali [2]. We consider the following forecast: $F = \{z|m^{'} \leq z \leq M^{'}\}$, where $m^{'} > m$ and $M^{'} \leq M$. In this paper, we call F a basic certain forecast.

Theorem 1. *When setting the risk tolerance level $t \geq 1$, the optimal reward order strategy under the basic certain forecast F is*

$$q^* = \min\Big\{ \max\{\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}, q_-\}, q_+\Big\} \tag{11}$$

Proof. The restricted competitive ratio under forecast F is

$$\overline{\lambda_q} = \sup_{z\in F} f(z,q) = \max\Big\{\frac{cq+l(M^{'}-q)}{cM^{'}}, \frac{cq+b(q-m^{'})}{cm^{'}}\Big\}.$$

Analyzing $\overline{\lambda_q}$, we know $\overline{\lambda_q}$ is monotony decreasing at $q \leq \frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}$, and monotony increasing at $q > \frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}$.

From the above monotony properties of $\overline{\lambda_q}$ and Lemma 2, we can look for the optimal reward strategy $q^* \in S_t$ that makes $\overline{\lambda_q}$ minimum, that is equations (11). This completes the proof of Theorem 1. □

3.2 Probability Forecast

We construct the following probability forecast:

Forecast F_1: $F_1 = \{k | m^{'} \leq k \leq M^{'}\}$, where $m^{'} > m$ and $M^{'} < M$. The probability when F_1 appears is P_1.

Forecast F_2: $F_2 = \{k | m \leq k < m^{'}$ *or* $M^{'} < k \leq M\}$. The probability when F_2 appears is P_2, where $P_1 + P_2 = 1$.

In this paper, we call $\{(F_1, P_1), (F_2, P_2)\}$ the basic probability forecast.

Theorem 2. *When setting the risk tolerance level $t \geq 1$, under the basic probability forecast $\{(F_1, P_1), (F_2, P_2)\}$ we have*

(1) *when* $\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)} < \frac{Mm(b+l)}{M(b+c)+m(l-c)}$, *the optimal reward order strategy is*

$$q^* = \begin{cases} \frac{Mm(b+l)}{M(b+c)+m(l-c)}, & \text{if } P_1 < \frac{m^{'}(l-c)}{M(b+c)+m^{'}(l-c)} \\ \Big[\max\{\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}, q_-\}, \frac{Mm(b+l)}{M(b+c)+m(l-c)})\Big] & \text{if } P_1 = \frac{m^{'}(l-c)}{M(b+c)+m^{'}(l-c)} \\ \max\{\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}, q_-\}, & \text{if } P_1 > \frac{m^{'}(l-c)}{M(b+c)+m^{'}(l-c)} \end{cases} \tag{12}$$

(2) *when* $\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)} \geq \frac{Mm(b+l)}{M(b+c)+m(l-c)}$, *the optimal reward order strategy is*

$$q^* = \begin{cases} \frac{Mm(b+l)}{M(b+c)+m(l-c)}, & \text{if } P_1 < \frac{M^{'}(b+c)}{m(l-c)+M^{'}(b+c)} \\ \Big[\frac{Mm(b+l)}{M(b+c)+m(l-c)}, \min\{\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}, q_+\}\Big] & \text{if } P_1 = \frac{M^{'}(b+c)}{m(l-c)+M^{'}(b+c)} \\ \min\{\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}, q_+\}, & \text{if } P_1 > \frac{M^{'}(b+c)}{m(l-c)+M^{'}(b+c)} \end{cases} \tag{13}$$

Proof. We only discuss the case that $\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)} < \frac{Mm(b+l)}{M(b+c)+m(l-c)}$, and the case that $\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)} \geq \frac{Mm(b+l)}{M(b+c)+m(l-c)}$ can be discussed similarly.

We first compute the restricted competitive ratio under the basic probability forecast

$$\widetilde{\lambda_q} = \sum_{i=1}^{2} P_i \overline{\lambda_{q_i}}, \text{ where } \overline{\lambda_{q_i}} = \sup_{z \in F_i} \Big\{ \frac{Cost_q(z)}{Cost_{opt}(z)} \Big\}.$$

Consequently,

$$\widetilde{\lambda_q} = P_1 \overline{\lambda_{q_1}} + P_2 \overline{\lambda_{q_2}} = P_1 \sup_{z \in F_1} f(z,q) + P_2 \sup_{z \in F_2} f(z,q),$$

which can be formulated as

$$\widetilde{\lambda_q} = \begin{cases} P_1 \frac{cq+l(M^{'}-q)}{cM^{'}} + P_2 \frac{cq+l(M-q)}{cM}, & \text{if } q < \frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)} \\ P_1 \frac{cq+b(q-m^{'})}{cm^{'}} + P_2 \frac{cq+l(M-q)}{cM}, & \text{if } \frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)} \leq q \leq \frac{Mm(b+l)}{M(b+c)+m(l-c)} \\ P_1 \frac{cq+b(q-m^{'})}{cm^{'}} + P_2 \frac{cq+b(q-m)}{cm}, & \text{if } q > \frac{Mm(b+l)}{M(b+c)+m(l-c)} \end{cases}$$

Solving $\frac{\partial \widetilde{\lambda_q}}{\partial q}$, we find that

(a) when $P_1 < \frac{m^{'}(l-c)}{M(b+c)+m^{'}(l-c)}$, $\widetilde{\lambda_q}$ is monotony decreasing at $q \leq \frac{Mm(b+l)}{M(b+c)+m(l-c)}$, and monotony increasing at $q > \frac{Mm(b+l)}{M(b+c)+m(l-c)}$;

(b) when $P_1 = \frac{m^{'}(l-c)}{M(b+c)+m^{'}(l-c)}$, $\widetilde{\lambda_q}$ is monotony decreasing at $q < \frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}$, constant at $\frac{m^{'}(l-c)}{M(b+c)+m^{'}(l-c)} \leq q \leq \frac{Mm(b+l)}{M(b+c)+m(l-c)}$ and monotony increasing at $q < \frac{Mm(b+l)}{M(b+c)+m(l-c)}$;

(c) when $P_1 > \frac{m^{'}(l-c)}{M(b+c)+m^{'}(l-c)}$, $\widetilde{\lambda_q}$ is monotony decreasing at $q \leq \frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}$, and monotony increasing at $q > \frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}$.

From the monotony properties of $\widetilde{\lambda_q}$ and Lemma 2, we can look for the optimal reward order strategy, that is equations (12). This completes the proof of Theorem 2. □

Corollary 1. *When* $P_1 = 1$, $q^* = \min\{\max\{\frac{M^{'}m^{'}(b+l)}{M^{'}(b+c)+m^{'}(l-c)}, q_-\}, q_+\}$; *When* $P_1 = 0$, $q^* = \frac{Mm(b+l)}{M(b+c)+m(l-c)}$.

Corollary 1 shows that probability forecast is a generalization of certain forecast.

3.3 Probability Distribution

By dividing F_i into $F_{i,1}$ and $F_{i,2}$ (where $F_{i,1} \cup F_{i,2} = F_i$ and $F_{i,1} \bigcap F_{i,2} = \phi$), and dividing P_i into $P_{i,1}$ and $P_{i,2}$ (where $P_{i,1} + P_{i,2} = P_i$), we construct a more

detailed probability forecast based on $\{(F_i, P_i)|i = 1, 2, ..., m\}$, that is $\{(F_1, P_1), (F_2, P_2), ..., (F_{i-1}, P_{i-1}), (F_{i,1}, P_{i,1}), (F_{i,2}, P_{i,2}), (F_{i+1}, P_{i+1}), ..., (F_m, P_m)\}$. For the newsboy problem, we obtain the probability distribution of the demand z, when repeatedly dividing $\{(F_1, P_1), (F_2, P_2)\}$ in the above way. Thus, probability distributions are special cases of probability forecasts. Let the demand be a stochastic variable z, which is drawn from a known probability distribution with probability density function $g(z)$, with support $[m, M]$. The restricted competitive ratio under the probability distribution $g(z)$ is

$$\widetilde{\widetilde{\lambda}}_q = E_z \frac{Cost_q(z)}{Cost_{opt}(z)} = \int_m^M \frac{Cost_q(z)}{Cost_{opt}(z)} g(z). \tag{14}$$

According to (4), (5) and (14), we have that

$$\widetilde{\widetilde{\lambda}}_q = \frac{c+b}{c} \int_m^q \frac{q}{z} g(z)dz - \frac{b}{c} \int_m^q g(z)dz + \frac{c-l}{c} \int_q^M \frac{q}{z} g(z)dz + \frac{l}{c} \int_q^M g(z)dz.$$

Theorem 3. *When setting the risk tolerance level $t \geq 1$, under the probability distribution $g(z)$, the optimal reward order strategy is*

$$q^* = \min\{\max\{q_0, q_-\}, q_+\}, \tag{15}$$

where q_0 satisfies $\frac{c+b}{c} \int_m^{q_0} \frac{1}{z} g(z)dz + \frac{c-l}{c} \int_{q_0}^M \frac{1}{z} g(z)dz = 0$.

Proof. Differentiating $\widetilde{\widetilde{\lambda}}_q$, we get that

$$\frac{\partial \widetilde{\widetilde{\lambda}}_q}{\partial q} = \frac{c+b}{c} \int_m^q \frac{1}{z} g(z)dz + \frac{c-l}{c} \int_q^M \frac{1}{z} g(z)dz,$$

and

$$\frac{\partial^2 \widetilde{\widetilde{\lambda}}_q}{\partial q^2} = \frac{b+l}{c} \frac{g(q)}{q}.$$

It is easy to verify that

$$\lim_{q \to m} \frac{\partial \widetilde{\widetilde{\lambda}}_q}{\partial q} = \frac{c-l}{c} \int_m^M \frac{1}{z} g(z)dz < 0,$$

$$\lim_{q \to M} \frac{\partial \widetilde{\widetilde{\lambda}}_q}{\partial q} = \frac{c+b}{c} \int_m^M \frac{1}{z} g(z)dz > 0,$$

and

$$\lim_{q \to M} \frac{\partial^2 \widetilde{\widetilde{\lambda}}_q^2}{\partial q} = \frac{b+l}{c} \frac{g(q)}{q} > 0.$$

Therefore, there is a unique q_0 satisfying

$$\frac{\partial \widetilde{\widetilde{\lambda}}_q}{\partial q}\Big|_{q=q_0} = 0.$$

Based on Lemma 2, we can find the optimal reward order strategy that makes $\widetilde{\widetilde{\lambda_q}}$ minimum, which is equation (15). This completes the proof. □

Since the expression

$$\frac{\partial \widetilde{\widetilde{\lambda_q}}}{\partial q}\Big|_{q=q_0} = 0$$

is a non-linear equation on q, we have difficulty in obtaining an analytic representation of q_0. In general, we may use the dichotomous search algorithm to determine the value q_0 in the polynomial time $O(nlog(n))$.

From Theorem 3, we have Corollary 2.

Corollary 2. *When setting the risk tolerance level $r \geq 1$, under the uniform distribution,*

$$f(z) = \begin{cases} \frac{1}{M-m}, & m \leq z \leq M \\ 0, & other \end{cases}$$

the optimal reward order strategy is

$$q^* = \min\{\max\{e^{\frac{(l-c)lnM+(b+c)lnm}{b+l}}, q_-\}, q_+\}.$$

4 Numerical Examples

In this section, we present numerical examples to illustrate our approach. Consider the following newsboy example:

$$c = 3, \quad b = 1, \quad l = 6$$

For different value of the fixed ordering support $[m, M]$, we calculate the optimal on-line order strategy and the optimal competitive ratio, which are summarized in Table 1.

Table 1. The optimal order strategies with support $[m, M]$

$[m, M]$	the optimal order strategies q^*	the optimal competitive ratio λ_{q*}
$[100, 200]$	127.27	1.36
$[100, 300]$	140	1.53
$[100, 400]$	147.34	1.63
$[100, 500]$	152.17	1.7

Table 1 shows the more uncertain on demand, namely the greater the ratio of upper/lower on demand, the greater the optimal competitive ratio of the newsboy problem. To improve the optimal competitive ratio, the newsboy may designs the optimal reward order strategies under the established forecasts on input sequence of the off-line rival. Without loss of generality, we set $t = 1.3$, and then design the optimal reward strategies under the basic certain forecast, the basic probabilistic forecast and the uniform distribution respectively. The results are listed in Table 2, Table 3 and Table 4.

Table 2. The optimal reward order strategies under certain forecast

$[m, M]$	S_t	$[m^{'}, M^{'}]$	q^*
[100, 200]	[100, 147.73]	[110, 150]	124.2
[100, 300]	[100, 167]	[120, 160]	134.4
[100, 400]	[100, 178.16]	[160, 300]	178.16
[100, 500]	[100, 185.43]	[200, 400]	185.43

Table 3. The optimal order strategies under probabilistic forecast

$[m, M]$	S_t	$[m^{'}, M^{'}]$	P_1	q^*
[100, 200]	[100, 147.73]	[110, 150]	0.2	127.27
			0.6	124.2
[100, 300]	[100, 167]	[120, 160]	0.2	140
			0.6	134.4
[100, 400]	[100, 178.16]	[160, 300]	0.5	147.34
			0.9	178.16
[100, 500]	[100, 185.43]	[200, 400]	0.5	152.17
			0.9	185.43

Table 4. The optimal reward order strategies under the uniform distribution forecast

$[m, M]$	S_t	q^*
[100, 200]	[100, 147.73]	134.59
[100, 300]	[100, 167]	160.13
[100, 400]	[100, 178.16]	178.16
[100, 500]	[100, 185.43]	185.43

5 Conclusions

Traditional newsboy models assume full knowledge of the demand probability distribution. Recently, some researchers [15,8,16,7] focused on the newsboy problem with partial information (e.g., rang, mean, mode, variance, symmetry). The models given by Yue et al. [15] and Perakis and Roels [8] were to minimize the absolute regret, the model given by Zhu et al. [16] was to minimize the relative regret, and the model given by Ma et al. [7] used the competitive ratio analysis. Perakis and Roels [8] noted that the minimax regret was analogous to the competitive ratio. However, the existing newsboy models with partial information are fit to risk-neutral inventory managers. This paper analyzes the newsboy problem from the line of competitive ratio. Based on the approach presented in Al-Binali [2], we further develop a framework to manage reward and risk of the newsboy problem with range information. This approach can help the newsboy flexibly choose the optimal reward strategies, according to his own risk tolerance levels and different forecasts.

The proposed approach is a generic approach to manage reward and risk, so it also can be applied to other cases of newsboy problem. Moreover, since the minimax regret is analogous to the competitive ratio, we easily develop a similar framework based on the minimax regret. A comparative study of these two research lines may be an interesting future research.

References

1. Agrawal, P., Ganeshan, R.: Using Risk-Management Tools on B2Bs: an Exploratory Investigation. International Journal of Production Economics 108, 2–7 (2007)
2. Al-Binali, S.: A Risk-Reward Framework for the Competitive Analysis of Financial Games. Algorithmica 25, 99–115 (1999)
3. Bogataj, D., Bogataj, M.: Measuring the Supply Chain Risk and Vulnerability in Frequency Space. International Journal of Production Economics 108, 291–301 (2007)
4. Dong, Y.C., Xu, Y.F., Xu, W.J.: The On-Line Rental Problem with Risk and Probabilistic Forecast. In: Preparata, F.P., Fang, Q. (eds.) FAW 2007. LNCS, vol. 4613, pp. 117–123. Springer, Heidelberg (2007)
5. Fujiwara, H., Iwama, K.: Average-Case Competitive Analyses for Ski-Rental Problems. Algorithmica 42, 95–107 (2005)
6. Karp, R.: Online Algorithms Versus Offline Algorithms: How Much is it Worth to Know the Future? In: Proc. IFIP 12th World Computer Congress, pp. 416–429 (1992)
7. Ma, W.M., You, J., Xu, Y.F., Liu, M., Wang, K.L.: On the On-Line Number of Snacks Problem. Journal of Global Optimization 24, 449–462 (2002)
8. Perakis, G., Roels, G.: Regret in the Newsvendor Model with Partial Information. Operations Research 56, 188–203 (2008)
9. Savage, L.: The Theory of Statistical Decision. Journal of the American Statistical Association 46, 55–67 (1951)
10. Sleator, D.D., Tarjan, R.E.: Amortized Efficiency of List Update and Paging Rules. Communications of the ACM 28, 202–208 (1985)
11. Sounderpandian, J., Prasad, S., Madan, M.: Supplies from Developing Countries: Optimal Rrder Quantities under Loss Risks. Omega 36, 122–130 (2008)
12. Within, T.M.: Inventory Control and Price Theory. Management Science 2, 61–80 (1955)
13. Xiao, T., Qi, X.: Price Competition, Cost and Demand Disruptions and Coordination of a Supply Chain with One Manufacturer and Two Competing Retailers. Omega 36, 741–753 (2008)
14. Xu, Y.F., Xu, W.J., Li, H.Y.: On the On-Line Rent-or-Buy Problem in Probabilistic Environments. Journal of Global Optimization 38, 1–20 (2007)
15. Yue, J.F., Chen, B.T., Wang, M.C.: Expected Value of Distribution Information for the Newsvendor Problem. Operations Research 54, 1128–1136 (2006)
16. Zhu, Z., Zhang, J., Ye, Y.: Newsvendor Optimization with Limited Distribution Information. Working Paper, Stanford University, Stanford, CA (2006)

Optimal Semi-online Algorithm for Scheduling on a Batch Processing Machine

Ming Liu[1,2,⋆], Yinfeng Xu[1], Chengbin Chu[1,2], and Lu Wang[3]

[1] School of Management, Xi'an Jiaotong University
Xi'an, Shaanxi Province 710049, China
minyivg@gmail.com
[2] Laboratoire Génie Industriel, Ecole Centrale Paris, Grande Voie des Vignes
92295 Châtenay-Malabry Cedex, France
[3] Shanghai Vocational School of CAAC
Shanghai 200232, China

Abstract. We consider two semi-online scheduling problems on a single batch (processing) machine with jobs' nondecreasing processing times and jobs' nonincreasing processing times, respectively. Our objective is to minimize the makespan. A batch processing machine can handle up to B jobs simultaneously. We study an unbounded model where $B = \infty$. The jobs that are processed together construct a batch, and all jobs in a batch start and complete at the same time. The processing time of a batch is given by the longest processing time of any job in the batch. Jobs arrive over time. Let p_j denote the processing time of job J_j. Given job J_j and its following job J_{j+1}, we assume that $p_{j+1} \geq \alpha p_j$, where $\alpha \geq 1$ is a constant number, for the first problem with jobs' nondecreasing processing times. For the second problem, we assume that $p_{j+1} \leq \alpha p_j$, where $0 < \alpha < 1$ is a constant number. We propose an optimal algorithm for both problems with a competitive ratio $\frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$ for the first problem and $\frac{\sqrt{4\alpha+1}+1}{2}$ for the second problem.

Keywords: Online scheduling, Batch processing, Makespan.

1 Introduction

Batch processing machine scheduling has been motivated by burn-in operations in the final testing stage of semiconductor manufacturing in [7] and [6]. Batch scheduling means that a machine can process up to B jobs simultaneously as a batch, and the processing time of a batch is equal to the longest processing time of the jobs assigned to it. Unbounded model means that B is sufficiently large and the bounded model means that B is a constant positive integer.

In recent years, one of the basic assumptions made in deterministic scheduling was that all the useful information of the problem instance was known in advance. However, this assumption is usually not realistic. This reason promotes

⋆ Corresponding author.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 346–353, 2009.

the emergence of online scheduling. Three online models are commonly considered in [5]. The first one assumes that there are no release times and that the jobs arrive in a list (one by one). The online algorithm has to schedule (or assign) the first job in this list before it sees the next job in the list. The second model assumes that the running time of a job is unknown until the job finishes. The online algorithm knows whether a job is still running or not. The third model assumes that jobs arrive over time. At each time when the machine is idle, the online algorithm decides which one of the available jobs is scheduled, if any. When all information is available at the beginning (before scheduling), the problem is called *offline*.

We use the competitive analysis [1] to measure the performance of an online algorithm. For any input job sequence I, let $C_{ON}(I)$ denote the objective (function) value of the schedule produced by the online algorithm $\mathcal{A}_{ON}$ and $C_{OPT}(I)$ denote the objective value of the optimal schedule. We say that $\mathcal{A}_{ON}$ is ρ-competitive if

$$\rho = \sup\left\{\frac{C_{ON}(I)}{C_{OPT}(I)}\right\}.$$

We also say that ρ is the competitive ratio of $\mathcal{A}_{ON}$. An algorithm is called optimal if the competitive ratio of this algorithm matches the lower bound of competitive ratio for all online algorithms.

In this paper, we consider the third model where jobs arrive over time. Zhang et al. [10] considered the problem of online scheduling on a batch processing machine. They provided an optimal online algorithm with a competitive ratio $\frac{\sqrt{5}+1}{2}$. Based on their results, we study the semi-online problems by using additional jobs' information. We present an optimal semi-online algorithm with a competitive ratio no more than $\frac{\sqrt{5}+1}{2}$. Let p_j denote the processing time of job J_j. Given job J_j and its following job J_{j+1}, we assume that $p_{j+1} \geq \alpha p_j$, $\alpha \geq 1$ is a constant number, in the first problem with jobs' nondecreasing processing times. For the second problem, we assume that $p_{j+1} \leq \alpha p_j$, $0 < \alpha < 1$ is a constant number.

Online scheduling on a (parallel) batch processing machine has been studied in the last decade. In the unbounded model, for the problem of online scheduling on a single batch machine to minimize the makespan, Zhang et al. [10] and Deng et al. [2] independently provided an online algorithm with a competitive ratio $\frac{\sqrt{5}+1}{2}$. Poon et al. [9] showed that for the same problem in the bounded model, any FBLPT-based (Full Batch Longest Processing Time) algorithm is 2-competitive. Moreover, they presented an algorithm with a competitive ratio $\frac{7}{4}$ for batch size $B = 2$. If a batch is allowed to restart, Fu et al. [4] showed that for minimizing makespan on an unbounded batch machine there is no algorithm with a competitive ratio less than $\frac{5-\sqrt{5}}{2}$. Further, they provided an online algorithm with a competitive ratio $\frac{3}{2}$. For this problem, Fu et al. [3] further considered limited restarts which means that any batch containing a job has already been restarted once cannot be restarted any more. They present an optimal algorithm with a competitive ratio $\frac{3}{2}$. Recently, Nong et al. [8] considered family jobs constraint in the single batch machine scheduling problem. For the unbounded

case, they provided an optimal online algorithm with a competitive ratio 2. For the bounded case, they gave a 2-competitive algorithm.

For convenience, we use online algorithm to denote semi-online algorithm in the remainder. The rest of this paper is organized as follows. In Section 2, we give the problem definitions and some notations. In Section 3, we present a lower bound $\varphi = \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$ ($\alpha \geq 1$) for the first problem and a lower bound $\varphi = \frac{\sqrt{4\alpha+1}+1}{2}$ ($0 < \alpha < 1$) for the second problem, respectively. In Section 4, we use the lower bounds obtained in the above section to design algorithm H_α^∞. After that, we prove that this algorithm is optimal for both problems.

2 Problem Definitions and Notations

We are given a job instance $\mathcal{I} = \{J_1, ..., J_n\}$ ($n \geq 2$) where each job J_j is associated with a release time r_j and a processing time p_j. The jobs are to be processed by a batch processing machine of capacity $B = \infty$, i.e., we study unbounded model. The processing time of a batch is the longest processing time of any job in the batch. Jobs arrive over time, i.e., each job's character, such as processing time, becomes known at its arrival. Let p_j denote the processing time of job J_j. Let J_{i+1} denote the following job of J_i. In the first problem with jobs' nondecreasing processing times, we assume that $p_{j+1} \geq \alpha p_j$, where $\alpha \geq 1$ is a constant number. For the second problem, we assume that $p_{j+1} \leq \alpha p_j$, where $0 < \alpha < 1$ is a constant number. Our objective is to minimize the makespan. Using three field notations, our problems can be denoted by $1|online, r_j, B = \infty, nondecreasing|C_{max}$ and $1|online, r_j, B = \infty, nonincreasing|C_{max}$, respectively.

We use $U(t)$ to denote the set of unscheduled jobs available at time t.

3 Lower Bounds for Both Problems

In this section, we deal with the problems of online scheduling on a batch machine with jobs' nondecreasing processing times and jobs' nonincreasing processing times, respectively. We study unbounded model, i.e., the batch's size is sufficiently large. We first respectively give a lower bound of competitive ratio for each problem, then we provide an optimal algorithm for both problems. Let $\varphi = \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$ when $\alpha \geq 1$ and $\varphi = \frac{\sqrt{4\alpha+1}+1}{2}$ when $0 < \alpha < 1$. Note that $1 < \varphi \leq \frac{\sqrt{5}+1}{2}$ and we will use it in the remainder for simplicity.

Given an job instance $\mathcal{I}$, let C_{ON} and C_{OPT} denote the makespan obtained by an online algorithm $\mathcal{A}_{ON}$ and the value of an optimal (offline) schedule, respectively.

Lemma 1. *For $1|online, r_j, B = \infty, nondecreasing|C_{max}$, there is no algorithm with a competitive ratio less than $\frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$, where $\alpha \geq 1$.*

Proof. For any online algorithm A_{ON}, we construct a special job instance $\mathcal{I}$ such that C_{ON}/C_{OPT} is as large as possible. Let ϵ be a sufficiently small positive

number. At time 0, we give the first job J_1 with $p_1 = 1$. We assume that A_{ON} schedules this job at time $T_{\mathcal{A}}$. Depending on $T_{\mathcal{A}}$, we discuss the following two cases.

Case 1. $T_{\mathcal{A}} \geq \frac{\sqrt{\alpha^2+4}-\alpha}{2}$.
No job arrives in future. We know that $C_{ON} \geq \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$. An optimal scheme consists of scheduling J_1 at time 0, i.e., $C_{OPT} = 1$. Therefore,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1 = \varphi.$$

Case 2. $0 \leq T_{\mathcal{A}} < \frac{\sqrt{\alpha^2+4}-\alpha}{2}$.
We further generate job J_2 with $p_2 = \alpha$ at time $T_{\mathcal{A}} + \epsilon$. We obtain $C_{ON} \geq T_{\mathcal{A}} + 1 + \alpha$. An optimal schedule consists of scheduling J_1 and J_2 in a batch at time $T_{\mathcal{A}} + \epsilon$. Therefore, $C_{OPT} = T_{\mathcal{A}} + \epsilon + \alpha$ due to $\alpha \geq 1$. It follows that

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{T_{\mathcal{A}} + 1 + \alpha}{T_{\mathcal{A}} + \epsilon + \alpha} = 1 + \frac{1}{T_{\mathcal{A}} + \alpha} > 1 + \frac{1}{\frac{\sqrt{\alpha^2+4}-\alpha}{2} + \alpha} = \varphi, \quad \epsilon \to 0.$$

According to the above two cases, the lemma holds.

Lemma 2. *For $1|online, r_j, B = \infty, nonincreasing|C_{max}$, there is no algorithm with a competitive ratio less than $\frac{\sqrt{4\alpha+1}+1}{2}$, where $0 < \alpha < 1$.*

Proof. For any online algorithm A_{ON}, we construct a special job instance $\mathcal{I}$ such that C_{ON}/C_{OPT} is as large as possible. Let ϵ be a sufficiently small positive number. At time 0, we give the first job J_1 with $p_1 = 1$. We assume that A_{ON} scheduling this job at time $T_{\mathcal{A}}$. Depending on $T_{\mathcal{A}}$, we discuss the following two cases.

Case 1. $T_{\mathcal{A}} \geq \frac{\sqrt{4\alpha+1}-1}{2}$.
No job arrives in future. We know that $C_{ON} \geq \frac{\sqrt{4\alpha+1}+1}{2}$. An optimal scheme consists of scheduling J_1 at time 0, i.e., $C_{OPT} = 1$. Therefore,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{\sqrt{4\alpha+1}+1}{2} = \varphi.$$

Case 2. $0 \leq T_{\mathcal{A}} < \frac{\sqrt{4\alpha+1}-1}{2}$.
We further give job J_2 with $p_2 = \alpha$ at time $T_{\mathcal{A}} + \epsilon$. We obtain $C_{ON} \geq T_{\mathcal{A}} + 1 + \alpha$. An optimal schedule consists of scheduling J_1 and J_2 in a batch at time $T_{\mathcal{A}} + \epsilon$. Therefore, $C_{OPT} = T_{\mathcal{A}} + \epsilon + 1$ due to $\alpha < 1$. It follows that

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{T_{\mathcal{A}} + 1 + \alpha}{T_{\mathcal{A}} + \epsilon + 1} = 1 + \frac{\alpha}{T_{\mathcal{A}} + 1} > 1 + \frac{\alpha}{\frac{\sqrt{4\alpha+1}-1}{2} + 1} = \varphi, \quad \epsilon \to 0.$$

According to the above cases, the lemma holds.

4 An Optimal Algorithm for Both Problems

Considering the proofs of two lower bounds of competitive ratios, we realize that the algorithm should wait for a while rather than scheduling the jobs immediately after their arrivals. We shift the release time of job J_j to $\overline{r}_j = \varphi r_j + (\varphi - 1)p_j$ in order to contain the waiting time. This equation can guarantee that if J_j is scheduled at time $\varphi r_j + (\varphi - 1)p_j$, its completion time cannot exceed φ times the optimal value, i.e., $\varphi r_j + (\varphi - 1)p_j + p_j = \varphi(r_j + p_j)$. This idea is used in [10]. We design the following algorithm using the same idea.

Algorithm H_α^∞ works as follows.

Step 1: Wait until a decision point, where the batch machine is idle and at least one job is available (If all jobs have been scheduled, output the schedule). Suppose this happens at time t. Choose a job J_j with the longest processing time in $U(t)$.
Step 2: If $\overline{r}_j \leq t$, schedule all jobs in $U(t)$ as a single batch;
otherwise, wait until a new job arrive or until time $\overline{r}_j$, whichever happens first.
Step 3: Go to **Step 1**.

We adopt some notations and definitions from [10]. Given an job instance, we assume that H_α^∞ generate m batches in total. We index these batches in nondecreasing order of their completion times. For convenience, in batch i, denote by $J_{(i)}$ the job with the longest processing time in that batch. Let $p_{(i)}$ and $r_{(i)}$ be the processing time and the release time (or arrival time) of job $J_{(i)}$, respectively. Let $s_{(i)}$ be the starting time of batch i. Note that batch i is processed either at time $\varphi r_{(i)} + (\varphi - 1)p_{(i)}$ or after batch $i - 1$ is finished at time $s_{(i-1)} + p_{(i-1)}$.

If batch i starts at time $\varphi r_{(i)} + (\varphi - 1)p_{(i)}$, we call it a *regular* batch; otherwise, it is called a *delayed* batch.

Similar to Lemma 2 in [10], we have the following lemma.

Lemma 3. *If batch i is a regular batch, then batch $i+1$ or batch $i+2$ is also a regular batch.*

Proof. We prove this lemma for two cases: $\alpha \geq 1$ and $0 < \alpha < 1$.

(1) $\alpha \geq 1$.
By contradiction. Suppose both batches $i+1$ and $i+2$ are delayed batches. We know that each job in batch $i+1$ arrives later than the starting time of batch i. Therefore, considering job $J_{(i+1)}$, we have $r_{(i+1)} > \varphi r_{(i)} + (\varphi - 1)p_{(i)}$. Since batch $i+1$ is a delayed batch, we obtain

$$\varphi(r_{(i)} + p_{(i)}) > \varphi r_{(i+1)} + (\varphi - 1)p_{(i+1)} > \varphi^2 r_{(i)} + \varphi(\varphi - 1)p_{(i)} + (\varphi - 1)p_{(i+1)}.$$

It follows that

$$(2\varphi - \varphi^2)p_{(i)} > (\varphi^2 - \varphi)r_{(i)} + (\varphi - 1)p_{(i+1)} \geq (\varphi - 1)p_{(i+1)}. \tag{1}$$

Considering $\varphi = \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$ and $\alpha \geq 1$, we have

$$p_{(i+1)} < \frac{2\varphi - \varphi^2}{\varphi - 1} p_{(i)} = \left[\frac{1}{\varphi - 1} - (\varphi - 1)\right] p_{(i)} = \alpha p_{(i)}.$$

This contradicts to the assumption that $p_{(i+1)} \geq \alpha p_{(i)}$ for $1|online, r_j, B = \infty, nondecreasing|C_{max}$.

(2) $0 < \alpha < 1$.
By contradiction. Suppose both batches $i+1$ and $i+2$ are delayed batches. We know that each job in batch $i+2$ arrives later than the starting time of batch $i+1$. Therefore, considering job $J_{(i+2)}$, we have $r_{(i+2)} > \varphi(r_{(i)} + p_{(i)})$. Since batch $i+2$ is a delayed batch, we obtain

$$\varphi(r_{(i)} + p_{(i)}) + p_{(i+1)} > \varphi r_{(i+2)} + (\varphi - 1)p_{(i+2)} > \varphi^2(r_{(i)} + p_{(i)}) + (\varphi - 1)p_{(i+2)}.$$

It follows that

$$p_{(i+1)} > (\varphi^2 - \varphi)(r_{(i)} + p_{(i)}) + (\varphi - 1)p_{(i+2)} \geq (\varphi^2 - \varphi)p_{(i)} = \varphi(\varphi - 1)p_{(i)}. \quad (2)$$

Considering $\varphi = \frac{\sqrt{4\alpha+1}+1}{2}$ and $\alpha < 1$, we have $\varphi(\varphi - 1) = \alpha$, which implies

$$p_{(i+1)} > \alpha p_{(i)}.$$

This contradicts to the assumption that $p_{(i+1)} \leq \alpha p_{(i)}$ for $1|online, r_j, B = \infty, nonincreasing|C_{max}$.
The lemma follows.

Corollary 1. *In the schedule obtained by algorithm H_α^∞, there do not exist two successive delayed batches.*

Proof. By the algorithm, the first batch must be a regular batch. Then we have the desired result.

Let C_H and C^* denote the value obtained by algorithm H_α^∞ and the optimal objective value (for an instance), respectively.

Lemma 4. *If the last batch is a regular batch, i.e., $s_{(m)} = \varphi r_{(m)} + (\varphi - 1)p_{(m)}$, then $C_H/C^* \leq \varphi$.*

Proof. Since $C_H = s_{(m)} + p_{(m)} = \varphi(r_{(m)} + p_{(m)})$ and $C^* \geq r_{(m)} + p_{(m)}$, the lemma follows.

Theorem 1. *Algorithm H_α^∞ is optimal for both problems: $1|online, r_j, B = \infty, nondecreasing|C_{max}$ and $1|online, r_j, B = \infty, nonincreasing|C_{max}$. ($\alpha \geq 1$ in the first problem and $0 < \alpha < 1$ in the second problem.)*

Proof. By contradiction. By Lemma 4, we only need to consider the case where the last batch m is a delayed batch. By algorithm, we know that the first batches

must be a regular batch. Thus, in this case, there must be some batch processed before batch m. By Corollary 1, there do not exist two successive delayed batches in the schedule obtained by algorithm H_α^∞. Therefore, batch $m-1$ must be a regular batch. It follows that

$$s_{(m-1)} = \varphi r_{(m-1)} + (\varphi - 1)p_{(m-1)} \geq (\varphi - 1)p_{(m-1)}. \tag{3}$$

We know

$$C_H = s_{(m-1)} + p_{(m-1)} + p_{(m)}. \tag{4}$$

Note that

$$r_{(m)} > s_{(m-1)}. \tag{5}$$

In the following, we respectively discuss two cases: $\alpha \geq 1$ and $0 < \alpha < 1$.

(1) $\alpha \geq 1$.
We further discuss two cases depending on whether $J_{(m)}$ and $J_{(m-1)}$ are in the same batch in an optimal schedule.

Case 1. In an optimal schedule, $J_{(m)}$ and $J_{(m-1)}$ are in the same batch.

By the assumption that $p_{j+1} \geq \alpha p_j$, we have $C^* \geq r_{(m)} + p_{(m)}$ and $p_{(m)} \geq \alpha p_{(m-1)}$. Note that $\varphi = \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$.

Considering equation (4) and inequalities (3) and (5), it follows

$$\frac{C_H}{C^*} \leq \frac{s_{(m-1)} + p_{(m-1)} + p_{(m)}}{r_{(m)} + p_{(m)}} < 1 + \frac{p_{(m-1)}}{s_{(m-1)} + p_{(m)}} \leq 1 + \frac{1}{\varphi - 1 + \alpha} = \varphi.$$

Case 2. In an optimal schedule, $J_{(m)}$ and $J_{(m-1)}$ are in different batches.

It follows that $C^* \geq r_{(m-1)} + p_{(m-1)} + p_{(m)}$. Considering equations (4) and (3), we have

$$\frac{C_H}{C^*} \leq \frac{s_{(m-1)} + p_{(m-1)} + p_{(m)}}{r_{(m-1)} + p_{(m-1)} + p_{(m)}} = \frac{\varphi(r_{(m-1)} + p_{(m-1)}) + p_{(m)}}{r_{(m-1)} + p_{(m-1)} + p_{(m)}} < \varphi.$$

(2) $0 < \alpha < 1$.
We discuss two cases depending on whether $J_{(m)}$ and $J_{(m-1)}$ are in the same batch in an optimal schedule.

Case 1. In an optimal schedule, $J_{(m)}$ and $J_{(m-1)}$ are in the same batch.

By the assumption that $p_{j+1} \leq \alpha p_j$, we have $C^* \geq r_{(m)} + p_{(m-1)}$ and $p_{(m)} \leq \alpha p_{(m-1)}$. Note that $\varphi = \frac{\sqrt{4\alpha+1}+1}{2}$.

Considering equation (4) and inequalities (3) and (5), it follows

$$\frac{C_H}{C^*} \leq \frac{s_{(m-1)} + p_{(m-1)} + p_{(m)}}{r_{(m)} + p_{(m-1)}} < 1 + \frac{p_{(m)}}{s_{(m-1)} + p_{(m-1)}} \leq 1 + \frac{\alpha}{\varphi} = \varphi.$$

Case 2. In an optimal schedule, $J_{(m)}$ and $J_{(m-1)}$ are in different batches.

It follows that $C^* \geq r_{(m-1)} + p_{(m-1)} + p_{(m)}$. Considering equations (4) and (3), we have

$$\frac{C_H}{C^*} \leq \frac{s_{(m-1)} + p_{(m-1)} + p_{(m)}}{r_{(m-1)} + p_{(m-1)} + p_{(m)}} = \frac{\varphi(r_{(m-1)} + p_{(m-1)}) + p_{(m)}}{r_{(m-1)} + p_{(m-1)} + p_{(m)}} < \varphi.$$

The theorem follows.

Acknowledgements

The authors would like to thank Zhiyi Tan and Yiwei Jiang for communication. This work is partially supported by NSF of China under Grants 70525004, 60736027 and 70702030.

References

1. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
2. Deng, X., Poon, C.K., Zhang, Y.: Approximation Algorithms in Batch Processing. Journal of Combinatorial Optimization 7, 247–257 (2003)
3. Fu, R., Tian, J., Yuan, J., He, C.: On-Line Scheduling on a Batch Machine to Minimize Makespan with Limited Restarts. Operational Research Letters 36, 255–258 (2008)
4. Fu, R., Tian, J., Yuan, J., Lin, Y.: On-Line Scheduling in a Parallel Batch Processing System to Minimize Makespan Using Restarts. Theoretical Computer Science 374, 196–202 (2007)
5. Pruhs, K., Sgall, J., Torng, E.: Online Scheduling. In: Leung, J.Y.-T. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis (2004)
6. Lee, C.Y., Uzsoy, R.: Minimizing Makespan on a Single Batch Processing Machine with Dynamic Job Arrivals. International Journal of Production Research 37, 219–236 (1999)
7. Lee, C.Y., Uzsoy, R., Martin-Vega, L.A.: Efficient Algorithms for Scheduling Semiconductor Burn-in Operations. Operations Research 40, 764–775 (1992)
8. Nong, Q., Yuan, J., Fu, R., Lin, L., Tian, J.: The Single-Machine Parallel-Batching On-Line Scheduling Problem with Family Jobs to Minimize Makespan. International Journal of Production Economics 111, 435–440 (2008)
9. Poon, C.K., Yu, W.: Online Scheduling Algorithms for a Batch Machine with Finite Capacity. Journal of Combinatorial Optimization 9, 167–186 (2005)
10. Zhang, G., Cai, X., Wong, C.: On-Line Algorithms for Minimizing Makespan on Batch Processing Machines. Naval Research Logistics 48, 241–258 (2001)

A Note on Online Scheduling for Jobs with Arbitrary Release Times*

Jihuan Ding[1,2] and Guochuan Zhang[2]

[1] College of Operations Research and Management Science,
Qufu Normal University, Rizhao 276826, China
dingjihuan@hotmail.com
[2] College of Computer Science, Zhejiang University
Hangzhou 310027, China
zgc@zju.edu.cn

Abstract. We investigate the problem of on-line scheduling for jobs with arbitrary release times on m identical parallel machines. The goal is to minimize the makespan. We derive a best possible online algorithm with competitive ratio of 2 for $m = 2$. For a special case that all the jobs have unit processing times, we prove that Algorithm LS has a tight bound of 3/2 for general m machines.

Keywords: On-line scheduling, Makespan, Competitive ratio, Identical parallel machines.

1 Introduction

Online scheduling has received great attention in decades. The most basic model is the classical on-line scheduling problem on m identical parallel machines which was proposed by Graham [1]. In the classical on-line scheduling problem, jobs arrive one by one (or over a job list), we do not know any information about the job list in advance, whenever a job arrives, it must be scheduled immediately on one of the machines without knowledge of any future jobs. Only after the current job is scheduled can the next job appear. The objective is to minimize the makespan. It is well known that the first online algorithm in scheduling theory, namely List Scheduling (LS) algorithm, for the problem was proposed by Graham [1].

Note that in the above model all jobs in the job list have release times zero. A more general version of the classical on-line scheduling problem on m identical parallel machines was given by Li and Huang [2]. In the general version, all jobs appear in form of orders at time zero. When a job appears, the scheduler is informed of the release time and processing time of the job. The problem can be formally defined as follows. A list of jobs is to be scheduled on m identical parallel machines, $M_1, M_2, ..., M_m$. We assume that all jobs appear on-line in a job list. Whenever a job J_j with release time r_j and processing time p_j appears, the

* This work has been supported by NSFC (60573020).

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 354–362, 2009.

scheduler has to assign a machine and a processing slot for J_j irrevocably without knowledge of any future jobs. The goal is to minimize the makespan. In this general on-line situation, the jobs' release times are assumed arbitrary, whereas in the existing literature the jobs' release times are normally non-decreasing. That is to say, although a job may appear first in the job sequence, but its release time may be greater than the release time of the job which appears later in the job sequence. It is obvious that the above model is just the model of classical on-line scheduling problem on m identical parallel machines if all the jobs have release time zero. The problem is called on-line scheduling problem for jobs with arbitrary release times [3].

The quality of an online algorithm A is usually measured by its competitive ratio

$$R(m, A) = sup_L \frac{C^A_{\max}(L)}{C^*_{\max}(L)}$$

where $C^A_{\max}(L)$ and $C^*_{\max}(L)$ denote the makespans of the schedule produced by algorithm A and an optimal off-line algorithm, respectively.

Li and Huang considered the on-line scheduling problem for jobs with arbitrary release times first. For the general model of the problem where job processing times are arbitrary, Li and Huang [2] gave a lower bound of 2, and then showed an algorithm LS with tight bound of $3-1/m$. A modified algorithm MLS is also proposed which is better than LS for any $m \geq 2$, and the competitive ratio of MLS is bounded by 2.9392 for any $m \geq 2$. For the special model where the job length is in $[p, rp](r \geq 1)$, the performance of algorithm LS is analyzed in [3]. They first gave an upper bound of

$$R(m, LS) = \begin{cases} 3 - \frac{1}{m} - \frac{1}{r}, & r \geq \frac{m}{m-1}; \\ 1 + \frac{2r}{1+2r} + \frac{(m-1)r}{m(1+2r)}, & 1 \leq r < \frac{m}{m-1}, \end{cases}$$

for general m and showed that the tight bound for $m = 1$ is $1 + \frac{r}{1+r}$. When $m = 2$, they presented a tight bound of the competitive ratio $1 + \frac{5r+4}{2(r+2)}$ for $r \geq 4$; For $r < 4$, they gave a lower bound and showed that 2 provides an upper bound for the competitive ratio.

Our results. In this paper, we consider the on-line scheduling problem for jobs with arbitrary release times. For the general model where job processing times are arbitrary, we derive a best possible online algorithm with competitive ratio of 2 for $m = 2$. Then we show that the idea of the algorithm for $m = 2$ cannot be extended to the case with $m \geq 3$ machines by a counterexample. Finally, for a special model that all the jobs have unit processing times, we prove that algorithm LS has a tight bound of 3/2 for general m machines.

2 A Best Possible Online Algorithm for $m = 2$

Before we give the best possible online algorithm for $m = 2$, we first describe the following definition which is defined in Li and Huang [2].

Definition 1. *Suppose that J_j is the current job with release time r_j and processing time p_j. We say that machine M_i has an idle time interval for job J_j , if there exists a time interval $[T_1, T_2]$ satisfying the following conditions:*

1. Machine M_i is currently idle in interval $[T_1, T_2]$ and a job has been assigned on M_i to start processing at T_2.

2. $T_2 - \max\{T_1, r_j\} \geq p_j$.

For the job set $\mathbf{J(i)} = \{J_1, J_2, \ldots, J_i\}$ that has been scheduled by the algorithm, let $M_1^i = \max_{1 \leq j \leq i}(r_j + p_j)$, $M_2^i = \frac{1}{2}\sum_{j=1}^{i} p_j$ and $M^i = \max\{M_1^i, M_2^i\}$. It is obvious that M^i is a lower bound of the optimal makespan for job set $\mathbf{J(i)} = \{J_1, J_2, \ldots, J_i\}$.

Algorithm:

---

1. Let L_1^i and L_2^i be the completion time of the last job on machine M_1 and M_2 immediately after J_i has been scheduled by the algorithm, respectively. Let J_{i+1} be a new job given to the algorithm.
2. If there exist some machines which have idle intervals for job J_{i+1}, select a machine M_i which has an idle interval $[T_1, T_2]$ for job J_{i+1} with minimal T_1. Then we start job J_{i+1} on machine M_i at time $\max\{T_1, r_{i+1}\}$ in the idle interval. Otherwise, go to step 3.
3. If $r_{i+1} > \max\{L_1^i, L_2^i\}$, then we assign job J_{i+1} to machine M_1 to start at time r_{i+1}. Otherwise, go to step 4.
4. If $r_{i+1} \leq \max\{L_1^i, L_2^i\}$ and $L_1^i > L_2^i$ and $L_1^i + p_{i+1} \leq 2M^{i+1}$, then we assign job J_{i+1} to machine M_1 to start at time L_1^i. Otherwise, go to step 5.
5. Start job J_{i+1} on machine M_2 at time $\max\{L_2^i, r_{i+1}\}$.

When it is our turn to schedule the next job J_{i+1}, if $r_{i+1} > \max\{L_1^i, L_2^i\}$, then we always assign job J_{i+1} to machine M_1 to start at time r_{i+1}. For the case that $r_{i+1} \leq \max\{L_1^i, L_2^i\}$, unless there is idle interval for job J_{i+1}, the main idea of the algorithm is that we always assign job J_{i+1} to machine with larger load if the completion time of job J_{i+1} is no more than $2M^{i+1}$. Otherwise we will assign job J_{i+1} to another machine to start as soon as possible.

Theorem 1. *The competitive ratio of the algorithm is 2.*

Proof. Let $\mathbf{J(n)} = \{J_1, J_2, \ldots, J_n\}$ be an arbitrary job list. The optimum makespan for job set $\mathbf{J(i)} = \{J_1, J_2, \ldots, J_i\}$ is denoted by OPT^i. For each i, $1 \leq i \leq n$, we will prove that $\max\{L_1^i, L_2^i\} \leq 2OPT^i$ holds immediately after job J_i has been scheduled by the algorithm. If $i = 1$, then it is obvious that the conclusion holds. Now we assume $\max\{L_1^i, L_2^i\} \leq 2OPT^i$ holds immediately after job J_i has been scheduled by the algorithm. Next we will show $\max\{L_1^{i+1}, L_2^{i+1}\} \leq 2OPT^{i+1}$ holds immediately after job J_{i+1} has been scheduled by the algorithm. It is obvious that $OPT^{i+1} \geq M^{i+1}$. So we have $\max\{L_1^{i+1}, L_2^{i+1}\} \leq 2OPT^{i+1}$ if job J_{i+1} is scheduled by Step 2 or Step 3 or

Step 4 of the algorithm. In the following we will show $\max\{L_1^{i+1}, L_2^{i+1}\} \leq 2OPT^{i+1}$ if job J_{i+1} is scheduled by Step 5 of the algorithm.

Case 1. $L_1^i > L_2^i$ and $L_1^i + p_{i+1} > 2M^{i+1}$.

Case 1.1. If $r_{i+1} \geq L_2^i$, then the algorithm will assign job J_{i+1} to machine M_2 to start at time r_{i+1}. By the inductive assumption and the fact that $OPT^{i+1} \geq M^{i+1}$, we have $\max\{L_1^{i+1}, L_2^{i+1}\} \leq 2OPT^{i+1}$.

Case 1.2. If $r_{i+1} < L_2^i$, then the algorithm will assign job J_{i+1} to machine M_2 to start at time L_2^i. Let T_2 be the least time point from which machine M_2 is busy to $L_2^i + p_{i+1}$, namely L_2^{i+1}. Let $l_2 = L_2^i - T_2$. We can describe the completion time of job J_{i+1} as $T_2 + l_2 + p_{i+1}$. If $T_2 = 0$, then by the fact that $OPT^{i+1} \geq \frac{1}{2}(l_2 + p_{i+1}) = \frac{1}{2}L_2^{i+1}$ and the inductive assumption we have $\max\{L_1^{i+1}, L_2^{i+1}\} \leq 2OPT^{i+1}$. If $T_2 > 0$, then let $J_k (k < i+1)$ be the first job scheduled on machine M_2 in time interval $[T_2, L_2^i]$. According to the algorithm, just before the algorithm will schedule job J_k, we must have $L_1^{k-1} > L_2^{k-1}$ and $L_1^{k-1} + p_k > 2M^k$. Otherwise, job J_k will be scheduled on machine M_1. So the start time of job J_k is r_k.

Let T_1 be the least time point from which machine M_1 is busy to L_1^{k-1}. Let $l_1 = L_1^{k-1} - T_1$, then we have $L_1^{k-1} + p_k = T_1 + l_1 + p_k > 2(r_k + p_k)$. Furthermore we have $M^k \geq T_1$, so $T_1 + l_1 + p_k > 2M^k \geq 2T_1$. Combining the above inequalities we can get $l_1 > r_k$. Therefore we have

$$L_2^{i+1} = T_2 + l_2 + p_{i+1} \leq r_k + l_2 + p_{i+1} \leq l_1 + l_2 + p_{i+1} \leq 2OPT^{i+1}.$$

So $\max\{L_1^{i+1}, L_2^{i+1}\} \leq 2OPT^{i+1}$.

Case 2. $L_1^i \leq L_2^i$. By the algorithm we have $r_{i+1} \leq L_2^i$, so the completion time of J_n is $L_2^i + p_{i+1}$. Similar to the Case 1.2, We can find the same T_2 and J_k on machine M_2, and T_1 on machine M_1. Completely repeat the above process, we can get

$$L_2^{i+1} = T_2 + l_2 + p_{i+1} \leq r_k + l_2 + p_{i+1} \leq l_1 + l_2 + p_n \leq 2OPT^{i+1}.$$

Hence we have $\max\{L_1^{i+1}, L_2^{i+1}\} \leq 2OPT^{i+1}$. □

Although the above algorithm is very simple, but the idea of it is impossible to extended to more machines case, even for 3 machines. In the following, a counterexample I for 3 machines is given to show the fact. For the sake of simplicity, let $j = (r_j, p_j)$ denote job j which has release time r_j and processing time p_j.

$I = \{(1-\epsilon, \epsilon), (0,1), (1-\epsilon, \epsilon), (1/3-\epsilon, \epsilon), (1/3-\epsilon, \epsilon), (2/3-\epsilon, \epsilon), (2/3-\epsilon, \epsilon), (0, 1/3), (0, 1/3), (0, 1/3), (1-\epsilon, \epsilon), (0,1), (0,1)\}$.

It is easy to verify the optimal makespan for I is $\frac{4}{3}+3\epsilon$. Let H be the algorithm designed completely by the idea of the algorithm for two-machine case. Then the makespan of Algorithm H for I is 3, see Fig. 1.

3 A Special Case for Jobs with Unit Processing Time

In this part, we give a tight bound of Algorithm LS for a special case that all the jobs have unit processing time. In the following we will describe Algorithm

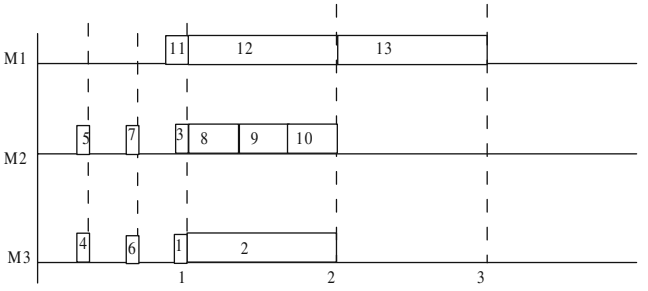


Fig. 1. A counterexample shows that the ratio is not smaller than $\frac{3}{1+1/3} = \frac{9}{4}$. The label on each job denotes not only the order but also the job.

LS which is defined in Li and Huang [2]. Essentially, the algorithm assigns a job to be processed as early as possible when its order arrives.

Algorithm LS

---

1. Assume that L_i is the scheduled completion time of machine $M_i (i = 1, \cdots, m)$. Reorder machines so that $L_1 \leq L_2 \leq \cdots \leq L_m$ and let J_n be a new job given to the algorithm with release time r_n and running time $p_n = 1$. Let $t = \max\{r_n, L_1\}$.
2. If there exist some machines which have idle intervals for job J_n, select a machine M_i which has an idle interval $[T_1, T_2]$ for job J_n with minimal T_1. Then start job J_n on machine M_i at time $\max\{T_1, r_n\}$ in the idle interval. Otherwise, assign job J_n to machine M_1 to start the processing at time t.

In order to give the proof of the result. We first give some explanation of symbols. $[s]$: the integral part of s; (s): the fractional part of s; $\lceil s \rceil$: the smallest integer that is not smaller than s.

The following Lemma gives a estimate of the number of jobs completed in time interval $[0, T]$ on one machine in schedule of Algorithm LS on the assumption that there is no idle time interval with length equal to or greater than 1.

Lemma 1. *Let N denote the number of jobs completed on one machine in time interval $[0, T]$. If the length of all the idle intervals in schedule of Algorithm LS is smaller than 1, then we have (1) $N \geq T/2$ if T is even; (2) $N \geq (T-1)/2$ if T is odd and there is one job with start time smaller than T which is completed after time T. (3) $N \geq (T+1)/2$ if T is odd and there is no job with start time smaller than T which is completed after time T.*

Proof. At most $N + 1$ idle intervals are generated by the process of N jobs in $[0, T]$ on one machine. Then the total length of these idle intervals is smaller than $N + 1$. So we have $(N + 1) + N > T - 1$. From the inequality we can get $N > (T - 2)/2$. Therefore $N \geq T/2$ if T is even;

If T is odd and there is one job with start time smaller than T which is completed after time T. Then we also have $(N+1)+N > T-1$. So we can get $k \geq (T-1)/2$.

If T is odd and there is no job with start time smaller than T which is completed after time T. Then we have $(N+1)+N > T$, namely $N \geq (T+1)/2$. The proof is then finished. □

Theorem 2. *The competitive ratio of LS is* $R_{LS} = 3/2$.

Proof. We assume $L = \{J_1, J_2, \ldots, J_n\}$ is an arbitrary job list. Let $C^{LS}_{\max}(L)$ and $C^{*}_{\max}(L)$ denote the makespans of the schedule produced by Algorithm LS and an optimal schedule, respectively. Without loss of generality, we assume J_n is the last job completed in schedule of Algorithm LS and J_n is the only job with completion time $C^{LS}_{\max}(L)$. Let s_n be the start time of job J_n in schedule of Algorithm LS. If $s_n = r_n$, then it is obvious that $R_{LS} \leq 3/2$. If $s_n > r_n$, then job J_n must be assigned on machine M_1 to start at time $s_n = L_1$ by the algorithm. In schedule of Algorithm LS, let s be the least time point from which all the m machines are busy to L_1, and let k be the number of jobs(including job J_n) completed after time s on machine M_1, then it is obvious that at least $(k-1)$ jobs are completed after time s on each of the other $m-1$ machines. So at least $m(k-1)+1$ jobs in total are completed after time s in schedule of Algorithm LS. We have $C^{LS}_{\max}(L) \leq s+k$ and $C^{*}_{\max}(L) \geq s+1$, because at least one job has release time s by the algorithm.

We first consider the case that all the idle intervals in schedule of Algorithm LS have length smaller than 1.

Case 1. $[s]$ is even. By Lemma 1, it is easy to know the optimal schedule completes at most $m \cdot [s]/2$ more jobs before time $[s]$ than the schedule of Algorithm LS does.

Case 1.1. If $2k \leq s+3$, then we have:

$$\frac{C^{LS}_{\max}(L)}{C^{*}_{\max}(L)} \leq \frac{s+k}{s+1} \leq \frac{2s+s+3}{2(s+1)} = \frac{3}{2}.$$

Case 1.2. If $2k \geq s+3 = [s]+2+1+(s)$, namely $2k \geq s+4$. Then at least $m(k-1)+1-m\cdot[s]/2$ jobs are processed after time $[s]$ in optimal schedule. So we have

$$\begin{aligned} C^{*}_{\max}(L) &\geq [s] + \left\lceil \frac{1}{m} \left[m(k-1)+1-m\cdot\frac{[s]}{2} \right] \right\rceil \\ &= \frac{1}{2}[s] + k. \end{aligned}$$

Therefore:

$$\frac{C^{LS}_{\max}(L)}{C^{*}_{\max}(L)} \leq \frac{s+k}{\frac{1}{2}[s]+k} \leq \frac{2s+[s]+4}{2([s]+2)}$$

$$= \frac{3([s]+2)+2(s)-2}{2([s]+2)}$$
$$\leq \frac{3}{2}.$$

Case 2. $[s]$ is odd.

Case 2.1. If $2k \leq s+3$, then:

$$\frac{C_{\max}^{LS}(L)}{C_{\max}^{*}(L)} \leq \frac{s+k}{s+1} \leq \frac{2s+s+3}{2(s+1)} = \frac{3}{2}.$$

Case 2.2. If $2k > s+3 = [s]+2+1+(s)$, namely, $2k \geq s+5$. By Lemma 1, one machine in optimal schedule completes at most $\frac{[s]+1}{2}$ more jobs before $[s]$ than it does in schedule of Algorithm LS, and at least one machine in optimal schedule(the machine is idle immediately before s) completes at most $\frac{[s]-1}{2}$ more jobs before $[s]$ than it does in schedule of Algorithm LS. So at least $[m(k-1)+1]-\left[(m-1)\frac{[s]+1}{2}+\frac{[s]-1}{2}\right]$ jobs must be processed after time $[s]$ in optimal schedule. Hence we can derive

$$C_{\max}^{*}(L) \geq [s] + \left\lceil \frac{[m(k-1)+1]-\left[(m-1)\frac{[s]+1}{2}+\frac{[s]-1}{2}\right]}{m} \right\rceil$$
$$= [s] + (k-1) - \frac{[s]+1}{2} + 1$$
$$= \frac{1}{2}[s] + k - \frac{1}{2}.$$

Case 2.2.1. If $(s) \leq \frac{1}{2}$, then:

$$\frac{C_{\max}^{LS}(L)}{C_{\max}^{*}(L)} \leq \frac{s+k}{\frac{1}{2}[s]+k-\frac{1}{2}}$$
$$\leq \frac{2s+[s]+5}{[s]+[s]+5-1}$$
$$= \frac{3([s]+2)+2(s-[s])-1}{2([s]+2)}$$
$$\leq \frac{3}{2}.$$

Case 2.2.2. $(s) > \frac{1}{2}$. In this case we need to estimate the number of jobs completed after time s in schedule of Algorithm LS more carefully.

Case 2.2.2.a. The time point s appears exactly on machine M_1, namely M_1 is idle immediately before time s, and p jobs with start time smaller than $[s]$ are completed after time s in schedule of Algorithm LS. For the p machines that process such p jobs, k jobs are completed after time s on each of them in schedule of Algorithm LS. By Lemma 1, each of the p machines in optimal schedule

completes at most $\frac{[s]+1}{2}$ more jobs before time $[s]$ than it does in schedule of Algorithm LS and each of the the other $m-p$ machines in optimal schedule completes at most $([s]-1)/2$ more jobs before time $[s]$ than it does in schedule of Algorithm LS. So we have:

$$\begin{aligned} C^*_{\max}(L) &\geq [s] + \left\lceil \frac{1}{m} \left\{ [m(k-1)+p+1] - \left[p \cdot \frac{[s]+1}{2} + (m-p)\frac{[s]-1}{2} \right] \right\} \right\rceil \\ &= [s] + (k-1) - \frac{[s]-1}{2} + 1 \\ &= \frac{1}{2}[s] + k + \frac{1}{2}. \end{aligned}$$

Therefore:

$$\begin{aligned} \frac{C^{LS}_{\max}(L)}{C^*_{\max}(L)} &\leq \frac{s+k}{\frac{1}{2}[s]+k+\frac{1}{2}} \leq \frac{2s+[s]+5}{2([s]+3)} \\ &= \frac{3([s]+3)+2(s-[s])-4}{2([s]+3)} \\ &\leq \frac{3}{2}. \end{aligned}$$

Case 2.2.2.b. The time point s does not appear on machine M_1 and the start time of the first completed job after time s on machine M_1 is not greater than $[s]$. Then we have

$$C^{LS}_{\max}(L) \leq k + [s].$$

Hence

$$\begin{aligned} \frac{C^{LS}_{\max}(L)}{C^*_{\max}(L)} &\leq \frac{[s]+k}{\frac{1}{2}[s]+k-\frac{1}{2}} \\ &\leq \frac{2[s]+[s]+5}{[s]+[s]+5-1} \\ &= \frac{3([s]+2)-1}{2([s]+2)} \\ &\leq \frac{3}{2}. \end{aligned}$$

Case 2.2.2.c. The time point s does not appear on machine M_1, and p jobs with start time smaller than $[s]$ are completed after time s in schedule of Algorithm LS, and the start time of the first completed job after time s on machine M_1 is greater than $[s]$. Similar to the analysis in Case 2.2.2.a, we have

$$\begin{aligned} C^*_{\max}(L) &\geq [s] + \left\lceil \frac{1}{m} \left\{ [m(k-1)+p+1] - \left[p \cdot \frac{[s]+1}{2} + (m-p)\frac{[s]-1}{2} \right] \right\} \right\rceil \\ &= [s] + (k-1) - \frac{[s]-1}{2} + 1 \\ &= \frac{1}{2}[s] + k + \frac{1}{2}. \end{aligned}$$

So

$$\begin{aligned}\frac{C_{\max}^{LS}(L)}{C_{\max}^{*}(L)} &\leq \frac{s+k}{\frac{1}{2}[s]+k+\frac{1}{2}} \leq \frac{2s+[s]+5}{2([s]+3)} \\ &= \frac{3([s]+3)+2(s-[s])-4}{2([s]+3)} \\ &\leq \frac{3}{2}.\end{aligned}$$

Secondly, if there exist idle time intervals with length not smaller than 1 in schedule of Algorithm LS. We select a such idle time interval with latest over time, say b, and choose a time point a such that $b-a=1$. By the Algorithm LS, all the jobs with start time greater than a must have release time greater than a also. So similar to the analysis in the first part of the proof, we can get $R_{LS} \leq 3/2$.

Finally, the following instance shows that the bound of Algorithm LS is tight. The instance consists $2m$ jobs in total, the first m jobs have release time $1-\epsilon$ and the last m jobs have release time zero. It is easy to know the makespan of Algorithm LS for the instance is $3-\epsilon$, but the optimal makespan is 2. So $R_{LS} \geq (3-\epsilon)/2$, let ϵ tend to zero, we get the bound is tight. □

4 Final Remarks

We consider the problem of scheduling for jobs with arbitrary release times on m identical parallel machines. For $m=2$, we gave a best possible online algorithm. It must be a challenging work to design best possible online algorithm for $m \geq 3$. For a special model where all the jobs have processing time 1, we derived that Algorithm LS has a tight bound of 3/2. Furthermore, some other special models of the problem are also worth to considering in future.

References

1. Graham, R.L.: Bounds on Multiprocessing Timing Anomalies. SIAM Journal on Applied Mathematics 17, 416–429 (1969)
2. Li, R., Huang, H.C.: On-line Scheduling for Jobs with Arbitrary Release Times. Computing 73, 79–97 (2004)
3. Li, R., Huang, H.C.: List Scheduling for Jobs with Arbitrary Release Times and Similar Lengths. Journal of Scheduling 10, 365–373 (2007)

Size-Constrained Tree Partitioning: A Story on Approximation Algorithm Design for the Multicast k-Tree Routing Problem

Zhipeng Cai*, Randy Goebel**, and Guohui Lin***

Department of Computing Science, University of Alberta
Edmonton, Alberta T6G 2E8, Canada
{zhipeng,goebel,ghlin}@cs.ualberta.ca

Abstract. In the Multicast k-Tree Routing Problem, a data copy is sent from the source node to at most k destination nodes in every transmission. The goal is to minimize the total cost of sending data to all destination nodes, which is measured as the sum of the costs of all routing trees. This problem was formulated out of optical networking and has applications in general multicasting. Several approximation algorithms, with increasing performance, have been proposed in the last several years; The most recent ones are heavily relied on a *tree partitioning* technique. In this paper, we present a further improved approximation algorithm along the line. The algorithm has a worst case performance ratio of $\frac{5}{4}\rho + \frac{3}{2}$, where ρ denotes the best approximation ratio for the Steiner Minimum Tree problem. The proofs of the technical routing lemmas also provide some insights on why such a performance ratio could be the best possible that one can get using this tree partitioning technique.

Keywords: Capacitated Multicast Tree Routing, Approximation Algorithm, Tree Partitioning.

1 Introduction

Multicast is a point-to-multipoint communication that a source node sends data to multiple destinations [2,19,13,11,18]. In computer and communication networks supporting multimedia applications, such as news feed and video distribution, multicast is an important service. Implementing multicast on local area networks (LANs) is easy because nodes connected to a LAN usually communicate over a broadcast network. In contrast, implementing multicast on wide area networks (WANs) is yet quite challenging [20,9] because nodes connected to a WAN typically communicate via a switched/routed network. Basically, to perform multicast in WANs, the source node and all the destination nodes must be interconnected. So, finding a multicast routing in a WAN is equivalent to

* Supported by AICML and NSERC.
** Supported by AICML, iCORE, and NSERC.
*** To whom correspondence should be addressed. Supported by NSERC.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 363–374, 2009.

finding a multicast tree T in the network such that T spans the source node and all the destination nodes. The objective of the routing is to minimize the *cost* of T, which is defined to be the total weight of edges in T.

In certain networks such as wavelength-division multiplexing (WDM) optical networks with limited light-splitting capabilities, during each transmission, only a limited number of destination nodes can be assigned to receive the data copies sent from the source node. A routing model for such networks, called the *multi-tree model* [14,9,10,12], has been introduced in the literature. Under this model, we are interested in the problem of finding a collection of routing trees such that each tree spans the source node and a limited number of destination nodes that are assigned to receive data copies, and every destination node must be designated to receive a data copy in one of the routing trees. We call this problem the *capacitated multicast routing problem*. In particular, when the number of destination nodes in each routing tree is limited to a pre-specified number k, we call it the *multicast k-tree routing* (kMTR) problem. Correspondingly, a feasible routing solution is called a *k-tree routing*. Compared with the traditional multicast routing model without the capacity constraint — the *Steiner Minimum Tree* (SMT) problem, which allows any number of receivers in the routing tree, this simpler model makes multicast easier and more efficient to be implemented, at the expense of increasing the total routing cost.

We next formally define the kMTR problem. For a graph G, we denote its node set by $V(G)$. The underlying communication network is modeled as a triple (G, s, D), where G is a simple, undirected, and edge-weighted complete graph, $s \in V(G)$ is the *source* node, and $D = \{d_1, d_2, \ldots, d_n\} \subseteq V(G) - \{s\}$ is the set of *destination* nodes. The weight of each edge e in G, denoted by $w(e)$, is nonnegative and represents the routing cost of e. The *weight* (or *cost*, used interchangeably) of a subgraph T of G, denoted by $w(T)$, is the total weight of edges in T. Let k be a given positive integer. The kMTR problem asks for a *minimum-weight k-tree routing*, that is, a partition of D into disjoint sets $D_1, D_2, \ldots, D_\ell$, such that each D_i contains no more than k destination nodes, and a Steiner tree T_i spanning the source node s and the destination nodes in D_i for $i = 1, 2, \ldots, \ell$, such that $\sum_{i=1}^{\ell} w(T_i)$ is minimized. It is worth pointing out that the union of all these routing trees does not necessarily remain as a tree, since some destination nodes in D_i could appear as *Steiner* nodes for purely routing purpose in T_j for $j \neq i$ (i.e., they are not counted as destination nodes towards D_j). In the following, we assume without loss of generality that the edge weight function (the shortest path metric) satisfies triangle inequality.

When $k \geq |D|$, kMTR reduces to the well-known SMT problem. The SMT problem is NP-hard, and its current best approximation ratio is $\rho \approx 1.55$ [8,17]. (We reserve ρ to denote this best approximation ratio throughout the paper.) On the other hand, when $k = 1$ or 2, kMTR can be solved efficiently [9,10]. The algorithmically most interesting case is $3 \leq k < |D|$, where kMTR differs from the SMT problem yet remains NP-hard [3,15].

Let $\{T_1^*, T_2^*, \ldots, T_m^*\}$ be the set of trees in an optimal k-tree routing. Let $w(T_j^*)$ denote the weight of tree T_j^*, the sum of the weights of edges in T_j^*.

Let $R^* = \sum_{j=1}^{m} w(T_j^*)$ be the weight of this optimal k-tree routing. Since every destination node d_i in tree T_j^* satisfies $w(s, d_i) \leq w(T_j^*)$, due to non-negative edge weights, we have

$$\sum_{i=1}^{n} w(s, d_i) \leq k \times R^*. \tag{1}$$

Let T be a tree in (G, s, D) containing a subset D_T of destination nodes. For ease of presentation, we define the *size* of tree T to be $|D_T|$, though T might contain other non-destination nodes. Tree T can be used in a feasible k-tree routing to route as many as k destination nodes in D_T. For doing so, the incurred routing cost will be the weight of tree T, $w(T)$, plus a *connection cost*, $c(T)$, which is measured as the minimum weight of edges between the source node s and all nodes in $V(T)$. It follows that if source node s is in $V(T)$, then $c(T) = 0$; In the other cases, we can always have

$$c(T) \leq \min_{d \in D_T} w(s, d) \leq \frac{1}{|D_T|} \sum_{d \in D_T} w(s, d). \tag{2}$$

As noted by Jothi and Raghavachari [12], an algorithm presented by Altinkemer and Gavish [1] about twenty years ago for a slightly different problem serves as a $(2\rho + 1)$-approximation algorithm for kMTR. Hu and his colleagues [9,10] are probably the first to study the kMTR problem, and they presented an approximation algorithm starting with a Hamiltonian cycle on $s \cup D$ obtained by the $\frac{3}{2}$-approximation algorithm for the metric *Traveling Salesman Problem* (TSP) [7], partitioning it into segments each containing exactly k destination nodes (except one segment), and then connecting these segments to source node s separately via the minimum weight edge to form a k-tree routing. Note that the weight of an optimal TSP tour on $s \cup D$ is upper bounded by $2R^*$. By Eqs. (1) and (2), the connection cost for this k-tree routing is bounded from above by $\frac{1}{k} \sum_{i=1}^{n} w(s, d_i) \leq R^*$. Therefore, the weight of this k-tree routing is at most $(\frac{3}{2} \times 2 + 1)R^* = 4R^*$, and their algorithm is a 4-approximation, an improvement over the $(2\rho + 1)$-approximation [1,12].

Lin [15] proposed to start with a good Steiner tree on $s \cup D$, obtained by any current best approximation algorithms for the SMT problem, partition it into small trees of size at most k without duplicating any edges (and thus does not increase the weight of the Steiner tree), and then connect each such obtained tree to the source via the minimum weight edge. He demonstrated that in a top-down fashion the Steiner tree can be partitioned into a collection of feasible routing trees, such that each tree has size in $(\frac{1}{6}k, k]$ and its connection cost is less than or equal to the minimum edge weight of a distinct set of at least $\frac{5}{12}k$ source-to-destination edges. Equivalently speaking, on average, each routing tree has size at least $\frac{5}{12}k$. By Eqs. (1) and (2), the connection cost for this k-tree routing is bounded from above by $\frac{12}{5}k \sum_{i=1}^{n} w(s, d_i) \leq 2.4R^*$. Thus it is an improved $(\rho + 2.4)$-approximation algorithm for kMTR.

Subsequently, two groups of researchers [3,6,12] independently designed $(\rho + 2)$-approximation algorithms for kMTR. Cai and his colleagues [3,6] continued

the study from [9,10,15]; Jothi and Raghavachari [12] were directed from [1] to consider a variant of kMTR in which the destination nodes have varying integral amounts of request and no destination nodes can be used as Steiner points to assist the routing. When the given network is completely connected, Jothi and Raghavachari [12] designed an approximation algorithm for the variant. This approximation algorithm turns out to be a $(\rho + 2)$-approximation algorithm for kMTR. The two $(\rho+2)$-approximation algorithms are surprisingly similar in the design nature that, both algorithms start with a Steiner tree on $s \cup D$, partition it into feasible routing trees without duplicating any edges, and then connect each such obtained tree to the source via the minimum weight edge. The difference between them is that the algorithm by Cai *et al.* partitions the tree in a top-down fashion to guarantee that, equivalently speaking, each tree has size in the range $[\frac{1}{2}k, k]$, while the algorithm by Jothi and Raghavachari cuts iteratively from the Steiner tree a subtree of size in the range $[\frac{1}{2}k, k]$. It follows from Eqs. (1) and (2) that the connection cost of both k-tree routings are at most $2R^*$, implying that the two algorithm are both $(\rho + 2)$-approximations for kMTR.

All the above approximation algorithms [1,9,10,3,6,12] show that the total cost of a k-tree routing consists of two components: the weight of the initial infeasible solution subgraph (such as a Hamiltonian cycle or a Steiner tree) and the connection cost depending on the size range of the achieved routing trees using Eqs. (1) and (2). These two components are seemingly independent but actually closely related to each other. Efforts have been invested in developing better tree partitioning schemes without increasing the total weight of the routing trees too much, compared with the weight of the initial infeasible solution subgraph. For example, Morsy and Nagamochi [16] presented a tree partitioning scheme that can, roughly speaking, guarantee a lower bound of $\frac{2}{3}k$ on the size of the routing trees at a cost of $\frac{1}{3}$ the weight of the starting Steiner tree. This gives a new approximation algorithm with a worst-case performance ratio of $(\frac{4}{3}\rho+\frac{3}{2})$. Unfortunately, this is not an improvement over the $(\rho + 2)$-approximation algorithms unless $\rho < 1.5$ [16]. Cai *et al.* [5] were able to do better. Last year, they presented at COCOA 2008 a slightly different but better tree partitioning scheme to guarantee a lower bound of $\frac{5}{8}k$ on the size of the routing trees at the expense of $\frac{1}{4}$ the weight of the starting Steiner tree. Their algorithm is thus a $(\frac{5}{4}\rho + \frac{8}{5})$-approximation, a genuine improvement over the $(\rho + 2)$-algorithms, given that $\rho \approx 1.55$. Most recently, Cai *et al.* [4] further improved their tree partitioning scheme to guarantee a better lower bound of $\frac{80}{\sqrt{2089}+77}k$ on the size of the routing trees, giving rise to a $(\frac{5}{4}\rho + \frac{\sqrt{2089}+77}{80})$-approximation algorithm for kMTR.

In this paper, we show that at the same expense of $\frac{1}{4}$ the weight of the starting Steiner tree, the lower bound of $\frac{2}{3}k$ on the size of the routing trees can be guaranteed. This results in a $(\frac{5}{4}\rho+\frac{3}{2})$-approximation algorithm for the kMTR problem. On one hand, this algorithm outperforms the previous best by 0.0338; On the other hand, it beats the $(\frac{4}{3}\rho + \frac{3}{2})$-approximation algorithm by Morsy and Nagamochi [16] for any possible value of ρ. The following Table 1 contains a historical record of the approximation algorithms for kMTR, to the best of

Table 1. A historical record of the approximation algorithms for kMTR

Year	References	Performance Ratio
1988	[1]	$2\rho + 1 = 4.10$
2004	[9,10]	4
2004/2005	[3,15]	$\rho + 2.4 = 3.95$
2004/2005	[3,12,6]	$\rho + 2 = 3.55$
2008	[16]	$\frac{4}{3}\rho + \frac{3}{2} = 3.5667$
2008	[5]	$\frac{5}{4}\rho + \frac{8}{5} = 3.5375$
2009	[4]	$\frac{5}{4}\rho + \frac{\sqrt{2089}+77}{80} = 3.4713$
2009	this paper	$\frac{5}{4}\rho + \frac{3}{2} = 3.4375$

our knowledge. Nevertheless, it is worth pointing out that our new design and analysis presented here are nothing but more careful case analysis, yet we doubt that any further improvement is achievable along this tree partitioning line of research, if no new techniques are introduced.

2 A $(\frac{5}{4}\rho + \frac{3}{2})$-Approximation Algorithm for kMTR

Following previous design, we first apply the best known ρ-approximation algorithm for the SMT problem to obtain a Steiner tree T^0 on $\{s\} \cup D$ in the underlying network (G, s, D). As discussed earlier, $w(T^0) \leq \rho R^*$, where R^* denotes the weight of an optimal k-tree routing. Root tree T^0 at source s and denote it as T^0_s. One may use $\frac{n}{k}$ copies of tree T^0_s to form a k-tree routing, which is apparently very expensive. In the sequel, we use T_v to denote the rooted subtree at v in T^0_s, and D_v denotes the associated destination node set of T_v (i.e., $D_v = D \cap T_v$). Also, for a child u of v in T_v, the subtree T_u together with edge (v, u) is called the *branch rooted at v and containing u*. In the algorithm to be presented, we will iteratively cut from T^0_s a rooted subtree T_r of certain size if $|T^0_s| > \frac{4}{3}k$. This cutting process does not duplicate any edge and thus would not increase the tree weight. Nonetheless, (at most) one node might need to be duplicated for connectivity purpose. We then show that using tree T_r, the destination nodes in D_r can be routed at a cost

$$\leq \frac{5}{4}w(T_r) + \frac{3}{2} \times \frac{1}{k} \sum_{d \in D_r} w(s, d). \tag{3}$$

Given the rooted tree T^0_s, for any internal node v, we have the destination node set D_v for the rooted subtree T_v. For ease of presentation, we say v has size $|D_v|$. We assume the non-trivial case that source s has size $> \frac{4}{3}k$. At each iteration, the cutting process examines T^0_s in a bottom-up fashion. First of all, for any u_1, u_2 being two children of v, if their total size is $\leq k$, then the corresponding two branches are merged into one (via a copy of v, say v', and a dummy edge (v, v') of cost 0). If the process locates a node r of size in the range $[\frac{2}{3}k, k]$, it

cuts T_r off the tree and the destination nodes in D_r are routed by Lemma 1 with the routing cost bounded by Eq. (3); Otherwise, define a node as *extreme* if its size is $> k$ but none of its children has size $> k$. It follows that every extreme node v has at least two children (of size $< \frac{2}{3}k$).

If there exists an extreme node r having more than two children, the process cuts off exactly three branches rooted at r, together with a copy of r, to form a three-branch subtree T_r. From the fact that the total size of every pair of two subtrees is greater than k, one concludes that the size of T_r is in the open interval $(\frac{3}{2}k, 2k)$. The destination nodes in D_r are routed by Lemma 3 with the routing cost accordingly bounded by Eq. (3). When every extreme node has only two children, the process searches for a node of size in the range $[\frac{4}{3}k, 2k]$.

If a such node is found, say u, then u will have exactly one descendent, say r, which is extreme. The process cuts T_u off the Steiner tree and re-roots it at r, subsequently denoted as T_r. Two cases are distinguished depending on the size of D_r, the set of destination nodes in T_r. If $|D_r| \geq \frac{3}{2}k$, then these destination nodes are routed again by Lemma 3 with the routing cost accordingly bounded by Eq. (3); If $|D_r| < \frac{3}{2}k$, then the destination nodes are routed by Lemma 4 with the routing cost accordingly bounded by Eq. (3). The remaining case is that every extreme node has exactly two children, and every node has size in the range $(0, \frac{2}{3}k] \cup (k, \frac{4}{3}k) \cup (2k, n]$.

Now, for any node v of size $> 2k$, if none of its children has size $> 2k$, then v is called *super*. Clearly, a super node must have at least two children, each of which has size in the range $(0, \frac{2}{3}k] \cup (k, \frac{4}{3}k)$. Note that if there are two children of size $\leq \frac{2}{3}k$, we can merge the two corresponding branches into one (via a copy of v, say v', which becomes an extreme node, and a dummy edge (v, v') of cost 0). It follows that we may assume without loss of generality that there is at most one child of size $\leq \frac{2}{3}k$. Consequently, there are at least two children of size in the open interval $(k, \frac{4}{3}k)$. The process locates a super node r, cuts off exactly two of its branches of size $> k$, together with a copy of r, to form a two-branch subtree T_r. The number of destination nodes in this subtree T_r, $|D_r|$, is thus in the range $(2k, \frac{8}{3}k)$. Two cases are distinguished depending on the actual size of D_r: If $|D_r| \leq \frac{5}{2}k$, then these destination nodes are routed by Lemma 5 with the routing cost accordingly bounded by Eq. (3); If $|D_r| > \frac{5}{2}k$, then the destination nodes are routed by Lemma 6 with the routing cost accordingly bounded by Eq. (3).

At the end, when no subtree can be cut out of the base Steiner tree, still denoted as T_s^0, we conclude that the size of T_s^0 is in the range $(0, \frac{2}{3}k] \cup (k, \frac{4}{3}k)$. In the former case, this residual tree is taken as a routing tree to route the destination nodes therein, with routing cost $w(T_s^0)$; In the latter case, the residual tree is split into two routing trees to route the destination nodes therein, with the total routing cost $\leq w(T_s^0) + \frac{1}{k}\sum_{d \in D_s} w(s, d)$ [5]. Summing up, the main result is the following:

Theorem 1. *k*MTR ($k \geq 3$) *admits a* $(\frac{5}{4}\rho + \frac{3}{2})$*-approximation algorithm, where* ρ *is the currently best performance ratio for approximating the* Steiner Minimum Tree *problem.*

Proof. Notice that whenever a subtree T_r is cut out of the base Steiner tree T_s^0, we do not increase the weight of the trees, though we might need to duplicate a certain (Steiner or destination) node for connectivity purpose. The total routing cost for T_r, as proven in Lemmas 1 and 3-6, is upper bounded by Eq. (3): $\frac{5}{4}w(T_r) + \frac{3}{2} \times \frac{1}{k}\sum_{d\in D_r} w(s,d)$. The total routing cost for the residual Steiner tree is also upper bounded by Eq. (3). Therefore, the total routing cost for the output k-tree routing is $R \leq \frac{5}{4}w(T_s^0) + \frac{3}{2} \times \frac{1}{k}\sum_{d\in D} w(s,d) \leq \frac{5}{4}w(T_s^0) + \frac{3}{2}R^*$, where the last inequality follows from Eq. (1). Since $w(T_s^0) \leq \rho R^*$, we have $R \leq (\frac{5}{4}\rho + \frac{3}{2})R^*$. □

2.1 Technical Lemmas

Lemma 1. [5] *Given a Steiner tree T_r such that*

- $\frac{2}{3}k \leq |D_r| \leq k$,

the routing cost for T_r is $\leq w(T_r) + \frac{3}{2} \times \frac{1}{k}\sum_{d\in D_r} w(s,d)$.

Lemma 2. [3,6] *Given a Steiner tree T_r such that*

- $k < |D_r| \leq \frac{3}{2}k$.

It is always possible to partition T into two subtrees of size $\leq k$, such that the total routing cost for these subtrees is $\leq w(T_r) + 2 \times \frac{1}{k}\sum_{d\in D_r} w(s,d)$.

Lemma 3. [5] *Given a Steiner tree T_r such that*

- $\frac{3}{2}k < |D_r| \leq 2k$;
- *root node r has exactly three child nodes v_1, v_2, v_3; and*
- $|D_{v_1}| < \frac{2}{3}k$, $|D_{v_2}| < \frac{2}{3}k$, *and* $|D_{v_1}| + |D_{v_2}| > k$.

It is always possible to partition T_r into two or three subtrees of size $\leq k$, such that the total routing cost for these subtrees is $\leq \frac{5}{4}w(T_r) + \frac{3}{2} \times \frac{1}{k}\sum_{d\in D_r} w(s,d)$.

Lemma 4. *Given a Steiner tree T_r such that*

- $\frac{4}{3}k \leq |D_r| < \frac{3}{2}k$;
- *root node r has exactly three child nodes v_1, v_2, v_3; and*
- $|D_{v_1}| < \frac{2}{3}k$, $|D_{v_2}| < \frac{2}{3}k$, *and* $|D_{v_1}| + |D_{v_2}| > k$.

It is always possible to partition T_r into two subtrees of size $\leq k$, such that the total routing cost for these subtrees is $\leq \frac{5}{4}w(T_r) + \frac{3}{2} \times \frac{1}{k}\sum_{d\in D_r} w(s,d)$.

Proof. By the conditions in the lemma, $|D_{v_3}| < \frac{1}{2}k$. Without loss of generality, we assume that $|D_{v_1}| \leq |D_{v_2}|$. Then, $|D_{v_2}| > \frac{1}{2}k$. For each $i \in \{1,2,3\}$, let B_i be the branch rooted at r and containing v_i. We distinguish two cases as follows.

Case 1: [4] $|D_{v_3}| + |D_{v_2}| \leq k$. In this case, $|D_{v_3}| + |D_{v_1}| \leq k$. Among the nodes in D_r, we find the $\frac{2}{3}k$ closest nodes to s, and form them into a set C. Similarly, among the nodes in D_r, we find the $\frac{2}{3}k$ farthest nodes from s, and form them

into a set F. Since $|D_r| \geq \frac{4}{3}k$, $F \cap C = \emptyset$. Moreover, since $|D_{v_i}| < \frac{2}{3}k$ for each $i \in \{1,2,3\}$, there are at least two indices $i \in \{1,2,3\}$ such that $(D_{v_i}) \cap C \neq \emptyset$. If $(D_{v_3}) \cap C = \emptyset$, then we set $X_1 = B_1$ and construct X_2 by initializing it as the union of B_2 and B_3 and then minus r. Otherwise, we find an index $i \in \{1,2\}$ with $(D_{v_i}) \cap C \neq \emptyset$, set $X_1 = B_i$ and construct X_2 by initializing it as the union of B_j and B_3 and then minus r, where j is the other index in $\{1,2\} - \{i\}$. In any case, $|D_r \cap X_1| \leq k$, $|D_r \cap X_2| \leq k$, $(D_r \cap X_1) \cap C \neq \emptyset$, and $(D_r \cap X_2) \cap C \neq \emptyset$. Obviously, one of $D_r \cap X_1$ and $D_r \cap X_2$ contains d' which is the closest destination node to s among the nodes in D_r. We assume that $D_r \cap X_1$ contains d'; the other case is symmetric. Then, $c(X_1) \leq w(s,d') \leq \frac{3}{2} \times \frac{1}{k} \sum_{d \in C} w(s,d)$. Moreover, since $(D_r \cap X_2) \cap C \neq \emptyset$, $c(X_2) \leq w(s,d'')$ where d'' is the farthest destination node from s among the nodes in C. Furthermore, since $C \cap F = \emptyset$, $w(s,d'') \leq w(s,d''')$ where d''' is the closest destination node to s among the nodes in F. Thus, $c(X_2) \leq w(s,d''') \leq \frac{3}{2} \times \frac{1}{k} \sum_{d \in F} w(s,d)$. Therefore, $c(X_1) + c(X_2) \leq \frac{3}{2} \times \frac{1}{k} \sum_{d \in D_r} w(s,d)$. Consequently, the total routing cost of X_1 and X_2 is at most $w(T_r) + \frac{3}{2} \times \frac{1}{k} \sum_{d \in D_r} w(s,d)$, and the lemma is proved.

Case 2: $|D_{v_3}| + |D_{v_2}| > k$. We assume that $|D_{v_2}| = (\frac{1}{2} + p)k$. Note that since $|D_{v_1}| + |D_{v_2}| > k$, we have $|D_{v_1}| > (\frac{1}{2} - p)k$; Likewise, we have $|D_{v_3}| > (\frac{1}{2} - p)k$. These two together imply that $|D_{v_1}| + |D_{v_3}| \in ((1-2p)k, (1-p)k)$. Therefore, in the first routing option, we set $X_1 = B_2$ and construct X_2 by initializing it as the union of B_1 and B_3 and then minus r. It follows that the connection cost $c(X_1) \leq \frac{1}{\frac{1}{2}+p} \times \frac{1}{k} \sum_{d \in D_{v_2}} w(s,d)$, and the connection cost $c(X_2) \leq \frac{1}{1-2p} \times \frac{1}{k} \sum_{d \in D_{v_1} \cup D_{v_3}} w(s,d)$. Since $p \in (0, \frac{1}{6})$, the total routing cost of X_1 and X_2 is $w_1 \leq w(T_r) + \frac{1}{\frac{1}{2}+p} \times \frac{1}{k} \sum_{d \in D_r} w(s,d)$.

Next, among the three branches, we select the one having the minimum weight and duplicate it. Assume without loss of generality that B_1 has the minimum weight. Then, $w(B_1) \leq \frac{1}{3} w(T_r)$. We create two routing trees out of this tree, one is the union of one copy of branch B_1 and branch B_2 and the other the union of the other copy of branch B_1 and branch B_3 and then minus r. The destination nodes routed by the first routing tree include the ones in D_{v_2} and a subset of D_{v_1}, such that their number is exactly $\frac{1}{2}|D_r|$, which is $> (\frac{3}{4} - \frac{1}{2}p)k$; The destination nodes routed by the second routing tree include the ones in D_{v_3} and the remainder of D_{v_1}. It follows that the total routing cost of the second routing option is $w_2 \leq \frac{4}{3} w(T_r) + \frac{1}{\frac{3}{4} - \frac{1}{2}p} \times \frac{1}{k} \sum_{d \in D_r} w(s,d)$.

Note that $\min\{w_1, w_2\} \leq \frac{1}{4} w_1 + \frac{3}{4} w_2 = \frac{5}{4} w(T_r) + \left(\frac{1}{4(\frac{1}{2}+p)} + \frac{3}{4(\frac{3}{4} - \frac{1}{2}p)} \right) \times \sum_{d \in D_r} w(s,d) = \frac{5}{4} w(T_r) + \frac{9+10p}{(2+4p)(3-2p)} \sum_{d \in D_r} w(s,d)$. From $0 < p < \frac{1}{6}$, we conclude that $\min\{w_1, w_2\} < \frac{5}{4} w(T_r) + \frac{3}{2} \times \frac{1}{k} \sum_{d \in D_r} w(s,d)$, since $\frac{9+10p}{(2+4p)(3-2p)} - \frac{3}{2} = \frac{p(6p-1)}{(1+2p)(3-2p)} < 0$. Therefore, in this case, choosing the better option between the two proves the lemma. □

Lemma 5. [5] *Given a Steiner tree T_r such that:*

- $2k < |D_r| \leq \frac{5}{2}k$;
- *root node r has exactly two child nodes v_1, v_2, and $k < |D_{v_1}|, |D_{v_2}| < \frac{4}{3}k$;*

- *for* $i = 1, 2$, *within* T_{v_i}, *there is a node* u_i *which has exactly two child nodes* x_{i1} *and* x_{i2};
- *for* $i = 1, 2$, $|D_{x_{i1}}| < \frac{2}{3}k$, $|D_{x_{i2}}| < \frac{2}{3}k$, *and* $|D_{x_{i1}}| + |D_{x_{i2}}| > k$.

It is always possible to partition T_r *into three subtrees of size* $\leq k$, *such that the total routing cost for these subtrees is* $\leq \frac{5}{4}w(T_r) + \frac{3}{2} \times \frac{1}{k}\sum_{d \in D_r} w(s, d)$.

Lemma 6. [4] *Given a Steiner tree* T_r *such that*

- $\frac{5}{2}k < |D_r| < \frac{8}{3}k$;
- *root node* r *has exactly two child nodes* v_1, v_2, *and* $k < |D_{v_1}|, |D_{v_2}| < \frac{4}{3}k$;
- *for* $i = 1, 2$, *within* T_{v_i}, *there is a node* u_i *which has exactly two child nodes* x_{i1}, x_{i2};
- *for* $i = 1, 2$, $|D_{x_{i1}}| < \frac{2}{3}k$, $|D_{x_{i2}}| < \frac{2}{3}k$, *and* $|D_{x_{i1}}| + |D_{x_{i2}}| > k$.

It is always possible to partition T_r *into three or four subtrees of size* $\leq k$, *such that the total routing cost for these subtrees is* $\leq \frac{5}{4}w(T_r) + \frac{3}{2} \times \frac{1}{k}\sum_{d \in D_r} w(s, d)$.

Proof. The proof of this lemma is included in Appendix for reviewing purpose, since the submission to Theoretical Computer Science is still under review. □

3 Conclusions

We have presented a $(\frac{5}{4}\rho + \frac{3}{2})$-approximation algorithm for kMTR. This performance ratio was targeted in several previous works [16,5,4], but never achieved. We conjecture that this ratio is the best possible by this line of tree-partitioning based approximation algorithms. To design better approximations, certain new techniques other than tree-partitioning might need to be introduced.

References

1. Altinkemer, K., Gavish, B.: Heuristics with Constant Error Guarantees for the Design of Tree Networks. Management Science 34, 331–341 (1988)
2. Bharath-Kumar, K., Jaffe, J.M.: Routing to Multiple Destinations in Computer Networks. IEEE Transaction on Communications 31, 343–351 (1983)
3. Cai, Z.: Improved Algorithms for Multicast Routing and Binary Fingerprint Vector Clustering. Master's Thesis, Department of Computing Science, University of Alberta (2004)
4. Cai, Z., Chen, Z.-Z., Lin, G.-H.: A 3.4713-Approximation Algorithm for the Capacitated Multicast Tree Routing Problem. The Special Issue of Theoretical Computer Science for COCOA 2008 (submitted) (2009)
5. Cai, Z., Chen, Z.-Z., Lin, G.-H., Wang, L.: An Improved Approximation Algorithm for the Capacitated Multicast Tree Routing Problem. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 286–295. Springer, Heidelberg (2008)
6. Cai, Z., Lin, G.-H., Xue, G.: Improved Approximation Algorithms for the Capacitated Multicast Routing Problem. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 136–145. Springer, Heidelberg (2005)

7. Christofides, N.: Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University (1976)
8. Gröpl, C., Hougardy, S., Nierhoff, T., Prömel, H.J.: Approximation Algorithms for the Steiner Tree Problem in Graphs. In: Du, D.-Z., Cheng, X. (eds.) Steiner Trees in Industries, pp. 235–279. Kluwer Academic Publishers, Dordrecht (2001)
9. Gu, J., Hu, X.D., Jia, X., Zhang, M.-H.: Routing Algorithm for Multicast under Multi-tree Model in Optical Networks. Theoretical Computer Science 314, 293–301 (2004)
10. Hu, X.-D., Shuai, T.-P., Jia, X., Zhang, M.-H.: Multicast Routing and Wavelength Assignment in WDM Networks with Limited Drop-offs. In: Proceedings of IEEE INFOCOM (2004)
11. Huitema, C.: Routing in the Internet. Prentice Hall PTR, Englewood Cliffs (2000)
12. Jothi, R., Raghavachari, B.: Approximation Algorithms for the Capacitated Minimum Spanning Tree Problem and its Variants in Network Design. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 805–818. Springer, Heidelberg (2004)
13. Kuo, F., Effelsberg, W., Garcia-Luna-Aceves, J.J.: Multimedia Communications: Protocols and Applications. Prentice Hall, Inc., Englewood Cliffs (1998)
14. Libeskind-Hadas, R.: Efficient Collective Communication in WDM Networks with a Power Budget. In: Proceedings of the Ninth IEEE Conference on Computer Communications and Networks, pp. 612–616 (2000)
15. Lin, G.-H.: An Improved Approximation Algorithm for Multicast k-Tree Routing. Journal of Combinatorial Optimization 9, 349–356 (2005)
16. Morsy, E., Nagamochi, H.: An Improved Approximation Algorithm for Capacitated Multicast Routings in Networks. Theoretical Computer Science 390, 81–91 (2008)
17. Robins, G., Zelikovsky, A.Z.: Improved Steiner Tree Approximation in Graphs. In: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 770–779 (2000)
18. Sahasrabuddhe, L.H., Mukherjee, B.: Multicast Routing Algorithms and Protocols: a Tutorial. IEEE Networks 14, 90–102 (2000)
19. Wang, Z., Crowcroft, J.: Quality-of-Service Routing for Supporting Multimedia Applications. IEEE Journal on Selected Areas in Communications 14, 1228–1234 (1996)
20. Zhang, X., Wei, J., Qiao, C.: Constrained Multicast Routing in WDM Networks with Sparse Light Splitting. In: Proceedings of IEEE INFOCOM, pp. 1781–1790 (2000)

Appendix: Proof of Lemma 6

The proof is provided here since Ref [4] is still under review.

Proof. We give two options to route all the destination nodes in D_r. In the first option, we obtain four routing trees each of size at most k by applying Lemma 2 separately to T_{v_1} and to the branch rooted at r and containing v_2. The total routing cost of these four resultant trees is $w_1 \leq w(T_r) + 2 \times \frac{1}{k} \sum_{d \in D_r} w(s,d)$.

We next describe the second option. For each $i \in \{1,2\}$ and each $j \in \{1,2\}$, let $B_{i,j}$ be the branch rooted at u_i and containing $x_{i,j}$. For each $i \in \{1,2\}$, let B_i be the branch rooted at r and containing v_i. Let T_3 be the tree obtained from B_1 by deleting $x_{1,1}$, $x_{1,2}$, and their descendants. Similarly, let T_4 be the tree obtained from B_2 by deleting $x_{2,1}$, $x_{2,2}$, and their descendants. Clearly, $D_r = D_{x_{1,1}} \cup D_{x_{1,2}} \cup (D_r \cap T_3) \cup (D_r \cap T_4) \cup D_{x_{2,1}} \cup D_{x_{2,2}}$. Moreover, $|D_r \cap T_3| \leq |D_{v_1}| - |D_{x_{1,1}}| - |D_{x_{1,2}}| + 1 \leq (\frac{4}{3}k - 1) - (k+1) + 1 < \frac{1}{3}k$. Similarly, $|D_r \cap T_4| < \frac{1}{3}k$. We distinguish two cases as follows.

Case 1: $w(B_{1,2}) + w(B_{2,2}) \leq \min\{w(T_4) + w(B_{1,1}), w(T_3) + w(B_{2,1})\}$. In this case, $w(B_{1,2}) + w(B_{2,2}) \leq \frac{1}{3} w(T_r)$. We find a subset Q_1 of $D_{x_{1,2}}$ such that $|D_{x_{1,1}}| + |Q_1| = \lceil \frac{1}{3}|D_r| \rceil$. Set Q_1 exists because $|D_{x_{1,1}}| < \frac{2}{3}k$, $|D_{x_{1,2}}| < \frac{2}{3}k$, $|D_{x_{1,1}}| + |D_{x_{1,2}}| > k$, and $\frac{5}{6}k < \lceil \frac{1}{3}|D_r| \rceil \leq k$. Similarly, we find a subset Q_2 of $D_{x_{2,2}}$ such that $|D_r \cap B_{2,1}| + |Q_2| = \lfloor \frac{1}{3}|D_r| \rfloor$. Set Q_2 exists because $|D_{x_{2,1}}| < \frac{2}{3}k$, $|D_{x_{2,2}}| < \frac{2}{3}k$, $|D_{x_{2,1}}| + |D_{x_{2,2}}| > k$, and $\frac{5}{6}k \leq \lfloor \frac{1}{3}|D_r| \rfloor \leq k$. We are now ready to construct three routing trees X_1, X_2, and X_3 as follows. For $i \in \{1,2\}$, we construct X_i by initializing it as T_{u_i}, with its destination node set being $D_{u_i} - (D_{x_{i,2}} - Q_i)$ and minus u_i if it has been routed. We construct X_3 from T_r by deleting $x_{1,1}$, $x_{2,1}$, and all of their descendants, and further excluding nodes of $Q_1 \cup Q_2$ from the routing. X_3 has its destination node set $D_r - D_{x_{1,1}} - D_{x_{2,1}} - Q_1 - Q_2$. Note that $|D_r \cap X_3|$ is equal to $\lfloor \frac{1}{3}|D_r| \rfloor$ or $\lceil \frac{1}{3}|D_r| \rceil$. Obviously, the total routing cost of X_1, X_2, and X_3 is $w_2 \leq \frac{4}{3} w(T_r) + \frac{6}{5} \times \frac{1}{k} \sum_{d \in D_r} w(s,d)$. Since $\min\{w_1, w_2\} \leq \frac{1}{4} w_1 + \frac{3}{4} w_2 \leq \frac{5}{4} w(T_r) + \frac{7}{5} \times \frac{1}{k} \sum_{d \in D_r} w(s,d)$. So, choosing the better option among the two proves the lemma.

Case 2: $w(B_{1,2}) + w(B_{2,2}) > \min\{w(T_4) + w(B_{1,1}), w(T_3) + w(B_{2,1})\}$. Without loss of generality, we assume that $w(T_4) + w(B_{1,1}) \leq w(T_3) + w(B_{2,1})$. Then, $w(T_4) + w(B_{1,1}) < \frac{1}{3} w(T_r)$. We find a subset Q_1 of $D_{x_{1,1}}$ such that $|Q_1| + |D_{x_{1,2}}| = k$. Note that $|D_{x_{1,1}} - Q_1| + |D_r \cap T_3| \leq \frac{1}{3}k$ because $|D_{v_1}| < \frac{4}{3}k$. We are now ready to construct three routing trees X_1, X_2, and X_3 as follows. We construct X_1 by initializing it as T_{u_1}, with its destination node set being $D_{u_1} - (D_{x_{1,1}} - Q_1)$ and minus u_1 if it has been routed. Note that the connection cost of X_1 is $c(X_1) \leq \frac{1}{k} \sum_{d \in (D_{x_{1,1}} - Q_1) \cup D_{x_{1,2}}} w(s,d)$.

We construct X_2 and X_3 as follows. First, we obtain a subtree Y from T_r by removing $x_{1,2}$, $x_{2,1}$, $x_{2,2}$, and all of their descendants, and further excluding nodes of $Q_1 \cup (D_r \cap T_4)$ from the routing. Note that $|D_r \cap Y| = |D_{x_{1,1}} - Q_1| + |D_r \cap T_3| \leq \frac{1}{3}k$. We then sort the nodes in $D_r - D_{x_{1,2}} - Q_1$ in ascending order of their distances to s in G. Let F contain the last $\frac{3}{4}k$ nodes in the sorted sequence,

and let C contain the other nodes in the sequence. Since $|D_r - D_{x_{1,2}} - Q_1| \geq \frac{3}{2}k$, we have $|C| \geq \frac{3}{4}k$. We now look at the four sets: $D_{x_{2,1}}$, $D_{x_{2,2}}$, $D_r \cap T_4$, and $D_r \cap Y$. Each of the first two sets is of size at most $\frac{2}{3}k - 1$ while each of the last two sets is of size at most $\frac{1}{3}k$. Thus, at least two of the four sets contain at least one node of C. Consequently, we can always divide the four sets into two groups $\mathcal{G}_1$ and $\mathcal{G}_2$ that satisfy the following two conditions:

1. $\mathcal{G}_1$ contains $D_{x_{2,1}}$ and one of $D_r \cap T_4$ and $D_r \cap Y$, while $\mathcal{G}_2$ contains $D_{x_{2,2}}$ and the other of $D_r \cap T_4$ and $D_r \cap Y$. (*Comment:* The total size of sets in $\mathcal{G}_1$ is at most k and the total size of sets in $\mathcal{G}_2$ is at most k.)
2. At least one set in $\mathcal{G}_1$ contains a node of C and at least one set in $\mathcal{G}_2$ contains a node of C.

If $\mathcal{G}_1$ contains $D_r \cap T_4$, then we let X_2 be the union of $B_{2,1}$ and T_4 and let X_3 be the union of $B_{2,2}$ and Y; otherwise, we let X_2 be the union of $B_{2,1}$ and Y and let X_3 be the union of $B_{2,2}$ and T_4. In any case, we exclude u_2 in one of X_2 and X_3 for routing if it has been routed in T_r. By Condition 1, $|D_r \cap X_2| \leq k$ and $|D_r \cap X_3| \leq k$. By Condition 2, $(D_r \cap X_2) \cap C \neq \emptyset$ and $(D_r \cap X_3) \cap C \neq \emptyset$. Obviously, one of $D_r \cap X_2$ and $D_r \cap X_3$ contains d' which is the closest destination node to s among the nodes in $(D_r \cap X_2) \cup (D_r \cap X_3)$. We assume that $D_r \cap X_2$ contains d'; the other case is similar. It follows that the connection cost $c(X_2) \leq w(s, d') \leq \frac{4}{3} \times \frac{1}{k} \sum_{d \in C} w(s, d)$. Moreover, since $(D_r \cap X_3) \cap C \neq \emptyset$, the connection cost $c(X_3) \leq w(s, d'')$ where d'' is the farthest destination node from s among the nodes in C. Furthermore, since $C \cap F = \emptyset$, $w(s, d'') \leq w(s, d''')$ where d''' is the closest destination node to s among the nodes in F. Thus, $c(X_3) \leq w(s, d''') \leq \frac{4}{3} \times \frac{1}{k} \sum_{d \in F} w(s, d)$. Therefore, $c(X_2) + c(X_3) \leq \frac{4}{3} \times \frac{1}{k} \sum_{d \in D_r - (D_r \cap X_1)} w(s, d)$. Consequently, the total routing cost of X_1, X_2, and X_3 is $w_2 \leq \frac{4}{3} w(T_r) + \frac{4}{3} \times \frac{1}{k} \sum_{d \in D_r} w(s, d)$, because $w(X_1) + w(X_2) + w(X_3) = w(T_r) + w(B_{1,1}) + w(T_4) \leq \frac{4}{3} w(T_r)$. Finally, since $\min\{w_1, w_2\} \leq \frac{1}{4} w_1 + \frac{3}{4} w_2 \leq \frac{5}{4} w(T_r) + \frac{3}{2} \times \frac{1}{k} \sum_{d \in D_r} w(s, d)$, choosing the better option among the two proves the lemma. □

On Disjoint Shortest Paths Routing on the Hypercube

Eddie Cheng[1], Shuhong Gao[2,⋆], Ke Qiu[3], and Zhizhang Shen[4]

[1] Dept. of Mathematics and Statistics, Oakland University, USA
echeng@oakland.edu
[2] Dept. of Mathematical Science, Clemson University, USA
sgao@math.clemson.edu
[3] Dept. of Computer Science, Brock University, Canada
kqiu@brocku.ca
[4] Dept. of Computer Science and Technology, Plymouth State University, USA
zshen@plymouth.edu

Abstract. We present a routing algorithm that finds n disjoint shortest paths from the source node to n target nodes in the n-dimensional hypercube in $O(n^3) = O(\log^3 N)$ time, where $N = 2^n$, provided that such disjoint shortest paths exist which can be verified in $O(n^{5/2})$ time, improving the previous $O(n^3 \log n)$ routing algorithm. In addition, the development of this algorithm also shows strong relationship between the problems of the disjoint shortest paths routing and matching.

1 Introduction

One of several well-known disjoint path paradigms [1,2,16] for any interconnection network is as follows: given a source node and k target nodes, find k disjoint paths from the source to each target node (node-to-set). Other paradigms include node-to-node and set-to-set. There are large amount of work done on many well-known interconnection networks for various paradigms that are too numerous to list, among them, vast number of publications in the literature are devoted to various routing algorithms on the hypercube alone. This specific node-to-set routing problem has also been studied extensively for the hypercube in, for example, [6,10,11], to list just a few. Algorithms for this routing are shown to be useful in designing efficient and fault tolerant routings for the corresponding network [2].

In this paper, we study a special version of this disjoint path problem for the hypercube: given a fixed node (source) and n other nodes (targets) in a hypercube of dimension n, find n disjoint *shortest* paths from the source node to all target nodes, where two paths are disjoint if they do not share any node except the source node, if these paths do exist. Because of the symmetry, without loss of generality, from now on, we assume that the source is the node 0. To the

⋆ Partially supported by the National Science Foundation under Grant DMS-0302549 and National Security Agency under grant H98230-08-1-0030.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 375–383, 2009.

best of our knowledge, [13] and [4] are two of the earliest works on this subject. Most of the routing algorithms for the hypercube find disjoint paths from 0 to n target nodes in an n-cube with $N = 2^n$ nodes whose lengths are bounded but are not necessarily the shortest. For example, it is shown in [15] that there exist n disjoint paths in an n-cube such that all paths have lengths no longer than $n+1$ while in [13], it is shown that n disjoint paths exist such that at most one path has length $n+1$ and all other paths have length no more than n. Please refer to [3] for the general problem of finding the *Rabin Number* of interconnection networks. Note that a path from 0 to node u is shortest if $d(0,u) = H(u)$, the Hamming weight of u. Also note that some routing algorithms may find the disjoint paths that are shortest possible for the set of target nodes but not the shortest.

When we impose the condition that these disjoint paths be the shortest, it is clear that such paths exist only for certain sets of n target nodes in an n-cube. A necessary and sufficient condition was presented for such paths (disjoint and shortest) to exist for a set of n target nodes in [13]. This condition was later generalized in [4]. In addition, an $O(n^4) = O(\log^4 N)$ routing algorithm was given to find n disjoint shortest paths in cases they do exist. An $O(n^3 \log n)$ algorithm was given later in [14].

In the next section, we present an $O(n^3)$ routing algorithm that finds n disjoint shortest paths from n target nodes to the node 0, thus improving the previous algorithms given in [4] and [14]. The possible relationship among different disjoint path paradigms is briefly discussed in Section 3.

2 Node-to-Set Disjoint Shortest Paths Routing

Before we present our routing algorithm, we briefly describe the following result given in [13,4]:

Given n non-zero distinct target nodes u_1, u_2, ..., u_n in a hypercube of dimension n with $N = 2^n$ nodes, there exist n disjoint shortest paths from node 0 to them if and only if for any $1 \leq k \leq n-1$, at most k nodes in the set of target nodes all have 0 at the same $n-k$ coordinates.

Although the proof was somewhat involved, simply stated, this condition says that such paths exist if and only if no k-subcube containing the node 0 contains more than k target nodes.

We can state this condition yet another way that is crucial to developing our routing algorithm as follows.

Assume that the n target nodes' binary representations are

$$\begin{aligned} &u_{11}u_{12}...u_{1n}, \\ &u_{21}u_{22}...u_{2n}, \\ &\qquad\vdots \\ &u_{n1}u_{n2}...u_{nn}. \end{aligned}$$

The binary representations of the n nodes can be viewed as the adjacency matrix of a bipartite graph and the condition that for any $1 \leq k \leq n-1$, at most k nodes in the set of target nodes all have 0 at the same $n-k$ coordinates means that Hall's condition [8] is satisfied, i.e., n disjoint shortest paths exist if and only if there exists a permutation $i_1 i_2 \cdots i_n$ of symbols from $\{1, 2, \cdots, n\}$ such that $u_{1,i_1} = u_{2,i_2} = \cdots = u_{n,i_n} = 1$. This is a direct result of Corollary 1 [4]. In other words, there exists a perfect matching for the bipartite graph or, equivalently, there is an optimal solution with cost n to the corresponding classic assignment problem [12] represented by the cost matrix which is the set of binary representations (note that to view the problem as an assignment problem, all 0 entries should be changed to a fixed value greater than 1). This description of the necessary and sufficient condition plays a critical role in our routing algorithm.

For example, for the following four nodes represented as a 4×4 cost matrix

$$\begin{pmatrix} 0\,1\,1\,0 \\ 1\,0\,0\,0 \\ 0\,1\,1\,1 \\ 1\,0\,0\,1 \end{pmatrix}$$

it represents a bipartite graph $G = (\{A, B, C, D, a, b, c, d\}, \{(A, b), (A, c), (B, a), (C, b), (C, c), (C, d), (D, a), (D, d)\})$. One possible assignment is

$$\begin{pmatrix} 0 & 1 & \textcircled{1} & 0 \\ \textcircled{1} & 0 & 0 & 0 \\ 0 & \textcircled{1} & 1 & 1 \\ 1 & 0 & 0 & \textcircled{1} \end{pmatrix}$$

where the assignment solution is circled. This particular assignment is equivalent to a perfect matching of $\{(A, c), (B, a), (C, b), (D, d)\}$. Note that the solution to the assignment problem (and thus the perfect matching problem) is not unique.

Finding a maximum (perfect) matching for a bipartite graph with $2n$ vertices can be done in $O(n^{5/2})$ [9].

We will first describe a key step in our routing algorithm in the next subsection. We then discuss its implementation and the analysis.

2.1 Node Reduction

The key step in our routing algorithm is based on the following simple observation:

Theorem 1. *Given n non-zero nodes satisfying the necessary and sufficient condition for n disjoint shortest paths to exist,*

$$\begin{aligned} u_1 &= u_{11}u_{12}...u_{1n} \\ u_2 &= u_{21}u_{22}...u_{2n} \\ &\vdots \\ u_n &= u_{n1}u_{n2}...u_{nn}, \end{aligned}$$

then for any node u_i, we can find $u_i^{'}$ such that $H(u_i^{'} \oplus u_i) = 1$, that is, $H(u_i^{'}) = H(u_i) - 1$ and $(u_i, u_i^{'})$ is an edge in the hypercube. Furthermore, if $H(u_i) > 1$, i.e., $u_i^{'} \neq 0$, then nodes u_1, u_2, ..., u_{i-1}, $u_i^{'}$, u_{i+1}, ..., u_n still satisfy the necessary and sufficient condition.

The theorem is trivially true since we can simply take the node next to u_i on its shortest path to 0 as $u_i^{'}$. We call the process to find such an $u_i^{'}$ from u_i a *reduction*. We now show how a reduction can be performed.

We assume that an assignment has been found.

For node u_i, for each 1 other than the 1 in the assignment solution, remove it and see whether the resulting $u_i^{'}$ is not equal to any of the other $n-1$ nodes. If such a 1 exists, remove it and we have found our $u_i^{'}$. Otherwise (removing any such a 1 results in a node equal to u_j for some $j \in \{1, 2, \cdots, n\} - \{i\}$), the situation can be illustrated below when removing the bold **1** will result in u_i becoming u_j:

$$
\begin{array}{ll}
u_1 & \\
u_2 & \\
\vdots & \\
u_j & u_{j1}u_{j2}\cdots 0\cdots \textcircled{1}\cdots 1\cdots u_{jn} \\
\vdots & \\
u_i & u_{j1}u_{j2}\cdots \mathbf{1}\cdots 1\cdots \textcircled{1}\cdots u_{jn} \\
\vdots & \\
u_n &
\end{array}
$$

Note that because the n target nodes satisfy the necessary and sufficient condition, there have to be two indices l and m ($l \neq m$) such that

$$u_{jl} = u_{jm} = u_{il} = u_{im} = 1,$$

and u_{jl} and u_{im} are part of the assignment. We can get a new assignment by switching the two ①'s while keeping the remaining assignment intact as follows:

$$
\begin{array}{ll}
u_1 & \\
u_2 & \\
\vdots & \\
u_j & u_{j1}u_{j2}\cdots 0\cdots 1\cdots \textcircled{1}\cdots u_{jn} \\
\vdots & \\
u_i & u_{j1}u_{j2}\cdots 1\cdots \textcircled{1}\cdots \mathbf{1}\cdots u_{jn} \\
\vdots & \\
u_n &
\end{array}
$$

Note that because the nodes do satisfy the necessary and sufficient condition for disjoint shortest paths to exist and crossing any 1 in u_i other than **1** results in another target node, we know for sure that removing the **1** will generate a node that is different from any other target node. More precisely, let $v = v_1 v_2 \cdots v_n$ be the node such that v and u_i only differ at position m, namely, $v_k = u_{ik}$ for $k \neq m$, $u_{im} = 1$, and $v_m = 0$. Let $P = \{p \in \{1, 2, \cdots, n\} - \{m\} | u_{ip} = 1\}$, i.e., the collection of positions except at the position m where u_i takes a 1. By assumption, for every $p \in P$, there is a target node u such that $u = u_i$, except that u has a 0 at position p. Hence, the assignment as guaranteed by the sufficient and necessary condition must include a position $p' \in P - \{p\}$, where u has a 1 at position p'. Since every such a position p corresponds to such a u, every position in P is included in the aforementioned assignment for $|P|$ such nodes.

Now consider v. For all positions $q \notin P$, including position m, v has a 0 at position q. Therefore, no such position q may be included in the assignment for v. Furthermore, by the definition of such an assignment, no position $p \in P$, at which position v has a 1, can be included in the assignment for v, either. Thus, none of the 1's in v can be "circled", a contradiction to the assumption that "an assignment has been found." Hence, v is indeed different from any other target node.

Now we can remove the 1 (in bold) formerly as part of the assignment in u_i to get $u_i^{'}$ and clearly, the new set of nodes still satisfy the necessary and sufficient condition as an assignment still exists, $H(u_i \oplus u_i^{'}) = 1$, and subsequently, $H(u_i^{'})$ $= H(u_i)$ - 1.

2.2 Implementation and Analysis

We now discuss how to perform the above reduction and its time.

Before the reduction, the following pre-processing is done:

1. Convert the n binary representations into integers, also compute the Hamming weights of all target nodes. Both sequences are then sorted.
2. Find an assignment (perfect matching).

Step 1 takes $O(n^2 + n \log n) = O(n^2)$ time while Step 2 takes $O(n^{5/2})$ time. Note that both steps are done only once at the very beginning of the entire routing algorithm.

Note that in order to reduce a node u_i, if we delete 1's that are not in the assignment solution, in the order from the most significant position to the least significant position, we will get a sorted sequence of numbers. This observation leads to an $O(n)$ reduction:

If deleting a 1 results in a number v that is not equal to any of the remaining $n - 1$ target nodes ($O(1)$ time), delete this 1, the new value v is inserted into the list of sorted targets ($O(n)$ time). Similarly, the Hamming weight of the new node is also computed in constant time since it is $H(u_i)$ - 1 and inserted to a sorted list of Hamming weights.

For example, for the cost matrix

$$\begin{array}{l} u_1 : 1 \quad 0 \quad 1 \quad \textcircled{1} \\ u_2 : 0 \quad 0 \quad \textcircled{1} \quad 1 \\ u_3 : 1 \quad \textcircled{1} \quad 0 \quad 1 \\ u_4 : \textcircled{1} \quad 0 \quad 1 \quad 0 \end{array}$$

and we want to reduce u_1. The sorted sequence of target nodes is 3, 10, (11,) 13. Crossing out the first 1 results in 3; Crossing out the second 1 results in 9. So we delete the second 1 and insert 9 into the sorted sequence of target nodes.

On the other hand, if no matter which 1 is deleted, the resulting node becomes one of the remaining target nodes, as illustrated in the two cases below where node u_1 is to be reduced, then we can simply find any one of the remaining target nodes and perform the reduction (by switching the 1's in the assignment first) and the updating mentioned above. The time for this operation is also $O(n)$.

$$\begin{array}{l} u_1 : 1 \quad 1 \quad \textcircled{1} \\ u_2 : 0 \quad \textcircled{1} \quad 1 \\ u_3 : \textcircled{1} \quad 0 \quad 1 \end{array}$$

$$\begin{array}{l} u_1 : 1 \quad 0 \quad \textcircled{1} \quad 1 \\ u_2 : 0 \quad 0 \quad 1 \quad \textcircled{1} \\ u_3 : 1 \quad \textcircled{1} \quad 0 \quad 1 \\ u_4 : \textcircled{1} \quad 0 \quad 1 \quad 0 \end{array}$$

Therefore, a node reduction can be done in $O(n)$ time.

2.3 The Routing Algorithm

We can now state the routing algorithm as follows:

1. While there exist nodes whose Hamming weights are all greater than 1, select one node w such that $H(w) = \max_u\{H(u) > 1\}$; (The Hamming weights of nodes can be computed at the beginning of the algorithm and updated after each reduction so that this step takes $O(n)$ time)
2. Perform a reduction as described above.

The routing algorithm is to simply apply the reduction $O(n^2)$ times, as there are $O(n^2)$ 1's in the n target nodes, until they are reduced to an assignment solution, namely, the n neighbours of the node 0.

As for the total time $t(n)$ of the routing algorithm for an n-cube, we need $O(n^2)$ time to convert the n nodes to integer values, $O(n \log n)$ time to sort them, $O(n^{5/2})$ to find a perfect matching to get an assignment (these two steps need to be done only once), and perform the $O(n)$-time reduction $O(n^2)$ times, resulting in a total time of $O(n^3) = O(\log^3 N)$.

Note that from the algorithm we always pick a node with the largest Hamming weight to reduce at any time. This is important in order to avoid routing conflict as described below.

$$\begin{array}{c}\vdots\\ \cdots\bar{1}\cdots 1\cdots 1\cdots : u\\ \vdots\\ \cdots 1\cdots 1\cdots 1\cdots 1\cdots : v\\ \vdots\end{array}$$

where $H(u) = 3$ and $H(v) = 4$ and $\bar{1}$ indicates that the 1 was previously deleted from u. If u is reduced first, we would have such a situation as described above. Then when v is reduced, the leftmost 1 is tried first and it is found that the resulting node is different from any of the other node, so that 1 is incorrectly removed, while in fact, the resulting node is actually u.

The proof that the paths so obtained are disjoint is thus clear: if after a reduction, the resulting node becomes a node that is reduced earlier, then it means that a node with smaller Hamming weight is reduced earlier, a contradiction.

Similarly, the paths found are the shortest since for each node, a 1 is removed each time.

We end this section by an example. Given $n = 5$ nodes as follows:

$$\begin{array}{l} u_1 : \textcircled{1}\ 1\ 0\ 0\ 0\\ u_2 : 0\ 1\ \textcircled{1}\ 0\ 0\\ u_3 : 1\ \textcircled{1}\ 1\ 0\ 0\\ u_4 : 0\ 0\ 0\ \textcircled{1}\ 1\\ u_5 : 0\ 0\ 1\ 1\ \textcircled{1}\end{array}$$

with assignment solution circled. Since $H(u_3) = 3$, we do the reduction on u_3. We first try the first 1 (from the left) and the resulting node is u_2. We then try the third 1 and the resulting node is u_1. We then switch the two 1's in the assignment solution in u_1 and u_3 and remove the second 1 in u_3 to get:

$$\begin{array}{l} u_1 : 1\ \textcircled{1}\ 0\ 0\ 0\\ u_2 : 0\ 1\ \textcircled{1}\ 0\ 0\\ u_3 : \textcircled{1}\ 0\ 1\ 0\ 0\\ u_4 : 0\ 0\ 0\ \textcircled{1}\ 1\\ u_5 : 0\ 0\ 1\ 1\ \textcircled{1}\end{array}$$

The next node to reduce is u_5. When the third 1 is removed, the resulting node becomes u_4. So we remove the fourth 1 to obtain

$$\begin{array}{l} u_1 : 1\ \textcircled{1}\ 0\ 0\ 0\\ u_2 : 0\ 1\ \textcircled{1}\ 0\ 0\\ u_3 : \textcircled{1}\ 0\ 1\ 0\ 0\\ u_4 : 0\ 0\ 0\ \textcircled{1}\ 1\\ u_5 : 0\ 0\ 1\ 0\ \textcircled{1}\end{array}$$

The next five reductions are straightforward and will end up with:

$$\begin{array}{l} u_1 : 0\ \textcircled{1}\ 0\ 0\ 0 \\ u_2 : 0\ 0\ \textcircled{1}\ 0\ 0 \\ u_3 : \textcircled{1}\ 0\ 0\ 0\ 0 \\ u_4 : 0\ 0\ 0\ \textcircled{1}\ 0 \\ u_5 : 0\ 0\ 0\ 0\ \textcircled{1} \end{array}$$

The five disjoint shortest paths are:

$$\begin{array}{l} 11000 \rightarrow 01000 \rightarrow 00000 \\ 01100 \rightarrow 00100 \rightarrow 00000 \\ 11100 \rightarrow 10100 \rightarrow 10000 \rightarrow 00000 \\ 00011 \rightarrow 00010 \rightarrow 00000 \\ 00111 \rightarrow 00101 \rightarrow 00001 \rightarrow 00000 \end{array}$$

3 Conclusion

We have developed an $O(n^3) = O(\log^3 N)$ routing algorithm that route n nodes to the node 0 in an n-cube with $N = 2^n$ nodes, such that all n paths are disjoint and shortest, provided that these paths exist. This time is an improvement over the previously best known $O(n^3 \log n)$ routing algorithm.

A trivial lower bound to this particular routing problem of finding disjoint shortest paths in an n-cube is $\Omega(n^2)$ as there are n paths each of length $O(n)$ (Indeed, if we drop the condition that the disjoint paths be the shortest, then n-disjoint paths can be found in the optimal $O(n^2)$ time [6]). It remains to be seen whether a better algorithm can be found. It is also clear from our discussion that disjoint shortest paths for n nodes in an n-cube imply a perfect matching. It remains to be seen what other possibly deeper relations exist between the problems of finding disjoint shortest paths and matching.

It is worth pointing out that the node-to-set disjoint shortest paths problem is a special instance of the disjoint shortest paths for pairs of vertices where given m pairs of vertices $[u_i, v_i]$, $1 \leq i \leq m$, we are to find m disjoint shortest paths from u_i to v_i such that each path is the shortest with distance $H(u_i \oplus v_i)$ and these paths are disjoint [5] (A more general discussion of the problem of finding pairwise disjoint paths in the hypercube can be found in [7]). This is so because after the first step in the node-to-set disjoint shortest paths routing, it becomes a routing from these n neighbours of 0 to the n target nodes. (An $O(n^4)$ routing algorithm is given in [5] for a specific case of this problem for pairs where $H(u_i \oplus v_i) = n$.) Therefore, an efficient routing algorithm for the pair-wise routing problem could be useful for our problem. Of course, before we use such an algorithm, we have to pair these vertices (target nodes and the neighbours of node 0) which itself is nontrivial.

It is also tempting to relate the two routing problems in other ways. For example, one intuition is that there exist m disjoint shortest paths for m pairs

if and only if there exist a source node in the m-cube such that there exist m disjoint shortest paths from the source node to m target nodes u_i or v_i, $1 \leq i \leq m$, and these paths pass through v_i or u_i. However, although this is indeed the case for some instances, examples can be found where such a claim is not true. Nevertheless, it is hoped that our approach can be used in routing algorithms for other disjoint paths paradigms in some way.

References

1. Chen, C.C., Chen, J.: Nearly Optimal One-to-Many Parallel Routing in Star Networks. IEEE trans. on Parallel and Distributed Systems 8(12), 1196–1202 (1997)
2. Dietzfelbinger, M., Madhavapeddy, S., Sudborough, I.H.: Three Disjoint Path Paradigms in Star Networks. In: Proc. of the 3rd IEEE Symposium on Parallel and Distributed Processing, pp. 400–406. IEEE Computer Society Press, Dallas (1991)
3. Duh, D.R., Chen, G.H.: On the Rabin Number Problem. Networks 30(3), 219–230 (1998)
4. Gao, S., Novick, B., Qiu, K.: From Hall's Matching Theorem to Optimal Routing on Hypercubes. Journal of Combinatorial Theory (B) 74(2), 291–301 (1998)
5. Gonzalez, T.F., Serena, D.: n-Cube Network: Node Disjoint Shortest Paths for Maximal Distance Pairs of Vertices. Parallel Computing 30, 973–998 (2004)
6. Gu, Q.P., Peng, S.T.: Node-to-Set and Set-to-Set Cluster Fault Tolerant Routing in Hypercubes. Parallel Computing 24, 1245–1261 (1998)
7. Gu, Q.P., Peng, S.T.: An Efficient Algorithm for the k-Pairwise Disjoint Paths Problem in Hypercubes. Journal of Parallel and Distributed Computing 60, 764–774 (2000)
8. Hall, P.: On Representatives of Subsets. J. London Math. Soc. 10, 26–30 (1935)
9. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. J. SIAM Comp. 2, 225–231 (1973)
10. Lai, C.N.: One-to-Many Disjoint Paths in the Hypercube and Folded Hypercube. Ph.D. Thesis, National Taiwan University (2001)
11. Latifi, S., Ko, H., Srimani, P.K.: Node-to-Set Vertex Disjoint Paths in Hypercube Networks. Technical Report CS-98-107, Colorado State University (1998)
12. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Dover, Mineola (1998)
13. Qiu, K., Novick, B.: Disjoint Paths in Hypercubes. Congressus Numerantium 119, 105–112 (1996)
14. Qiu, K.: An Efficient Disjoint Shortest Paths Routing Algorithm for the Hypercube. In: Proc. 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2008), pp. 43–47. IEEE Computer Society Press, Melbourne (2008)
15. Rabin, M.O.: Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. Journal of ACM 36(2), 335–348 (1989)
16. Stewart, I.A., Xiang, Y.H.: One-to-Many Node-Disjoint Paths in (n, k)-Star Graphs. Technical Report, Department of Computer Science, Durham University, UK (2008)

A New Approach for Rearrangeable Multicast Switching Networks

Hongbing Fan[1,⋆] and Yu-Liang Wu[2,⋆⋆]

[1] Wilfrid Laurier University, Waterloo, ON Canada N2L 3C5
hfan@wlu.ca
[2] The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
ylw@cse.cuhk.edu.hk

Abstract. In this paper we propose a new design for rearrangeable multicast switching networks, which uses the minimum number of intermediate nodes and a comparable number of switches. The newly designed 3-stage $N \times N$ switching network has the minimum $2N$ intermediate nodes and $O(N^{5/3})$ switches and an efficient routing algorithm, while the best known wide-sense nonblocking (and hence rearrangeable) multicast 3-stage network uses $O(N \log N/ \log \log N)$ intermediate nodes and $O(N^{3/2} \log N/ \log \log N)$ switches. The new design is constructed by adding switches to a rearrangeable unicast Clos network. The design and analysis of the design is done by a combinatorial approach, which represents a switching network as a multistage bipartite graph, and the middle stage as bipartite switch box, and routing requirements as hypergraph. The new routing algorithm is done by the edge ordering of regular hypergraphs, a technique originated from job scheduling.

1 Introduction

Our study on rearrangeable multicast switching networks was motivated by the design and implementation of on-chip reconfigurable interconnection networks for multiple applications. A specific application calls for a subset of functional modules with a specific interconnection pattern, and a switching network is used to make different interconnections through switch reconfiguration. The objectives of the on-chip switching network design is to yield a rearrangeable multicast capability with a minimized number of switches and nodes for area and power efficiency, a less number of depths for short signal delay, an efficient routing algorithm for reconfiguration computing, and a simple structure for fabrication.

Switching networks have been studied extensively in the fields of communication networks and parallel/distribuited computing since the early work of Clos [3]. Various routing capabilities such as strictly/wide-sense nonblocking, rearrangable, unicast and multicast have been proposed and studied [8]. For

⋆ Research partially supported by the NSERC, Canada.
⋆⋆ Research partially supported by RGC Earmarked Grant 2150500 and ITSP Grant 6902308, Hong Kong.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 384–394, 2009.

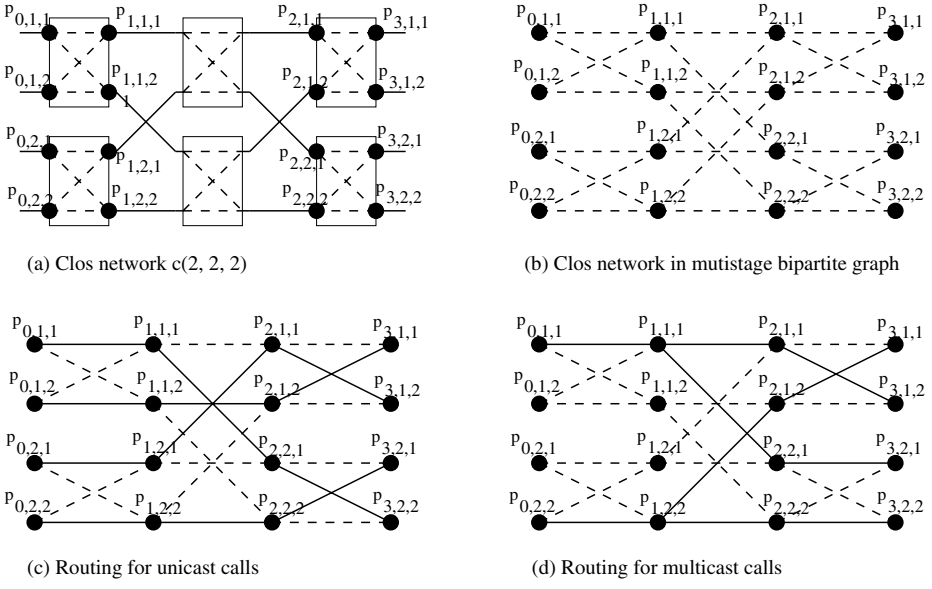
instance, the most well-known Clos network $c(m, n, r)$ is strictly nonblocking for unicast when $m \geq 2n - 1$ [3], and rearrangeable when $m \geq n$ [10,9]. For multicast, it was known that a k-stage rearrangeable multicast $N \times N$ network (of N inputs and N outputs) has a lower bound of $O(N^{1+1/k})$ switches and an upper bound of $O((N \log N)^{1+1/k})$ switches [4]. While the best known constructive design with an efficient routing algorithm is the Clos network $c(m, n, r)$ with $m \geq 2(n-1)(\log r / \log \log r) + (\log r)^{1/2}(n-1)+1$, which is wide-sense nonblocking with linear time routing algorithm. As a result, the corresponding $N \times N$ 3-stage wide-sense nonblocking (and hence rearrangeable) multicast network has $O(N^{3/2} \log N / \log \log N)$ switches [12].

In the application of reconfigurable interconnection network designs, we are primarily interested in rearrangeable multicast networks and we consider both switches and intermediate nodes as hardware cost. Since a nonblocking network is also rearrangeable, nonblocking networks can be used for reconfigurable interconnection network designs. However, the known nonblocking multicast network designs have a high cost on intermediate nodes. For instance, the wide-sense nonblocking multicast $N \times N$ Clos network in [12] has $O(N \log N / \log \log N)$ intermediate nodes, while the minimum number of intermediate nodes required in a 3-stage $N \times N$ switching network is $2N$. Our design strategy is to minimize the number of switches under the condition of using the minimum number of intermediate nodes.

In this paper we propose a new design for rearrangeable multicast switching networks, which uses the minimum number of intermediate nodes and has a low cost on the number of switches. As for 3-stage $N \times N$ networks, the new design has $O(N^{5/3})$ switches and $2N$ intermediate nodes and an efficient routing algorithm. The new design is derived by adding switches to the middle stage of the rearrangeable unicast Clos network $c(n, n, r)$, while the analysis is done by a combinatorial approach. We represent a 3-stage switching network by a 3-stage directed graph (or digraph) [8]. We then model the middle stage of a Clos type network as a switch box. By FPGA switch box design theory in [5], the 3-stage network is rearrangeable for multicast if the middle switch box is hyperuniversal. The middle switch box of Clos network $c(n, n, r)$ is not hyperuniversal, however, we proved that by adding some extra switches, a hyperuniversal switch box can be obtained. In Section 2, we give the graph definitions of switching networks, switch boxes, routing requirements, and routings in switch box and networks. Section 3 proves the rearrangebility of the newly designed network using the edge ordering of regular hypergraphs and a theorem from job scheduling.

2 Definitions

In this section we first describe the graph representation of a switching network, then give the definition of bipartite switch box and hyperuniversal switch box, and the relation between hyperuniversal switch box and rearrangeable multicast 3-stage switching network, and finally present the new rearrangeable multicast network design.



(a) Clos network c(2, 2, 2)

(b) Clos network in mutistage bipartite graph

(c) Routing for unicast calls

(d) Routing for multicast calls

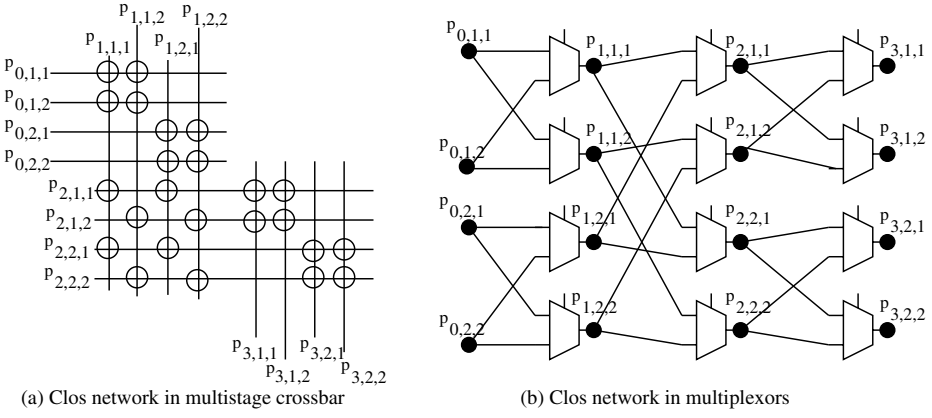
Fig. 1. Graph modeling of switching networks

2.1 The Graph Representation of Switching Networks

We use a multistage bipartite graph to represent a multistage switching network. A k-stage bipartite graph is a digraph $G = (V_0 \cup \cdots \cup V_k, E_1 \cup \cdots \cup E_k)$, where $V_0, \ldots, V_k$ are disjoint vertex sets, and each edge (or arc) in E_i is from a vertex in V_{i-1} to a vertex in V_i. The vertices of V_i are said to be at rank i, and the subgraph $G_i = (V_{i-1} \cup V_i, E_i)$ is called the i-th stage of G, and we write $G = G_1 + \cdots + G_k$.

A multistage bipartite graph represents a switching network with edges as switches and vertices as nodes. Particularly, vertices in V_0 and V_k represent input nodes and output nodes, and vertices in $V_i, 1 \le i \le k-1$ are intermediate nodes. Denoted by $e(G)$ and $n(G)$ the numbers of edges and vertices of G. With the graph modeling, the routing of a unicast (or one-to-one) call/request is a path from an input vertex to an output vertex, and the routing of a multicast (or one-to-many) call/request is a tree with root at an input vertex and leaves at output vertices. Fig.1(a), (b), (c) and (d) show the Clos network $c(2, 2, 2)$, its representation as a 3-stage bipartite graph, and a routing for the unicast calls $\{(p_{0,1,1}, \{p_{3,2,2}\}), (p_{0,1,2}, \{p_{3,1,1}\}), (p_{0,2,1}, \{p_{3,1,2}\}), (p_{0,2,2}, \{p_{3,2,1}\})\}$, and a routing for the multicast calls $\{(p_{0,1,1}, \{p_{3,1,2}, p_{3,2,1}\}), (p_{0,2,2}, \{p_{3,1,1}, p_{3,2,2}\})\}$.

A switching network modeled as a multistage bipartite graph can be implemented by a circuit of multistage crossbar, in which an edge corresponds to a crosspoint switch, and a vertex corresponds to a wire (either vertical or horizontal). Fig.2(a) shows the multistage crossbar of Fig.1(b). It can also be implemented by a multiplexor network, in which an edge corresponds to a wire and a non-input vertex corresponds to a multiplexor, see Fig.2(b). In both



(a) Clos network in multistage crossbar (b) Clos network in multiplexors

Fig. 2. Implementations of multistage switching networks

implementations, vertices and edges in the multistage bipartite graph contribute hardware costs. Therefore, we want to minimize the usage of both edges and vertices in switching network design for efficient implementations.

2.2 The Switch Box of Clos Network

We use the 3-stage Clos network as a base structure. Fig.3(a) shows the general Clos network $c(m, n, r)$, which can be represented as a 3-stage bipartite graph as, $c(m, n, r) = (V_0 \cup V_1 \cup V_2 \cup V_3, E_1 \cup E_2 \cup E_3)$, where

$$
\begin{aligned}
V_0 &= \{p_{0,j,s} : j = 1, \ldots, r, s = 1, \ldots, n\},\\
V_1 &= \{p_{1,j,s} : j = 1, \ldots, r, s = 1, \ldots, m\},\\
V_2 &= \{p_{2,j,s} : j = 1, \ldots, r, s = 1, \ldots, m\},\\
V_3 &= \{p_{3,j,s} : j = 1, \ldots, r, s = 1, \ldots, n\},\\
E_1 &= \{(p_{0,j,s}, p_{1,j,t}) : 1 \le j \le r, 1 \le s \le n, 1 \le t \le m\},\\
E_2 &= \{(p_{1,j,t}, p_{2,h,t}) : 1 \le j, h \le r, 1 \le t \le m\},\\
E_3 &= \{(p_{2,j,s}, p_{3,j,t}) : 1 \le j \le r, 1 \le s \le m, 1 \le t \le n\}.
\end{aligned}
$$

Fig.3(b) shows the graph representation of $c(n, n, r)$.

In terms of switch boxes [5], the middle stage of $c(m, n, r)$ is a bipartite switch box of r sides on each part and m vertices on each side. That is, the middle stage $G_2 = (V_1 \cup V_2, E_2)$ has part one V_1 consisting of r sides $I_j = \{p_{1,j,s} : s = 1, \ldots, m\}$, $j = 1, \ldots, r$, and part two V_2 consisting of r sides $O_j = \{p_{2,j,s} : s = 1, \ldots, m\}$, $j = 1, \ldots, r$. An edge in E_2 is joining a vertex in V_1 to a vertex in V_2. We refer to this type of switch box as $(2r, m)$-bipartite switch box (or $(2r, m)$-BSB).

In general, let $G_2' = (A_1 \cup \cdots \cup A_r \cup B_1 \cup \cdots \cup B_r, E)$ denote a $(2r, m)$-BSB, where $|A_1| = \cdots = |A_r| = |B_1| = \cdots = |B_r| = m$, and each edge in E is joining a vertex in $A_1 \cup \cdots \cup A_r$ and a vertex in $B_1 \cup \cdots \cup B_r$. A *multipin net* on a $(2r, m)$-BSB is a pair (a, b), where $a \in \{1, \ldots, r\}, b \subset \{1, \ldots, r\}$. Particularly, when

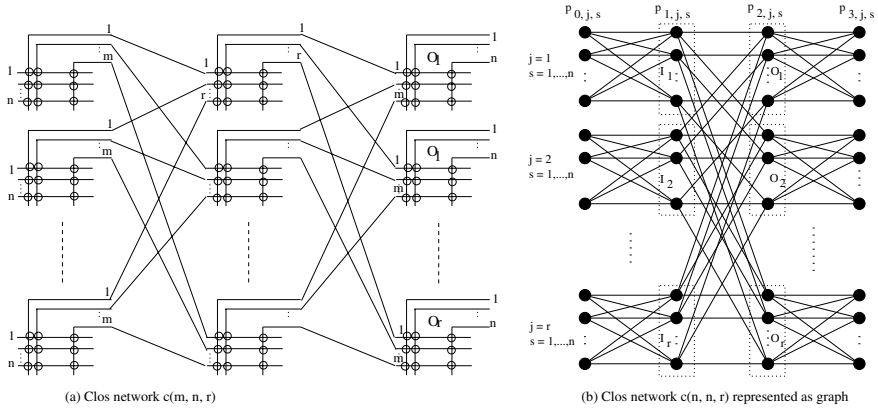


Fig. 3. Clos network $c(m, n, r)$) and its graph representation

$|b| = 1$, it is called a 2-pin net. A routing of net (a, b) in G_2' is a star shaped tree subgraph T with the root in A_a and $|b|$ leaves, one in each B_i for $i \in b$. A routing requirement on G_2' is a collection of nets $R' = \{(a_i, b_i) : i = 1, \ldots, t\}$ such that for each $j = 1, \ldots, r$, $|\{i : a_i = j, i = 1, \ldots, t\}| \leq m, |\{i : j \in b_i, i = 1, \ldots, t\}| \leq m$. This constraint means that the number of nets going to each side does not exceed m, the number of vertices on the side. The routing of R' in G_2' is a forest (a disjoint union of subtrees) $T = T_1 + \cdots + T_t$ such that T_i is a routing of $(a_i, b_i), i = 1, \ldots, t$. We say, G_2' is hyperuniversal if it has a routing for every routing requirement on G_2'. Particularly, G_2' is called universal [2] if it has a routing for every routing requirement consisting of 2-pin nets.

2.3 Hyperuniversal Bipartite Switch Box and Rearrangeable Multicast Network

Next we consider the Clos network $c(m, n, r)$ with $m = n$. The middle stage G_2 of $c(n, n, r)$ is a $(2r, n)$-BSB. Each side of the BSB is connected with a complete $n \times n$ bipartite graph in stage one/three. With such a construction, if the middle BSB is universal/hyperuniversal, then the 3-stage network is rearrangeable for unicast/multilcast. To illustrate this relation, we next show that the Clos network $c(n, n, r)$ is rearrangeable for unicast by showing that its middle $(2r, n)$-BSB G_2 is universal.

Clearly, G_2 is a disjoint union of n complete $r \times r$ bipartite graphs, i.e., $G_2 = M_1 + \cdots + M_n$, where $M_s = (\{p_{1,j,s}, p_{2,j,s} : j = 1, \ldots, r\}, \{(p_{1,j,s}, p_{2,j',s}) : j, j' = 1, \ldots, r\}), s = 1, \ldots, n$. Any 2-pin routing requirement R' on G_2 can be represented as a multiple bipartite graph over vertex set $\{i_1, \ldots, i_r\} \cup \{o_1, \ldots, o_r\}$ of degree at most n by converting 2-pin net $(a, \{a'\})$ to an edge (a, a'). Then by applying Hall's theorem [7], R' can be decomposed into at most n 1-factors (subgraph of maximum degree 1), $R' = F_1 + \cdots + F_n$. Since M_i is a complete $r \times r$ bipartite graph, F_i has a routing in $M_i, i = 1, \ldots, n$. This implies that G_2 is universal. Given a unicast call R on $c(n, n, r)$, R induces a 2-pin net routing

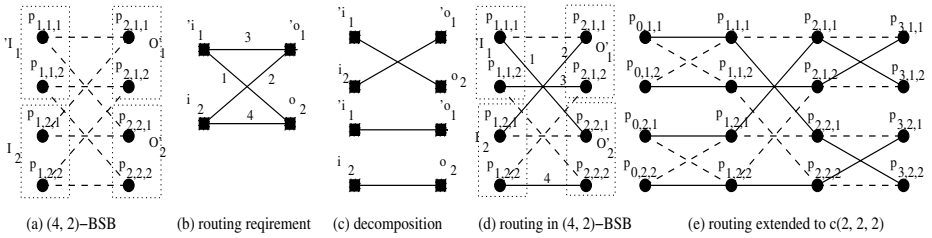


Fig. 4. Middle switch box and the routing of unicast call

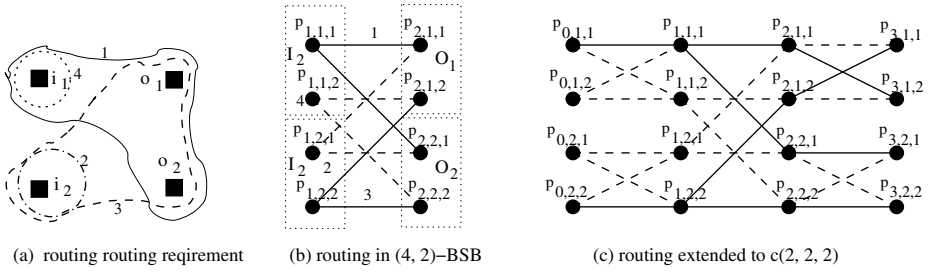


Fig. 5. Middle switch box and the routing of multicast call

requirement R' on G_2 by converting each request $(p_{0,j,s}, \{p_{3,j',s'}\})$ to a 2-pin net $(i_j, \{o_{j'}\})$. R' has a routing T in G_2 as G_2 is universal. The routing T on the middle switch box can then be extended to a routing of R on $c(n, n, r)$. Therefore, $c(n, n, r)$ is rearrangeable for unicast.

For example, Fig.4(a) shows the $(4, 2)$-BSB of $c(2, 2, 2)$. The unicast call $R = \{(p_{0,1,1}, \{p_{3,2,2}\}), (p_{0,1,2}, \{p_{3,1,1}\}), (p_{0,2,1}, \{p_{3,1,2}\}), (p_{0,2,2}, \{p_{3,2,1}\})\}$ induces a 2-pin routing requirement $R' = \{(i_1, \{o_2\}), (i_1, \{o_1\}), (i_2, \{o_1\}), (i_1, \{o_2\})\}$ shown in (b) and its decomposition in (c). Fig.4(d) gives the routing of R' in the $(4, 2)$-BSB, and (e) the routing of R on $c(2, 2, 2)$ extended from the routing R' on $(4, 2)$-BSB.

The routing of a multicast call can be done similarly in a Clos network. For example, the multicast call $R = \{(p_{0,1,1}, \{p_{3,1,2}, p_{3,2,1}\}), (p_{2,2,2}, \{p_{3,1,1}, p_{3,2,2}\})\}$ induces a multi-pin routing requirement $R' = \{(i_1, \{o_1, o_2\}), (i_2, \{o_1, o_2\})\}$, shown as a hypergraph in Fig.5(a). The routing of R' in the $(4, 2)$-BSB is shown Fig.5(b), and the extended routing on $c(2, 2, 2)$ is given in Fig.5(c).

It can be proved that the middle stage $(2r, n)$-BSB of the Clos network $c(n, n, r)$ is not hyperuniversal when $r \geq 4$. Our idea is to redesign the middle $(2r, n)$-BSB G_2 such that the resulting $(2r, n)$-BSB G_2' is hyperuniversal. Since any multicast call $R = \{(p_i, Q_i) : i = 1, \ldots, t\}$ on $G' = G_1 + G_2' + G_3$ induces a routing requirement $R' = \{(i_j, B_{i_j}) : j = 1, \ldots, t\}$ on G_2'. If G_2' is hyperuniveral, then R' has a routing $\{T_j : j = 1, \ldots, t\}$ on G_2'. Extend T_j to G_1 and G_2, a routing of R on $G' = G_1 + G_2' + G_3$ is obtained. Thus G' is rearrangeable for multicast if G_2 is hyperuniversal.

This observation leads us to consider the problem of designing an optimal (i.e., with the minimum number of edges) hyperuniversal $(2r, n)$-BSB. But it is difficulty to find an optimal hyperuniversal $(2r, n)$-BSB due to the difficulty in the verification of hyperuniversality. No polynomial algorithm time algorithm is known to find the routing of a given routing requirement in a BSB. Our approach is to find specially structured $(2r, n)$-BSBs, such that an efficient routing algorithm exits. The new hyperuniversal $(2r, n)$-BSB design we proposed is defined as follows.

Let $G_2 = (V_1 \cup V_2, E_2)$ be the middle stage of the Clos network $c(n, n, r)$. For any non-negative integer w, define

$$\begin{aligned} E_2(w) &= \{(p_{1,j,h}, p_{2,j',h'}) : |h - h'| \leq w\}, \\ G_2(w) &= G_2 + E_2(w), \\ c_w(n, n, r) &= G_1 + G_2(w) + G_3 = c(n, n, r) + E_2(w). \end{aligned} \tag{1}$$

Clearly, $E_2(0) = E_2$ and $c_0(n, n, r) = c(n, n, r)$. When $w > 0$, $G_2(w)$ is obtained by adding additional edges to G_2. In the next section, we will prove that $G_2(4r)$ is a hyperuniversal $(2r, n)$-BSB and therefore $c_{4r}(n, n, r)$ is rearrangeable for multicast.

3 Edge Ordering of Regular Hypergraphs

This section proves that $G_2(4r)$ defined in relation (1) is hyperuniveral. We use the property of edge ordering of regular hypergraphs and a job scheduling theorem.

Let $H = (V, E)$ be a multiple hypergraph, where V is the vertex set and E the edge set. E is a collection of subsets of V, namely, $e \subset V$ for every $e \in E$, and repetition of edges is allowed. The degree of a vertex u in H, denoted by $d_H(u)$, is the number of edges which contain (cover) u, i.e., $d_H(u) = |\{e \in E : u \in e\}|$. H is said to be w-regular if all its vertices have the same degree w. A regular hypergraph is said to be *minimal* if it does not contain a proper spanning (with the same vertex set) regular sub-hypergraph.

Given a regular hypergraph $H = (V, E)$ with $V = \{1, \ldots, k\}$ and $E = \{e_1, \ldots, e_m\}$. Let π be a permutation on $\{1, \ldots, m\}$, then $(e_{\pi(1)}, \ldots, e_{\pi(m)})$ is an ordering of the edges, and it determines a sequence of $m + 1$ sub-hypergraphs

$$H_j = (\{1, \ldots, k\}, \{e_{\pi(1)}, \ldots, e_{\pi(j)}\}), j = 0, \ldots, m,$$

or $H_0 = (\{1, \ldots, k\}, \emptyset), H_{j+1} = H_j + e_{\pi(j+1)}, j = 1, \ldots, m$. We define

$$g(H, \pi) = \max\{\max\{|d_{H_j}(u) - d_{H_j}(v)| : u, v \in e_{\pi(j+1)}\} : j = 0, \ldots, m-1\} \tag{2}$$

$$g(H) = \min\{g(H, \pi) : \text{over all permumations } \pi\} \tag{3}$$

For any integer $k \geq 2$, we define

$$g(k) = \max\{g(H) : \text{over all regular hypergraphs } H \text{ on } k \text{ vertices}\}. \tag{4}$$

First, we see that $g(k)$ is well-defined. Let H be any regular hypergraph on k vertices, then H can be decomposed into a union of edge disjoint minimal spanning regular hypergraphs $R_1, \ldots, R_t$, $H = R_1 + \cdots + R_t$, where a regular hypergraph is minimal if it can not be decomposed further. Since we can obtain an ordering of edges of H by ordering edges of $R_1, \ldots, R_t$ respectively and then put them together one following another, we have the following relations $g(H) \leq \max\{g(R_1), \ldots, g(R_t)\}$,

$$g(H) \leq \max\{g(R) : \text{over all minimal regular hypergraphs } R \text{ on } k \text{ vertices}\},$$

$$g(k) \leq \max\{g(R) : \text{over all minimal regular hypergraphs } R \text{ on } k \text{ vertices}\}.$$

On the other hand, since a minimal regular hypergraph on k vertices is also a regular hypergraph, by (4) we have

$$g(k) \geq \max\{g(R) : \text{over all minimal regular } s\text{-hypergraphs } R \text{ on } k \text{ vertices}\}$$

$$g(k) = \max\{g(R) : \text{over all minimal regular hypergraphs } R \text{ on } k \text{ vertices}\} \quad (5)$$

Since there are a finite number of minimal regular hypergraphs on k vertices [6], $g(k)$ is well-defined by (5).

Edge Ordering Problem of Regular Hypergraphs: Given an integer $k \geq 2$, determine the value of $g(k)$. Given a regular hypergraph H on k vertices, find an edge permutation π such that $g(H, \pi) \leq g(k)$.

The above edge ordering problem has not been solved. No value of $g(k)$ is known for $k \geq 7$. However, we prove an upper bound for $g(k)$ using the following the vector sum theorem in job scheduling theory [11,1].

Lemma 3.1 ([11]). *Let $\mathbf{v_1}, \ldots, \mathbf{v_t}$ be a sequence of k-dimensional vectors with $\sum_{i=1}^{t} \mathbf{v_i} = 0, ||\mathbf{v_i}|| \leq 1 (i = 1, \ldots, t)$, there is a permutation $i_1, \ldots, i_t$, such that $\max || \sum_{h=1}^{j} \mathbf{v_{i_h}} || \leq k$ for every $1 \leq j \leq t$, where the super norm $||\mathbf{v}||$ of a vector $\mathbf{v}$ is defined to be the maximum of the absolute values of the components of $\mathbf{v}$.*

Theorem 3.2. *$g(k) \leq 2k$ for any $k \geq 2$.*

Proof. Let $H = (V, E)$ be any w-regular hypergraph on k vertices. Suppose that $V = \{1, \ldots, k\}$ and $E = \{e_1, \ldots, e_m\}$. Let $\mathbf{v_i} = (n_{i,1}, \ldots, n_{i,k}) \in R^k$ be the vector representation of $e_i, i = 1, \ldots, m$.

For $i = 1, \ldots, m$, let $\mathbf{v'_i} = \mathbf{v_i}$, and for $i = m+1, \ldots, m+w$, let $\mathbf{v'_i} = (-1, \ldots, -1) \in R^k$. Then $\sum_{i=1}^{m+w} \mathbf{v'_i} = 0$. By Lemma 3.1, there is a permutation $i'_1, \ldots, i'_{m+w}$ of $1, \ldots, m+w$ such that $\mathbf{v'_{i'_1}}, \ldots, \mathbf{v'_{i'_{m+w}}}$ satisfy $|| \sum_{h=1}^{j} \mathbf{v'_{i'_h}} || \leq k, 1 \leq s \leq m+w$.

Removing the vectors equal to $(-1, \ldots, -1)$ from the sequence $\mathbf{v'_{i'_1}}, \ldots, \mathbf{v'_{i'_{m+w}}}$, we obtain a permutation $\mathbf{v_{i_1}}, \ldots, \mathbf{v_{i_m}}$ of $\mathbf{v_1}, \ldots, \mathbf{v_m}$.

For any $1 \leq j \leq m-1$, let $\mathbf{v'_{i'_{j'}}}$ correspond to $\mathbf{v_{i_j}}$. Then there are $j' - j$ vectors in $\mathbf{v'_{i'_1}}, \ldots, \mathbf{v'_{i'_{j'}}}$ which are equal to $(-1, \ldots, -1)$. Then, for every $1 \leq j \leq m-1$, we have

$$
\begin{aligned}
&\max\{|d_{H_j}(u) - d_{H_j}(v)| : u, v \in e_{i_{j+1}}\} \\
&= \max\{d_{H_j}(u) : u \in e_{i_{j+1}}\} - \min\{d_{H_j}(v) : v \in e_{i_{j+1}}\} \\
&= \max\{\textstyle\sum_{h=1}^{j} n_{i,h} : i \in e_{i_{j+1}}\} - \min\{\sum_{h=1}^{j} n_{i,h} : i \in e_{i_{j+1}}\} \\
&\le \max\{\textstyle\sum_{h=1}^{j} n_{i,h} : 1 \le i \le k\} - \min\{\sum_{h=1}^{j} n_{i,h} : 1 \le i \le k\} \\
&= \max\{\textstyle\sum_{h=1}^{j} n_{i,h} : 1 \le i \le k\} + (j' - j) - (\min\{\sum_{h=1}^{j} n_{i,h} : 1 \le i \le k\} + (j' - j)) \\
&= \max\{\textstyle\sum_{h=1}^{j} n_{i,h} + (j - j') : 1 \le i \le k\} - \min\{\sum_{h=1}^{j} n_{i,h} + (j' - j) : 1 \le i \le k\} \\
&\le 2\| \textstyle\sum_{h=1}^{j'} \mathbf{v}'_{\mathbf{i_h}} \| \le 2k.
\end{aligned}
$$

By the definitions (1)-(3) for $g(k)$, we have $g(k) \le 2k$. □

In [1], a polynomial time algorithm was given to find a permutation of the vectors with $\max \| \sum_{h=1}^{j} \mathbf{v}_{\mathbf{i_h}} \| \le 3k/2$ for every $1 \le j \le t$ in time $O(t^2k^3 + tk^4)$. Applying that algorithm, we can find an edge ordering π for regular hypergraph H in polynomial time with $g(H, \pi) \le 3k$ in time $O(m^2k^3 + mk^4)$.

Next we use the unicast call on $(2r, n)$-BSB G_2 of $c(n, n, r)$ to illustrate edge the ordering based routing algorithm. Let H be an n-regular bipartite graph obtained from a 2-pin net routing requirement R' on $G_2 = M_1 + \cdots + M_n$. By Hall's theorem, H can be decomposed into n 1-factors, $H = F_1 + \cdots + F_n$. Let π be the edge ordering that orders the edges in F_1, followed by an edge ordering of F_2, and so on, until an ordering of F_n. Then we have $g(H, \pi) = 0$. Next we find the routing of R' by the edge ordering as follows. At step $i - 1$, we have partial hypergraph H_{i-1}, let (a_i, b_i) be the next edge and assume that $d_{H_{i-1}}(a) = d_{H_{i-1}}(b) = j$, then route (a_i, b_i) in M_{j+1} and let $H_i = H_{i-1} + (a_i, b_i)$. By induction we know this routing algorithm will end with a routing of R' in G_2. For example, Fig.4(b) shows an edge ordering π_1 of regular graph $H_1 = (\{i_1, i_2\} \cup \{o_1, o_2\}, \{\{i_1, o_2\}, \{i_2, o_1\}, \{i_1, o_1\}, \{i_2, o_2\}\}$ labeled $1, 2, 3, 4$. We have $g(H_1, \pi_1) = 0$. Fig.4 (d) shows the routing of the routing requirements following the edge ordering of H_1.

The edge ordering routing method can be applied to multicast. Fig.5(a) gives an edge ordering π_2 of the 2-regular hypergraph $H_2 = (\{i_1, i_2\} \cup \{o_1, o_2\}$, $\{\{i_1, o_1, o_2\}, \{i_2\}, (i_2, o_1, o_2\}, \{i_1\}\}$ labeled $1, 2, 3, 4$. Then $g(H_2, \pi_2) = 0$, Fig.5(b) shows the routing of the routing requirements according to this edge ordering π_2. Applying the edge ordering method, we have the following theorem.

Theorem 3.3. *If $w \ge g(2r)$, then $G_2(w)$ is hyperuniversal $(2r, n)$-BSB with at most $(2w + 1)r^2n$ switches.*

Proof. Let $R' = \{(a_i, b_i) : i = 1, \ldots, t\}$ be a routing requirement on $(2r, n)$-BSB $G_2(w)$. Then R' corresponds to a hypergraph $H = (\{i_0, \ldots, i_r, o_1, \ldots, o_r\}, N)$ of degree at most n, where $N = \{\{a_i\} \cup b_i : i = 1, \ldots, t\}$. If H is not n-regular, then add singletons to H to obtain an n-regular hypergraph, still denoted by H.

Suppose that $H = (\{i_1, \ldots, i_r\} \cup \{o_1, \ldots, o_r\}, \{e_1, \ldots, e_m\})$. By the definition of $g(2r)$ in relation (3), there is an edge ordering π of H such that $g(H, \pi) \le g(2r) \le w$. Then we find a routing of R' in $G_2(w)$ according to the edge ordering $(e_{\pi(1)}, \ldots, e_{\pi(m)})$ as follows. For $i = 1$, suppose that $e_{\pi(1)}$ corresponds to multi-pin net $(a_{\pi(1)}, b_{\pi(1)})$ and $b_{\pi(1)} = \{b_{1,1}, \ldots, b_{1,k_1}\})$, then

route $e_{\pi(1)}$ by edges $(p_{1,a_{\pi(1)},1}, p_{2,b_{1,1},1}), \ldots, (p_{1,a_{\pi(1)},1}, p_{2,b_{1,k_1},1})$. Assume that $e_{\pi(i-1)}$ has been routed. Next we route $e_{\pi(i)}$. Suppose that $e_{\pi(i)} = (a_{\pi(i)}, b_{\pi(i)})$ and $b_{\pi(i)} = \{b_{i,1}, \ldots, b_{i,k_i}\}$). Then route $e_{\pi(i)}$ by edges $(p_{1,a_{\pi(i)},c}, p_{2,b_{i,1},c_1}), \ldots, (p_{1,a_{\pi(i)},c}, p_{2,b_{i,k_i},c_{k_i}})$, where c is the smallest index such that $p_{1,a_{\pi(i)},c}$ is not used by previous nets in $I_{a_{\pi(i)}}$, and c_j is the smallest index such that $p_{1,b_{i,j},c_j}$ is not used by previous nets in $O_{b_{i,j}}$. By the definition of $G_2(w)$, the edge set of $G_2(w)$ is $E_2(w) = \{(p_{1,j,h}, p_{2,j',h'}) : |h - h'| \leq w\}$. Since $g(H, \pi) \leq g(2r) \leq w$, we have $|c - c_j| \leq w, j = 1, \ldots, k_i$, therefore, $(p_{1,a_{\pi(i)},c}, p_{2,b_{i,1},c_1}), \ldots, (p_{1,a_{\pi(i)},c}, p_{2,b_{i,k_i},c_{k_i}}) \in E_2(s)$, the routing for $e_{\pi(i)}$ is feasible. Continue this process, we obtain the routing of all nets $e_{\pi(1)}, \ldots, e_{\pi(m)}$.

Since the degree of each vertex of V_1 of $G_2(w)$ is at most $(2w+1)r$ and it has rn vertices, therefore $G_2(w)$ has at most $(2w+1)r^2n$ edges. □

Theorem 3.4. *$G_2(4r)$ is a hyperuniveral $(2r, n)$-BSB and $c_{4r}(n, n, r)$ is rearrangeable for multicast and has $2nr$ intermediate nodes and at most $(8r+1)r^2n + 2rn^2$ switches.*

Proof. By Theorem 3.2, we have $g(2r) \leq 4r$, and by Theorem 3.3 we have that $G_2(4r)$ is a hyperuniversal $(2r, n)$-BSB. Therefore, the 3-stage switching network $c'_{2r}(n, n, r) = G_1 + G_2(4r) + G_3$ is rearrangeable for multicast, and it has $2nr$ intermediate nodes and at most $(8r + 1)r^2n + 2rn^2$ switches. □

When $rn = N$, choose $n = N^{2/3}, r = N^{1/3}$. Then by Theorem 3.4, we know that $c_{4N^{1/3}}(N^{2/3}, N^{2/3}, N^{1/3})$ is rearrangeable for multicast and it has $2N$ intermediate nodes and $(8N^{1/3} + 1)N^{1/3}N^{2/3} + 2N^{1/3}(N^{2/3})^2 = O(N^{5/3})$ switches. Moreover, if we choose $w = 6r$, then $c_{6N^{1/3}}(N^{2/3}, N^{2/3}, N^{1/3})$ is rearrangeable for multicast and has $2N$ intermediate nodes, $O(N^{5/3})$ switches, and a routing algorithm in time $O(m^2N + mN^{1/3})$, where m is the number of connection requests.

Remarks

1. Even though $c_{4N^{1/3}}(N^{2/3}, N^{2/3}, N^{1/3})$ provides a better solution in terms of intermediate nodes and switches, it is not practical for a large N. In a practical implementation, we choose $c_w(N^{1/2}, N^{1/2}, N^{1/2})$ for a small value of $w = 0, 1, 2, 3$. Such a network will have $O(N^{3/2})$ switches though the mulicast rearrangeability is not guaranteed when N is large. We also apply heuristic routing algorithm for such switching networks. One open question is that what is the lower bound of w such that $c_w(n, n, r)$ is rearrangeable for multicast.

2. The fanout of the above routing algorithm is at stage two and three. If we also do the fanout at the first stage, then the routing capacity can be further increased with the current design.

3. On the other hand, the routing capacity can be improved by increasing the number of intermediate nodes with $c_w(m, n, r) = c(m, n, r) + E_2(w), m > n$. However, there is no result on a better bound for w and m for rearrangeable multcast networks.

4. If we substitute each middle $r \times r$ crossbar of $c(N^{2/5}, N^{2/5}, N^{3/5})$ by the new $c_{4N^{3/10}}(N^{3/10}, N^{3/10}, N^{3/10})$, we obtain a 5-stage Clos network with $O(N^{7/5})$ switches and the minimum $4N$ intermediate nodes.

4 Conclusions

We presented a new rearrangeable multicast 3-stage switching network design. As a tradeoff between the number of switches and intermediate nodes, the newly designed $N \times N$ network uses $O(N^{5/3})$ switches and the minimum $2N$ intermediate nodes. The new design has a simple structure, obtained by adding extra switches to a unicast Clos network. The new network also accommodate an efficient routing algorithm. The design and analysis was done by a combinatorial approach, which makes use of existing graph theory, scheduling results and algorithms. The combinatorial approach reveals a new investigation direction to explore the tradeoff among the number of switches, the number of intermediate nodes, delays, routing capacity, and routing algorithms for efficient implementations of on-chip switching networks.

References

1. Barany, I.: A Vector-Sum Theorem and Its Application to Improving Flow Guarantees. Mathematics of Operations Research 6, 445–455 (1981)
2. Chang, Y.W., Wong, D.F., Wong, C.K.: Universal Switch Modules for FPGA Design. ACM Transactions on Design Automation of Electronic Systems 1(1), 80–101 (1996)
3. Clos, C.: A Study of Nonblocking Switching Networks. Bell Systems Technical J. 22, 406–424 (1953)
4. Dolev, D., Dwork, C., Pippenger, N., Wigderson, A.: Superconcentrators, Generalizers and Generalized Connectors with Limited Depth. In: STOC, pp. 42–51 (1983)
5. Fan, H., Liu, J., Wu, Y.L.: General Models and a Reduction Design Technique for FPGA Switch Box Designs. IEEE Transactions on Computers 52(1), 21–30 (2003)
6. Fan, H., Wu, Y.L., Cheung, C., Liu, J.: Decomposition Design Theory and Methodology for Arbitrary-Shaped Switch Boxes. IEEE Transactions on Computers 55(4), 373–384 (2006)
7. Hall, P.: On Representatives of Subsets. J. London Math. Soc. 10, 26–30 (1935)
8. Hawang, F.K.: The Mathematical Theory of Nonblocking Switching Networks. World Scientific Publishing Co., Inc., River Edge (2004)
9. Hwang, F.K.: Rearrangeahility of Multiconnection Three-Stage Networks. Networks 2, 301–306 (1972)
10. Masson, G.M., Jordan, B.W.: Generalized Multi-Stage Connection Networks. Networks 2, 191–209 (1972)
11. Sevast'yanov, S.V.: On Approximate Solutions of Scheduling Problems. Metody Discretnogo Analiza 32, 66–75 (1978)
12. Yang, Y., Masson, G.M.: Nonblocking Broadcast Switching Networks. IEEE Trans. Comput. 40(9), 1005–1015 (1991)

Bicriteria Scheduling on Single-Machine with Inventory Operations*

Baoqiang Fan[1], Rongjun Chen[2], and Guochun Tang[3]

[1] Department of Mathematics and Information, Ludong University
Yantai 264025, China
baoqiangfan@163.com
[2] Department of Mathematics, Changzhou Institute of Technology
Changzhou 213002, China
chenrjecust@163.com
[3] Institute of Management Engineering, Shanghai Second Polytechnic University
Shanghai 201209, China
gtang@sh163.net

Abstract. In this paper, we consider the single machine scheduling problem with inventory operations. The objective is to minimize makespan subject to the constraint that the total number of tardy jobs is minimum. We show the problem is strongly NP-hard. A polynomial $(1+1/(m-1))$-approximation scheme for the problem is presented, where m is defined as the total job's processing times $\sum p_j$ divided by the capacity c of the storage, and an optimal algorithm for a special case of the problem, in which each job is one unit in size, is provided.

Keywords: scheduling, bicriteria, NP-hardness, approximation algorithm, performance ratio.

1 Introduction

We consider a maker-to-order production-distribution system consisting of one supplier and more customers. At the beginning of a planning horizon, each customer places a order with the supplier. The supplier needs to process these orders and deliver the completed orders to the customers. Each order has a due date specified by the customer and is required to be delivered by its due date. However, that all orders would just be completed at their respective due dates by the supplier is great difficulty. Some orders have to be scheduled to complete ahead of their due dates so that the supplier can deliver orders on time as much as possible.

The problem is often faced by the manufactures who make time-sensitive products such as perishable food, which must be stored in the special storage for those jobs(products) completed ahead of their due dates. Another factor the manufacturer has to consider is that the capacity of the storage is limited. The

* Supported by the National Science Foundation of China (No. 70731160015).

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 395–402, 2009.

total size of the jobs stored should not be more than such capacity in any time. The problem we study in this paper is to find a schedule for the jobs so that some objectives are optimized. For example, to minimize the total number of tardy jobs or makespan. To be able to refer to the problems under study in a concise manner, we shall use the notation of Graham et al. [9], extended to job field with inventory operations. The problem of scheduling jobs on single machine with inventory operations is represented by $1|inven|\gamma$, where $inven$ stands for the jobs with inventory operations.

As for bicriterion scheduling problems, two different criteria are considered together. This can be accomplished in a number of ways. One approach is to minimize the less important criterion, subject to the restriction that the most important criterion is optimized. The two criteria are assumed to be prioritized as primary and secondary and the objective is to find the best schedule for the secondary criterion among all alternative optimal schedules for the primary criterion. This problem is denoted by $1||Lex(\gamma_1, \gamma_2)$, where γ_1 is the primary criterion and γ_2 is the secondary criterion.

In this paper, considering a single machine scheduling problem, where n jobs $\{J_1, J_2, \cdots, J_n\}$ are ready for processing at time zero. Each job J_j has a processing time p_j, a size v_j and a due date d_j. If π is a schedule of the n jobs, we let C_j denote the completion time of job J_j in π. If $C_j < d_j$, the job needs to be stored until its due date. If $C_j \geq d_j$, the job would be delivered immediately by its completion time. We are given a storage with capacity c meaning that the total size of inventory is up to c at any time. The objective is to minimize makespan subject to minimize the total number of tardy jobs $\sum U_j$, where U_j is $0-1$ indicator variable that takes the value 1 if J_j is tardy, i.e., if $C_j > d_j$, and the value 0 if J_j is on time, i.e., $C_j \leq d_j$. We represent the two dual criteria scheduling problems by $1|inven|Lex(\sum U_j, C_{\max})$.

When the capacity of the storage is unlimited or $\sum_{j=1}^{n} v_j \leq c$, our problem becomes a normal bicriterion scheduling problem $1||Lex(\sum U_j, C_{\max})$, which is equal to scheduling problem $1||\sum U_j$. For the problem, a schedule with minimum number of tardy jobs can be obtained by the Hodgson-Moore algorithm [15], which schedules jobs in ascending order of due dates. Most of the work reported about multicriterion scheduling has been concerned with bicriterion scheduling problems on a single machine. Smith [17] may be the first researcher to study bicriterion scheduling problem on a single machine. He developed a polynomial-time algorithm to minimize the total completion time, subject to the constraint that no job is late. In 1975, Emmons [7] studied the problem $1||Lex(\sum U_j, \sum C_j)$. He proposed a branch-and-bound algorithm which in the worst case runs in exponential time. Later, Chen and Bulfin [4] proved that the problem is NP-hard with respect to id-encoding. Vairaktarakis and Lee [19] studied the problem $1||Lex(\sum U_j, \sum T_j)$. They gave a polynomial-time algorithm when the set of tardy jobs is specified. As well, a branch-and-bound algorithm was given for the general problem. In 2007, Huo et al. [12] proved the problem is binary NP-hard. Cheng [5] developed a solution methodology for minimizing the flow time and missed due dates. Surveys on algorithms and complexity results of bicriterion

scheduling problems have been given by Chen and Bulfin [3], Nagar et al. [16] and Hoogeveen [11].

If we consider the inventory cost instead of capacity constrain, and the tardiness penalty instead of $\sum U_j$, the problem $1|inven|Lex(\sum U_j, C_{\max})$ becomes a normal JIT (just-in-time) scheduling problem $1||\sum(E_j + T_j)$. Hall and Posner [10] showed the problem is NP-hard. When all jobs have a common due date $d_j \equiv d$, they provided an $O(n\sum p_j)$ pseudo-polynomial time algorithm. For such problem, Bagchi et al. [1] proved that the number of optimal schedules is $2^{\lfloor \frac{n}{2} \rfloor}$. Further results about single machine JIT scheduling problems can be found in [2,6,13,14].

In Section 2, we prove that the problem $1|inven|Lex(\sum U_j, C_{\max})$ is strongly NP-hard and a polynomial approximation scheme for the problem is presented. A special case of the problem, in which each job is one unit in size, is provided an optimal algorithm. Section 3 is a brief conclusion.

2 To Minimize $Lex(\sum U_j, C_{\max})$

Without loss of generality, we assume that $v_j \leq c, p_j \leq d_j (j = 1, 2, \cdots, n)$ and $\sum_{j=1}^{n} p_j > c$. In many applications, job has a larger size if its processing time is larger. In the following, we consider the case that $v_j = p_j (j = 1, 2, \cdots, n)$. Firstly, we show that the problem of minimizing makespan on single machine with inventory operations is strongly NP-hard.

2.1 The Proof of the NP-Hardness

The problem $1|inven|Lex(\sum U_j, C_{\max})$ is strongly NP-hard. This is done by reducing the strongly NP-hard 3-Partition[8] to the decision version of our problem.

3-Partition. Given positive integers t, A and a set of integers $S = \{a_1, a_2, \cdots, a_{3t}\}$ with $\sum_{j=1}^{3t} a_j = tA$ and $A/4 < a_j < A/2$ for $1 \leq j \leq 3t$, does there exit a partition $\langle S_1, S_2, \cdots, S_t \rangle$ of S into 3-element sets such that

$$\sum_{a_j \in S_i} a_j = A$$

for each i?

Theorem 1. *The problem* $1|inven|Lex(\sum U_j, C_{\max})$ *is strongly NP-hard.*

Proof. Given the 3-partition problem t, A and a set of integers $\{a_1, a_2, \cdots, a_{3t}\}$. We will first describe the decision version I of the problem $1|inven|Lex(\sum U_j = 0, C_{\max})$.

There are basically two classes of jobs in I. The first class, $\{J_j^1 | 1 \leq j \leq t\}$, where job lengths and due date times are specified as follows:

$$\begin{cases} p_j^1 = tA + 1, & j = 1, 2, \cdots, t; \\ d_j^1 = j(t+1)A + j, j = 1, 2, \cdots, t. \end{cases}$$

The second class, $\{J_j^2 | 1 \leq j \leq 3t\}$, with job lengths and due dates specified as follows:

$$\begin{cases} p_j^2 = a_j, & j = 1, 2, \cdots, 3t; \\ d_j^2 = t^2A + tA + t, j = 1, 2, \cdots, 3t. \end{cases}$$

The job sizes $v_j = p_j, j = 1, 2, \cdots, 4t$. We define the capacity of the storage is c, where $c = tA$. The bound is given by $\delta = t^2A + tA + t$. All the remains is to show that the desired partition of S exists if and only if a schedule for I exists, which length less than or equal to δ and all jobs are on time.

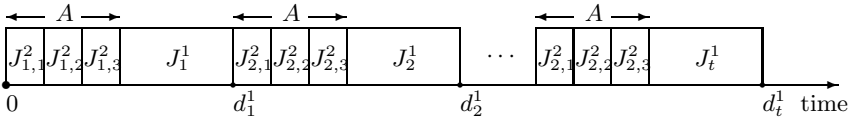


Fig. 1. Illustration of the scheduling π

Firstly, suppose a partition $\langle S_1, S_2, \cdots, S_t \rangle$ exists which has the desired form. That is each set S_i consists of three elements a_{i1}, a_{i2} and a_{i3}, such that for all $1 \leq i \leq t, \sum_{j=1}^{3} a_{ij} = A$. Then the following schedule π has length $\delta = t^2A + tA + t$. About the first class jobs in such schedule, the job J_j^1 is processed with the completed time

$$C(J_j^1) = d_j^1 = j(t+1)A + j, j = 1, 2, \cdots, t.$$

From Fig.1, we note that this basic framework just leaves a series of t "*time slots*" open before the time $t^2A + tA + t$, each of which length exactly A, and the due date of the second class jobs is $t^2A + tA + t$. These are precisely tailored so that we can fit the second class jobs as follows. For each $i = 1, 2, \cdots, t$,

$$\begin{aligned} s(J_{i1}^2) &= d_{i-1}^1 \\ s(J_{i2}^2) &= d_{i-1}^1 + a_{i1} \\ s(J_{i3}^2) &= d_{i-1}^1 + a_{i1} + a_{i2} \end{aligned}$$

Since $\sum_{j=1}^{3} a_{ij} = A (1 \leq i \leq t)$ and the total size of jobs in storage is less than the capacity c in any time, this yields a valid schedule with $C_{\max}(\pi) = \delta$ and no tardy job.

Conversely, suppose a schedule π with $C_{\max}(\pi) = \delta$ does exist. Because the total length of jobs in I is $\sum p_j = t^2A + tA + t$, we must have $C_{\max}(\pi) = \delta = t^2A + tA + t$, and the machine is no idle in π. Because of no tardy job in π, from the constructor of the jobs and the capacity $c = tA$, the first class jobs must be scheduled as the same way they are in Fig.1. Thus there are again t slots of length A into which the second class jobs can be placed.

Since the total length of the second class jobs is $\sum_{j=1}^{3t} p_j^2 = tA$, every one of these t slots must be filled completely, and hence must contain a set of the second class jobs whose total length is exactly A. Now since every $a_j > A/4$, no

such set can contain more than three jobs. Similarly, since $a_j < A/2$, no such set can contain less than three jobs. Thus each set contains exactly three jobs of the second class. Hence, by setting $S_i = \{a_i | d^1_{i-1} < S_{J_i^2} < d_i - p^1_i\}, i = 1, 2, \cdots, t$, we obtain our desired partition. □

2.2 Approximating Optimal Makespan in Polynomial Time

Since the problem $1|inven|Lex(\sum U_j = 0, C_{\max})$ is strongly NP-hard, we design an approximation algorithm for the problem. Let $\sum_{j=1}^{n} p_j = mc$, where m is defined as the total job's processing times $\sum p_j$ divided by the capacity c of the storage. From $\sum_{j=1}^{n} p_j > c$, $m > 1$. Firstly, we introduce some useful properties associated with optimal schedules as follows.

Lemma 1. *For the problem* $1|inven|Lex(\sum U_j = 0, C_{\max})$, *if*

$$\begin{cases} \sum_{j=k}^{n} p_j > c \\ \sum_{j=k+1}^{n} p_j \leq c \end{cases} \tag{1}$$

then

$$C^*_{\max} \geq d_k,$$

where $k \in \{1, 2, \cdots, n-1\}$ *and* $C^*_{\max}$ *is the value of the optimal schedule.*

Proof. If every job is completed before the time d_k, there are at less $n - k$ stocking jobs such as $\{J_k, J_{k+1}, \cdots, J_n\}$ at the time d_k. Since $\sum_{j=k}^{n} p_j > c$, it follows that there must exist some jobs to be processed after the time d_k, or the job J_k to be completed at the time d_k. Thus the optimal value of the problem is $C^*_{\max} \geq d_k$. □

If the problem exists a feasible schedule, there exists a partly feasible schedule about the jobs $\{J_1, J_2, \cdots, J_j\}$ before the time d_j, $j = 1, 2, \cdots, n$. However, tardy job is possible if the jobs $\{J_{k+1}, \cdots, J_n\}$ begin to process at time d_k. To avoid the tardy job, we need to look for an more useful boundary $d_r (k + 1 \leq r < n)$. From the Lemma 1, there is a conclusion as follows.

Lemma 2. *For the problem* $1|inven|Lex(\sum U_j = 0, C_{\max})$, *let*

$$d_k + \sum_{i=k+1}^{r} p_i - d_r = \max\{d_k + \sum_{i=k+1}^{j} p_i - d_j, j = k+1, \cdots, n\}.$$

If

$$d_k + \sum_{i=k+1}^{r} p_i - d_r > 0,$$

then there exists an optimal schedule π *with jobs* $\{J_{r+1}, \cdots, J_n\}$ *begin to process at time* d_r *and*

$$C_{\max}(\pi) \leq d_k + \sum_{j=k+1}^{n} p_j. \tag{2}$$

Where $k+1 \leq r \leq n$ *and* k *subject to the Lemma 1.*

Based on the Lemma 1 and Lemma 2, we now provide an approximating algorithm for the problem $1|inven|Lex(\sum U_j = 0, C_{\max})$.

Heuristic Alg.1

Step 1. To search for the $k \in \{1, 2, \cdots, n\}$ s.t. $\sum_{j=k}^{n} p_j > c$ and $\sum_{j=k+1}^{n} p_j \leq c$.

Step 2. Let $d_k + \sum_{i=k+1}^{r} p_i - d_r = \max\{d_k + \sum_{i=k+1}^{j} p_i - d_j, j = k+1, \cdots, n\}$. If $d_k + \sum_{i=k+1}^{r} p_i - d_r > 0$, then $k \doteq r$.

Step 3. Let $C_k = d_k, C_j = \min\{d_j, C_{j+1} - p_{j+1}\}, j = k-1, \cdots, 1$, and $s_j = C_{j-1}, j = k+1, \cdots, n$.

By (1) and $\sum_{j=1}^{n} p_j = mc$, we have

$$d_k \geq \sum_{j=1}^{k} p_j > (m-1)c,$$

then

$$\sum_{j=k+1}^{n} p_j < \frac{1}{m-1} d_k.$$

This, together with (2), implies that the makespan of the schedule π found by the Alg.1 satisfies

$$C_{\max}(\pi) \leq \left(1 + \frac{1}{m-1}\right) d_k.$$

From the above discussions, we have the following theorem.

Theorem 2. *For* $1|inven|Lex(\sum U_j = 0, C_{\max})$, *the* Alg.1 *has a worst-case competitive ratio of* $1 + \frac{1}{m-1} (m > 1)$, *and the time complexity of the* Alg.1 *is* $O(n^2 logn)$.

2.3 Each Job Is One Unit in Size

Consider the following inventory operations. Each job is one unit in size, $v_j \equiv 1 (j = 1, 2, \cdots, n)$. We are given a storage of capacity c. Thus the storage can store up to c jobs in any time.

For the remainder of this section, we suppose that all jobs have be indexed by EDD rule, $d_1 \leq d_2 \leq \cdots \leq d_n$. We first provide a optimality property for the problem.

Lemma 3. *For the problem* $1|v_j \equiv 1, inven|Lex(\sum U_j = 0, C_{\max})$*, there exists an optimal schedule, in such schedule the jobs are processed by the EDD rule.*

Proof. Suppose π is an optimal schedule, in which the jobs are not processed by the EDD rule. Since all jobs are completed on time in π and have the same size, the result can be established by a standard job interchange argument about the schedule π. □

For any schedule π of $\{J_1, J_2, \cdots, J_n\}$, in which the jobs are processed by the EDD rule, let $I_{d_k} = \{J_i | C_i < d_k, i \geq k\}$ denote those jobs which are stored at the time d_j. If the number of the jobs stored is not more than c, such that $\|I_{d_k}\| \leq c (j = 1, 2, \cdots, n)$, the schedule π is feasible.

Algorithm Qusia-EDD

Step 1. Set $d_0 = 0, C_0 = 0$ and $C_1 = p_1$.
Step 2. For $j = 1, 2, \cdots, n$.
Let $d_{k-1} \leq C_{j-1} + p_j < d_k, k \leq j$,
and compute $I_{d_k} = \{J_i | C_i < d_k, i \geq k\}$.
If $\|I_{d_j}\| \leq c$, set $C_j = C_{j-1} + p_j$.
Else, set $C_j = d_k$.
Set $j = j + 1$.

Given Lemma 3, the optimality of this algorithm can be easily proved. Hence we state the following result without proof.

Theorem 3. *Algorithm Qusia-EDD can find an optimal schedule for the problem* $1|v_j \equiv 1, inven|Lex(\sum U_j = 0, C_{\max})$ *in* $O(n^2)$ *time.*

3 Concluding Remarks

In this paper, we address the problem $1|inven|Lex(\sum U_j, C_{\max})$ and give the proof of strongly NP-hard. A polynomial time 2-approximation scheme for the problem is presented and an optimal algorithm for a special case of the problem. We will go on researching this problem with other objective (i.e. $Lex(\sum U_j, \sum C_j)$, $Lex(\sum U_j, T_{\max})$ and $Lex(T_{\max}, \sum C_j)$). Another research topic is about the complexity of the open problem $1||Lex(\sum U_j, C_{\max})$.

References

1. Bagchi, U., Sullivan, R.S., Zhang, Y.-L.: Minimizing Absolute and Squared Deviation of Completion Times about a Common Due Date. Nav. Res. Log. 33, 227–240 (1986)
2. Baker, K.R., Scudder, G.D.: Sequencing with Earliness and Tardiness Penalties: A review. Oper. Res. 38, 22–36 (1990)
3. Chen, C.L., Bulfin, R.L.: Scheduling Unit Processing Time Jobs on a Single Machine with Multiple Criteria. Comp. Oper. Res. 17, 1–7 (1990)

4. Chen, C.L., Bulfin, R.L.: Complexity of Single Machine Multicriteria Scheduling Problems. Eur. J. Oper. Res. 70, 115–125 (1993)
5. Cheng, T.C.E.: Minimizing the Flow Time and Missed Due Dates in a Single Machine Sequencing. Math. Comp. Model. 13, 71–77 (1990)
6. Davis, J.S., Kanet, J.J.: Single-Machine Scheduling with Early and Tardy Completion Costs. Nav. Res. Log. 40, 85–101 (1993)
7. Emmons, H.: One Machine Sequencing to Minimize Mean Flow Time with Minimum Number Tardy. Nav. Res. Log. 22, 585–592 (1975)
8. Garey, M.R., Johnson, D.S.: Computers and Intactability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
9. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and Approximation in Deterministic Sequencing and Scheduling. Ann. Discrete Math. 5, 287–326 (1979)
10. Hall, N.G., Posner, M.E.: Earliness-Tardiness Scheduling Problems, I. Weighted Deviation of Completion Times about a Common Due Date. Oper. Res. 39, 836–846 (1991)
11. Hoogeveen, H.: Multicriteria Scheduling. Eur. J. Oper. Res. 167, 592–623 (2005)
12. Huo, Y.-M., Leung, J.Y.-T., Zhao, H.-R.: Complexity of Two Dual Criteria Scheduling Problems. Oper. Res. Lett. 35, 211–220 (2007)
13. Kim, Y.-D., Yano, C.A.: Minimizing Mean Tardiness and Earliness in Single-Machine Scheduling Problems with Unequal Due Dates. Nav. Res. Log. 41, 913–933 (1994)
14. Li, C.-L., Chen, Z.-L., Cheng, T.C.E.: A Note on One Processor Scheduling with Asymmetric Earliness and Tardiness Penalties. Oper. Res. Lett. 13, 45–48 (1993)
15. Moore, J.M.: An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs. Management Sci. 15, 102–109 (1968)
16. Nagar, A., Haddock, J., Heragu, S.: Multiple and Bicriteria Scheduling: a Literature Survey. Eur. J. Oper. Res. 81, 88–104 (1995)
17. Smith, W.E.: Various Optimizers for Single Stage Production. Nav. Res. Log. 3, 59–66 (1956)
18. T'kindt, V., Billaut, J.-C.: Multicriteria Scheduling: Theory, Models and Algorithms. Springer, Berlin (2002)
19. Vairaktarakis, G.L., Lee, C.-Y.: The Single-Machine Scheduling Problem to Minimize Total Tardiness Subject to Minimum Number of Tardy Jobs. IIE Trans. 27, 250–256 (1995)

Approximation Algorithm for Minimizing the Weighted Number of Tardy Jobs on a Batch Machine*

Jianfeng Ren[1,**], Yuzhong Zhang[1], Xianzhao Zhang[1], and Guo Sun[2]

[1] College of Operations Research and Management Sciences, Qufu Normal University
Shangdong 276826, China
qrjianfeng@163.com
[2] Department of Mathematics, Qufu Normal University
Shangdong 276826, China

Abstract. We consider the problem of minimizing the weighted number of tardy jobs ($\sum_{j=1}^{n} w_j U_j$) on an unbounded batch processing machine. The batch processing machine can process up to B ($B \geq n$) jobs simultaneously. The jobs that are processed together form a batch, and all jobs in a batch start and complete at the same time. For a batch of jobs, the processing time of the batch is equal to the largest processing time among the jobs in this batch. In this paper, we design a fully polynomial time approximation scheme (FPTAS) to solve the unbounded batch scheduling problem $1|B \geq n|\sum_{j=1}^{n} w_j U_j$. This is the strongest possible polynomial time approximation result that we can derive for an NP-hard problem (unless $P = NP$ holds).

Keywords: Approximation algorithms, Unbounded batch machine, Dynamic programming.

1 Introduction

An unbounded batch machine or batch processing machine is a machine that can process up to B ($B \geq n$) jobs simultaneously. The jobs that are processed together form a batch. Specifically, we are interested in the so-called burn-in model, in which the processing time of a batch is equal to the maximum processing time of any job assigned to it. All jobs contained in the same batch start and complete at the same time, since the completion time of a job is equal to the completion time of the batch to which it belongs. This model is motivated by the problem of scheduling burn-in operations for large-scale integrated circuit (IC) chips manufacturing (see Lee [7] for the detailed process). In this paper, we study the problem of scheduling n independent jobs on an unbounded machine

* Supported by the National Natural Science Foundation (Grant Number 10671108) and the Natural Science Foundation of Shandong Province (Grant Number Y2005A04).

** Corresponding author.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 403–410, 2009.

to minimize the weighted number of tardy jobs. Denote by C_j the completion of job J_j in some scheduling.

- if $C_j \leq d_j$, then job J_j is early and $U_j = 0$.
- if $d_j < C_j$, then job J_j is tardy and $U_j = 1$.

Using the notation of Graham et al [4], we denote this problem as $1|B \geq n|\sum_{j=1}^{n} w_j U_j$. Brucker [2] showed that the problem $1|B \geq n|\sum_{j=1}^{n} w_j U_j$ is NP-hard in the ordinary sense.

Previous related work: Brucker [2] showed that $1|B \geq n|\sum_{j=1}^{n} w_j U_j$ is NP-hard in the ordinary sense and proved that it is maximal pseudopolynomially solvable. For the problem of $1|p_j = p, r_j|\sum_{j=1}^{n} w_j U_j$, Baptiste [1] has shown that it is maximal polynomially. Lawler [5] proved that the problem $1|r_i, pmtn|\sum_{j=1}^{n} w_j U_j$ is maximal pseudopolynomially solvable. Lawer [6] and Karp [9] respectively proved that the problem $1||\sum_{j=1}^{n} w_j U_j$ is maximal pseudopolynomially solvable. It is shown in [8] that $1|B \geq n|\sum_{j=1}^{n} T_j$ is binary NP-hard. Also, they establish the pesudopolynomial solvability of the unbounded batch machine scheduling problem with job release dates and any regular objective. Concerning $1|B \geq n, r_j|\sum_{j=1}^{n} w_j C_j$, Deng and Zhang [3] establish its NP-hardness and present polynomial algorithms for several special cases.

Our contributions: In this paper, we design a fully polynomial time approximation scheme (FPTAS) to solve the unbounded batch scheduling problem $1|B \geq n|\sum_{j=1}^{n} w_j U_j$.

2 Problem Description, Notation, and Elementary Definitions

The scheduling model that we study is as following. There are n independent jobs $J_1, J_2, \cdots, J_n$ that have to be scheduled on an unbounded batch machine. Each job $J_j (j = 1, 2, \cdots, n)$ has a processing time p_j, a positive weight w_j, and a due date d_j. All jobs are available for processing at time 0. The goal is to scheduling the jobs without preemption on the unbounded batch machine such that the weighted number of tardy jobs is minimized.

The set of real numbers is denoted by IR, and the set of non-negative integers is denoted by IN; note that $0 \in IN$. The base two logarithm of z denoted by $\log z$, and the natural logarithm by $\ln z$.

We recall the following well-known properties of binary relations $\preceq$ on a set Z. The relation $\preceq$ is called

- *reflexive* if for any $z \in Z$: $z \preceq z$,
- *symmetric* if for any $z, z' \in Z$: $z \preceq z'$ implies $z' \preceq z$,
- *anti-symmetric* if for any $z, z' \in Z$: $z \preceq z'$ and $z' \preceq z$ implies $z' = z$,
- *transitive* if for any $z, z', z'' \in Z$: $z \preceq z'$ and $z' \preceq z''$ implies $z \preceq z''$.

A relation on z is called a *partial order*, if it is reflexive, anti-symmetric, and transitive. A relation on Z is called a *quasi-order*, if it is reflexive and transitive.

A quasi-order on Z is called a quasi-linear order, if any two elements of Z are comparable.

For $Z' \subseteq Z$, an element $z \in Z'$ is called a *maximum* in $Z^{'}$ with respect to $\preceq$, if $z' \preceq z$ holds for all $z' \in Z'$. The element $z \in Z'$ is called a *maximal* in $Z^{'}$ with respect to $\preceq$, if the only $z' \in Z'$ with $z \preceq z'$ is z itself.

Proposition [**6**]. *For any binary relation $\preceq$ on a set Z, and for any finite subsetZ' of Z the following holds.*

(i) *If $\preceq$ is a partial order, then there exists a maximal element in Z.*
(ii) *If $\preceq$ is a quasi-line order, then there exists at least one a maximum element in Z.*

Woeginger [10] showed that dynamic programming algorithms with a special structure automatically lead to a *fully polynomial time approximation scheme*. Assume that we have an approximation algorithm that always returns a near-optimal solution whose cost is at most a factor of ρ away from the optimal cost, where $\rho > 1$ is some real number: In minimization problems the near-optimal is at most a multiplicative factor of ρ above the optimum, and in maximization problems it is at most a factor of ρ below the optimum. Such an approximation algorithm is called a ρ-approximation algorithm. A family of $(1+\varepsilon)$-approximation algorithms over all $\varepsilon > 0$ with polynomial running times is called a polynomial time approximation scheme or PTAS, for short. If the time complexity of a PTAS is also polynomially bounded in $(1/\varepsilon)$, then it is called a fully polynomial time approximation scheme or, FPTAS, for short. An FPTAS is the strongest possible polynomial time approximation result that we can derive for an NP-hard problem (unless, of course, P=NP holds). Woeginger et al. [10] considered a GENEric optimization problem GENE and provided a natural and uniform approach to design the fully polynomial time approximation scheme for it.

A DP-simple optimization problem GENE is called DP-benevolent iff there exist a partial order $\preceq_{dom}$, a quasi-line order $\preceq_{qua}$, and a degree-vector D such that its dynamic programming formulation DP fulfills the Conditions C.1-C.4. The DP-benevolent problem is easy to approximate.

Lemma 1. [10] *If an optimization problem GENE is DP-benevolent, then the dynamic program TDP is an FPTAS for it.*

3 The Dynamic Programming and Algorithm MTDP

As we can firstly schedule all jobs whose processing times are zero, delete all jobs with due dates being zero and let all non-zero processing times and due date be integers by multiplying the same suitable positive number and keep the same structure of optimal schedule, in the following we assume that all p_j $(j = 1, 2, \cdots, n)$ and d_j $(j = 1, 2, \cdots, n)$ are non-negative integers. We renumber the jobs such that $d_1 \leq d_2 \leq \cdots \leq d_n$. A straightforward job interchange argument can prove the following lemma.

Lemma 2. *Under this numbering, there exists an optimal schedule in which all early jobs are processed before all tardy jobs and all early jobs are processed in increasing order of indices. Moreover, an optimal schedule will not contain any idle time.*

3.1 The Dynamic Programming

Now let $\alpha = 3$ and $\beta = 4$. For $k = 1, 2, \cdots, n$ we define the input vector $X_k = [p_k, w_k, d_k]$. A state $S = [s_1, s_2, s_3, s_4] \in S_k$ encodes a partial schedule for the first jobs $J_1, J_2, \cdots, J_k$: the coordinate s_1 measures the total processing time of the scheduled early jobs in the partial schedule, and s_2 measures the total weight of the tardy jobs which is scheduled on the machine in the partial schedule. s_3 stores the due date of the job with lowest index in the latest batch which is scheduled early on the machine in the partial schedule. s_4 stores the largest processing time in the latest batch which are scheduled early on the machine in the partial schedule. Set $\mathcal{F}$ consists of three functions F_1, F_2, F_3.

$$F_1[p_k, w_k, d_k, s_1, s_2, s_3, s_4] = [s_1 + \max\{0, p_k - s_4\}, s_2, s_3, \max\{s_4, p_k\}].$$

$$F_2[p_k, w_k, d_k, s_1, s_2, s_3, s_4] = [s_1 + p_k, s_2, d_k, p_k].$$

$$F_3[p_k, w_k, d_k, s_1, s_2, s_3, s_4] = [s_1, s_2 + w_k, s_3, s_4].$$

Intuitively speaking, the function F_1 adds the job J_k to the last batch of the partial schedule $S = [s_1, s_2, s_3, s_4] \in S_k$ for the jobs $J_1, J_2, \cdots, J_k$ and schedule it early, i.e. adds job J_k to the l-th machine so that it does not start the last batch.

The function F_2 adds the job J_k to the end of the partial schedule $S = [s_1, s_2, s_3, s_4] \in S_k$ for the jobs $J_1, J_2, \cdots, J_k$ and schedules it early. i.e. adds job J_k so that it starts the last batch. The function F_3 schedules the job J_k tardy. The functions H_1, H_2 and H_3 in $\mathcal{H}$ correspond to F_1, F_2 and F_3 respectively.

$$H_1[p_k, w_k, d_k, s_1, s_2, s_3, s_4] = s_1 + \max\{0, p_k - s_4\} - s_3$$

$$H_2[p_k, w_k, d_k, s_1, s_2, s_3, s_4] = s_1 + p_k - d_k$$

$$H_3[p_k, w_k, d_k, s_1, s_2, s_3, s_4] \equiv 0$$

Now the iterative computation in Line 5 of DP for all functions in $\mathcal{F}$ reads

If $s_1 + \max\{0, p_k - s_4\} - s_3 \leq 0$	then add $[s_1 + \max\{0, p_k - s_4\}, s_2, s_3, \max\{s_4, p_k\}]$
If $s_1 + p_k - d_k \leq 0$	then add $[s_1 + p_k, s_2, d_k, p_k]$
If $0 \leq 0$	then add $[s_1, s_2 + w_k, s_3, s_4]$

Finally, set $G[s_1, s_2, s_3, s_4] = s_2$ and initialize the space $S_0 = \{[0, 0, 0, 0]\}$.

Next we proof the problem $1|B \geq n| \sum_{j=1}^{n} w_j U_j$ is benevolent.

Let the degree vector $D = [1, 1, 0, 0]$. Note that the coordinates according to the state variables s_1, s_2 are 1 and the other two coordinates are 0. Let

$$S = [s_1, s_2, s_3, s_4] \in S_k, S' = [s'_1, s'_2, s'_3, s'_4] \in S_k.$$

The dominance relation is defined as follows:

$$S \preceq_{dom} S' \Leftrightarrow s'_1 \leq s_1, \quad s'_2 \leq s_2, \qquad s'_h = s_h \qquad h = 3, 4.$$

The quasi-linear order is defined:

$$S \preceq_{qua} S' \Leftrightarrow s'_1 \leq s_1.$$

Theorem 1. *For any $\Delta > 1$, for any $F \in \mathcal{F}$, for any $S, S' \in IN^4$, the following holds:*

(i) *If S is $[D, \Delta]$-close to S' and if $S \preceq_{qua} S'$, then (a) $F(X, S) \preceq_{qua} F(X, S')$ holds and $F(X, S)$ is $[D, \Delta]$-close to $F(X, S')$, or (b) $F(X, S) \preceq_{dom} F(X, S')$.*
(ii) *If $S \preceq_{dom} S'$, then $F(X, S) \preceq_{dom} F(X, S')$, where $X = [p_k, w_k, d_k]$, $k = 1, 2, \cdots, n$.*

Proof. (i) Consider a real number $\Delta > 1$, two vectors $S = [s_1, s_2, s_3, s_4]$ and $S' = [s'_1, s'_2, s'_3, s'_4]$ that fulfill S is $[D, \Delta]$-close to S' and $S \preceq_{qua} S'$. From S is $[D, \Delta]$-close to S', we get that

$$\Delta^{-1} s_1 \leq s'_1 \leq \Delta s_1 \tag{1}$$

$$\Delta^{-1} s_2 \leq s'_2 \leq \Delta s_2 \tag{2}$$

$$s'_h = s_h \qquad \text{for } h = 3, 4. \tag{3}$$

As $S \preceq_{qua} S'$, so

$$s'_1 \leq s_1 \tag{4}$$

From (3) and (4) we have

$$s'_1 + \max\{0, p_k - s'_4\} \leq s_1 + \max\{0, p_k - s_4\}, \text{ and } s'_1 + p_k \leq s_1 + p_k \tag{5}$$

(5) and (4) yield that

$$F_1(X, S) \preceq_{qua} F_1(X, S'),$$
$$F_2(X, S) \preceq_{qua} F_2(X, S'),$$
$$F_3(X, S) \preceq_{qua} F_3(X, S').$$

From (1-4) and $\Delta > 1$ we have

$$\begin{aligned} \Delta^{-1}(s_1 + \max\{0, p_k - s_4\}) &\leq s'_1 + \Delta^{-1} \max\{0, p_k - s_4\} \\ &= s'_1 + \Delta^{-1} \max\{0, p_k - s'_4\} \\ &\leq s_1 + \max\{0, p_k - s'_4\} \\ &\leq \Delta(s_1 + \max\{0, p_k - s_4\}). \end{aligned} \tag{6}$$

$$\Delta^{-1}(s_1 + p_k) \leq s_1' + \Delta^{-1}p_k \leq s_1 + p_k \leq \Delta(s_1 + p_k) \tag{7}$$

$$\Delta^{-1}(s_2 + w_k) \leq s_2' + \Delta^{-1}w_k \leq s_2 + w_k \leq \Delta(s_2 + w_k) \tag{8}$$

$$\max\{s_4', p_k\} = \max\{s_4, p_k\} \tag{9}$$

(1-3) and (6-9) imply that $F_1(X, S)$ is $[D, \Delta]$-close to $F_1(X, S')$, $F_2(X, S)$ is $[D, \Delta]$-close to $F_2(X, S')$ and $F_3(X, S)$ is $[D, \Delta]$-close to $F_3(X, S')$. Hence, for functions F_1, F_2 and F_3, Theorem 1(i) holds.
(ii) As $S \preceq_{dom} S'$, we have

$$s_1' \leq s_1, \quad s_2' \leq s_2, \quad s_h' = s_h \qquad for \quad h = 3, 4. \tag{10}$$

$$s_1' + \max\{0, p_k - s_4'\} \leq s_1 + \max\{0, p_k - s_4\}, \quad s_2' \leq s_2, \quad s_3' = s_3 \tag{11}$$

and

$$\max\{s_4', p_k\} = \max\{s_4, p_k\} \tag{12}$$

Then (10), (11) and (12) imply that $F_1(X, S) \preceq_{dom} F_1(X, S')$, and that F_1 satisfies Theorem 1(ii).

Similarly, we can verify that $F_2(X, S) \preceq_{dom} F_2(X, S')$ and $F_3(X, S) \preceq_{dom} F_3(X, S')$. □

Theorem 2. *For any $\Delta > 1$, for any $H \in \mathcal{H}$, for any $S, S' \in IN^4$, the following holds:*

(i) *If S is $[D, \Delta]$-close to S' and $S \preceq_{qua} S'$, then $H(X, S') \leq H(X, S)$.*
(ii) *If $S \preceq_{dom} S'$, then $H(X, S') \leq H(X, S)$.*

Proof. (i) By S is $[D, \Delta]$-close to S' and $S \preceq_{qua} S'$, from (3) and (4) we can easily get

$$s_1' + \max\{0, p_k - s_4'\} - s_3' \leq s_1 + \max\{0, p_k - s_4\} - s_3 \tag{13}$$

$$s_1' + p_k - d_k \leq s_1 + p_k - d_k \tag{14}$$

Then (12) and (13) yield that $H_1(X, S) \leq H_1(X, S')$ and $H_2(X, S) \leq H_2(X, S')$. As $H_3(X, S) \equiv 0$, so $H_3(X, S) = H_3(X, S')$ holds.
(ii) From $S \preceq_{dom} S'$ applying (10), we also can easily get

$H_1(X, S) \leq H_1(X, S')$, $H_2(X, S) \leq H_2(X, S')$, and $H_3(X, S) = H_3(X, S')$. □

Theorem 3. *Let $g = 1$, then for any $\Delta > 1$, and for any $S, S' \in IN^4$, the following holds:*

(i) *If S is $[D, \Delta]$-close to S' and if $S \preceq_{qua} S'$, then $G(S') \leq \Delta^g G(S) = \Delta G(S)$.*
(ii) *If $S \preceq_{dom} S'$, then $G(S') \leq G(S)$.*

Proof. (i) from S is $[D, \Delta]$-close to S' and $S \preceq_{qua} S'$, applying the definitions of $[D, \Delta]$-close and The quasi-order relation we get

$$\Delta^{-1}s_2 \leq s_2' \leq \Delta s_2 \tag{15}$$

As $G[s_1, s_2, s_3, s_4] = s_2$, Applying (15), we get $G(S') \leq \Delta G(S)$.
(ii) From $S \preceq_{dom} S'$, we have $s_2' \leq s_2$ Then holds $G(S') \leq \Delta G(S)$. □

Theorem 4. (i) *Every $F \in \mathcal{F}$ can be evaluated in polynomial time. Every $H \in \mathcal{H}$ can be evaluated in polynomial time. The function G can be evaluated in polynomial time. The relation $\preceq_{qua}$ can be decided in polynomial time.*

(ii) *The cardinality of $\mathcal{F}$ is polynomially bounded in n and $\log \overline{x}$.*

(iii) *For every instance I of $1|B \geq n|\sum_{j=1}^{n} w_j U_j$, the state space S_0 can be computed in time that is polynomially bounded in n and $\log \overline{x}$. As a consequence, also the cardinality of the state space S_0 is polynomially bounded in n and $\log \overline{x}$.*

(iv) *For an instance I of $1|B \geq n|\sum_{j=1}^{n} w_j U_j$, and for a coordinate l $(1 \leq l \leq 4)$, let $V_l(I)$ denote the set of the l-th components of all vectors in all state spaces S_k $(1 \leq k \leq n)$. Then the following holds for every instance I.*

For all coordinate l $(1 \leq l \leq 4)$, the natural logarithm of every value in $V_l(I)$ is bounded by a polynomial $\pi_1(n, \log \overline{x})$ in n and $\log \overline{x}$. Moreover, for coordinate l with $d_l = 0$, the cardinality of $V_l(I)$ is bounded by a polynomial $\pi_2(n, \log \overline{x})$ in n and $\log \overline{x}$.

Proof. (i-iii) are straightforward. For (iv), note that the coordinates with $d_l = 0$ only take the job processing times or job due dates these elements. Hence, (iv) is also true. □

3.2 The Algorithm MTDP

Based on the above dynamic programming we give following algorithm.

Algorithm MTDP

Step 0 Delete all jobs with zero processing times and jobs with due dates are zero, moreover change all other processing times into integers by multiplying the same positive number. We denote the new instance I'. For I' go to **Step 1**
Step 1 Initialize $\mathcal{T}_0 := S_0$
Step 2 For $k = 1$ to n do
Step 3 Let $\mathcal{U}_k := \phi$
Step 4 For every $T \in \mathcal{T}_{k-1}$ and every $F \in \mathcal{F}$ do
Step 5 If $H_F(X_k, w_k, d_k, T) \leq 0$ then add $F(X_k, w_k, d_k, T)$ to $\mathcal{U}_k$
Step 6 End-for
Step 7 Compute a trimmed copy $\mathcal{T}_k$ of $\mathcal{U}_k$
Step 8 End-for
Step 9 Output min $\{G(S) : S \in S_n\}$
Step 10 Schedule the jobs of instance I according to I' and insert sufficient zero batches scheduling jobs (deleted in **Step 0**) with zero processing times in the ahead of partial scheduling.

Theorems 1-4, which just express that the above dynamic programming fulfills the Conditions C.1-4, show that the problem $1|B \geq n|\sum_{j=1}^{n} w_j U_j$ is DP-benevolent. Applying Lemma 1 we can get the following result.

Theorem 5. *There is an FPTAS for* $1|B \geq n|\sum_{j=1}^{n} w_j U_j$ *that constructs a* $(1+\varepsilon)$*-approximation in time* $O\lceil n^2(1+2gn/\varepsilon)\rceil^2$*, where* $\overline{x} = \prod_{j=1}^{n} p_j w_j d_j$*.*

Proof. From Lemma 1 we get that MTDP is an FPTAS for problem $1|B \geq n|\sum_{j=1}^{n} w_j U_j$. Note, $|\mathcal{F}|$=4, $\mathcal{T}_k \leq (L+1) \times (L+1) \times n \times n = (n(L+1))^2 = \lceil n(1+2gn/\varepsilon)\rceil^2$ and the running time of deciding the relation $\preceq_{qua}$ on $\mathcal{T}_k$ is $O(n^2)$. So the total running time of Algorithm MTDP is $O\lceil n^2(1+2gn/\varepsilon)\rceil^2$, where $\Delta = 1 + 2gn/\varepsilon$ is the trimming parameter. □

References

1. Baptiste, P.: Polynomial Time Algorithms for Minimizing the Weighted Number of Late Jobs on a Single Machine with Equal Processing Times. Journal of Scheduling 2, 245–252 (1999)
2. Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M.Y., Potts, C.N., Tautenhahn, T., Van De Velde, S.: Scheduling a Batching Machine. Journal of Scheduling 1, 31–54 (1998)
3. Deng, X., Zhang, Y.: Minizing Mean Response Time in Batch Processing System. In: Asano, T., Imai, H., Lee, D.T., Nakano, S.-i., Tokuyama, T. (eds.) COCOON 1999. LNCS, vol. 1627, pp. 231–240. Springer, Heidelberg (1999)
4. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. Ann Disc Math 5, 287–326 (1979)
5. Lawler, E.L.: A Dynamic Programming Algorithm for Preemptive Scheduling of a Single Machine To Minimize the Number of Late Jobs. Annals of Operation Research 26, 125–133 (1990)
6. Lawler, E.L., Moore, J.M.: A Functional Equation and Its Application to Resource Allocation and Sequencing Problems. Management Science 16, 77–84 (1996)
7. Lee, C.Y., Uzsoy, R., Martin Vega, L.A.: Efficient Algorithms for Scheduling Semiconductor Burn-in Operations. Operation Research 40, 764–775 (1992)
8. Liu, Z., Yuan, J., Edwin Cheng, T.C.: On Schedulimg an Unbounded Batch Machine. Operations Research Letters 31, 42–48 (2003)
9. Karp, R.M.: Reducibility Among Combinatorial Problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1992)
10. Woeginger, G.J.: When Does a Dynamic Programming Formulation Guarantee the Existence of an FPTAS. Technical Report Woe-27. Tu-Graz, Austria (1998)

Scheduling with Rejection to Minimize the Makespan*

Yuzhong Zhang, Jianfeng Ren, and Chengfei Wang

Institute of Operations Research, Qufu Normal University
Rizhao 276826, China
yuzhongrz@163.com

Abstract. In this paper, we consider the scheduling with rejection. The objective functions are to minimize the maximum completion time of the processed ones when the total compression cost is given. Firstly, we prove that the problem $1|rej|C_{\max}/TCP$ is NP-hard, which implying that $P_m|rej|C_{\max}/TCP$, $1|rej, r_j|C_{\max}/TCP$, $1|rej, on-line|C_{\max}/TCP$ are all NP-hard. Secondly, for problem $P_m|rej|C_{\max}/TCP$, we design a pseudopolynomial time dynamic programming algorithm that solves it exactly and an $FPTAS$ (full polynomial time approximation scheme) when m is a constant. We also design a pseudopolynomial time dynamic programming algorithm and an $FPTAS$ for the case of non-identical job arrival problem $1|rej, r_j|C_{\max}/TCP$. In the end, we consider the on-line problem $1|rej, on-line|C_{\max}/TCP$ and prove that there doesn't exist any on-line algorithm with a constant competitive ratio for it, even if the jobs only have two different release times.

Keywords: Approximation algorithm, scheduling with rejection, dynamic programming, worst case performance ratio.

1 Introduction

In classical scheduling models, it is usually assumed that jobs must be processed and that the processing times are given unchanged. While in many applications, a job could not be processed if it's processing time or processing cost is very large. We could choose to pay some charge to send it to outside for "out processing", or buy finished product. That is, it needs a decision to choose some jobs processed, and some jobs rejected (namely process with rejection). Here, the scheduling problem we considered are not only to minimize the regular objective functions (such as maximum completion time or weighted completion time, maximum lateness etc), but also make the total cost of rejected job's be in an acceptable interval. This is the so-called problem of scheduling with rejection. There is not many results researched in this field. In 1996, Bartal et al. [2] consider the problem $P_m|rej|C_{\max}+TCP$, which is the first paper to study the problem

* Supported by the National Natural Science Foundation(Grant Number 10671108) and the Province Natural Science Foundation of Shandong (Grant Number $Y2005A04$).

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 411–420, 2009.

of scheduling with rejection. For off-line model, they give an $FPTAS$ and a $2-1/m$ approximation algorithm with running time $O(nlogn)$, and for on-line model, they give a $(3+\sqrt{5})/2$ approximation algorithm which is the best possible result for it. He and Min [7] provide the best competitive ratio algorithm for the special cases when the number of uniform machines is two or three. In 1998, Engels et al. [4] study $1|rej|\sum w_jC_j + TP$ and prove it's $NP-hard$ and also provide pseudo-polynomial time algorithm and an $FPTAS$ for it. Epstein et al. [5] consider the problem $1|on-line, rej, p_j \equiv 1|\sum C_j + TP$ and give a greedy algorithm with competitive ratio $(2+\sqrt{3})/2$ and a lower bound of 1.633 for it. In 2003, Sengupta et al. [9] study the scheduling problem with rejection which objective function is to minimize the sum of the maximum lateness and the total rejection cost and design their $PTAS$. For unrelated case and when the objective function is to minimize the sum of maximum completion time and the total rejection cost, Seiden et al. [8] prove that the problem is APX-hard and give a 1.58-approximation algorithm for it. While for identical or uniform model, they prove that the problem is NP-complete and design an $FPTAS$ for it. They also point out that the results on unrelated machines can be extended to multi-stage scheduling of free operation with preemption. It has been recently proved by Cao et al. [3] that the problem $1|rej, r_j|C_{\max} + \sum_{j\in R} e_j$ is NP-hard, even jobs only have two different arrival times. They also design a $PTAS$ for off-line case and give a $(\sqrt{5}+1)/2$ -competitive algorithm for on-line case when jobs have two different arrival times, which is the best possible result for it.

Next, we give some notations concerning in this paper. Let $J = \{1, 2, \cdots, n\}$ denote the set of jobs, where job j processing time is $p_j \geq 0$, arrival time is $r_j \geq 0$, and rejection cost is $e_j \geq 0$. For job j, if we accept to process it, then job j's processing time is p_j, and completion time is denoted as C_j. While if we reject (to process) it, then we should pay rejection cost e_j. Given a scheduling π, let $S(\pi) \subseteq J$ be the set of jobs accepted, then $R(\pi) = J/S(\pi)$ is the set of jobs rejected. Let H is any a given bound of total rejection cost, our object is to find a scheduling π such that $TCP = \sum_{j\in R(\pi)} e_j \leq H$ and minimize $C_{\max}(\pi) = \sum_{j\in S(\pi)} p_j$. Using the 3−field notation of Graham et al [6], we denote our problems as:

(1) $1|rej|C_{\max}/TCP$;
(2) $P_m|rej|C_{\max}/TCP$;
(3) $1|rej, r_j|C_{\max}/TCP$;
(4) $1|rej, on-line|C_{\max}/TCP$.

Where rej implies that the job can be rejected; P_m denotes m (where m is a constant) identical machines.

2 The NP-Hardness

We prove it's NP-hardness by reduction from the 0/1 **knapsack problem:** There is a set of n positive integers (v_j, w_j) and a integer B, given any threshold L, does there exist an index set $K \subseteq \{1, 2, \cdots, n\}$ such that $\sum_{j\in K} v_j \geq L$ and $\sum_{j\in K} w_j \leq B$?

Theorem 1. $1|rej|C_{\max}/TPC$ *is NP-hard.*

Proof. Given any instance $I = \{(B, v_j, w_j) : 1 \le j \le n\}$ of knapsack problem, we define an instance I' of the scheduling problem $1|rej|C_{\max}/TPC$ with n jobs $J_1, J_2, \cdots, J_n$, where $p_j = v_j$, $e_j = w_j$ $(j = 1, 2, \cdots, n)$ and $H = B$.

Obviously, the construction of I' can be done in polynomial time of the input size. Let $S(\pi)$ and $R(\pi)$ be the set of jobs accepted and rejected by the schedule π, respectively. Given any threshold L, we only need to prove the following two conditions are equivalent:

(a) There exists a schedule π of I' such that

$$C_{\max}(\pi) = \sum_{j \in S(\pi)} p_j \le L \quad \text{and} \quad \sum_{j \in R(\pi)} e_j \le H;$$

(b) There exists a set K of I' such that

$$\sum_{j \in K} v_j \ge \sum_{j=1}^{n} p_j - L \quad \text{and} \quad \sum_{j \in K} w_j \le B.$$

$(a) \Rightarrow (b)$ Suppose π is a feasible schedule of I' such that

$$C_{\max}(\pi) = \sum_{j \in S(\pi)} p_j \le L \quad \text{and} \quad \sum_{j \in R(\pi)} e_j \le H.$$

As $p_j = v_j$, $e_j = w_j$ and $\sum_{j=1}^{n} p_j = \sum_{j \in S(\pi)} p_j + \sum_{j \in R(\pi)} p_j$. Set $K = R(\pi)$, then

$$\sum_{j \in K} v_j \ge \sum_{j=1}^{n} p_j - L \quad \text{and} \quad \sum_{j \in K} w_j \le B.$$

$(b) \Rightarrow (a)$ can be proved similarly. □

The following three lemmas can be proved easily from the Theorem 1.

Lemma 1. $P_m|rej|C_{\max}/TPC$ *is NP-hard.*

Lemma 2. $1|rej, r_j|C_{\max}/TPC$ *is NP-hard.*

Lemma 3. $1|rej, on-line|C_{\max}/TPC$ *is NP-hard.*

3 Dynamic Programming Algorithm and an *FPTAS* for the Problem $P_m|rej|C_{\max}/TPC$

3.1 Dynamic Programming Algorithm for $P_m|rej|C_{\max}/TCP$

Let $\{J_j = (p_j, e_j) : 1 \le j \le n\}$ be the set of jobs, m be the number of identical parallel machines $\{M_1, M_2, \cdots, M_m\}$, H be the upper bound about the total

penalties (TPC) of rejected jobs, where m is a constant. We will get a schedule which minimizing the makespan and having TPC at most H by a dynamic programming algorithm. For any partial schedule for jobs $\{J_1, J_2, \cdots, J_j\}$, we say that its state is $C = (c_1, c_2, \cdots, c_m)$, where c_i is the finish time of machine M_i. If (j, C) can be obtained by some partial schedule, we say it's feasible. Let S_j denote the jth state space, i.e., the set of all the feasible states obtained by partial schedules for jobs $\{J_1, J_2, \cdots, J_j\}$. For any $(j, C) \in S_j$, using $\Gamma(j, C)$ represents the minimum TPC of partial schedules whose states are (j, C).

Algorithm A

Step 1 Set $S_1 := \{(1, 0), (1, p_1 E_v)\}$ $(v = 1, 2, \cdots, m)$, $S_j := \phi$ $(j = 2, \cdots, n)$, $\Gamma(1, 0) := e_1$ and $\Gamma(1, p_1 E_v) := 0$.
Step 2 For $j = 2$ to n; if $\Gamma(j-1, C) + e_j > H$, set $S_j := \{(j, C + p_j E_l)\}$; if $\Gamma(j-1, C) + e_j \leq H$, then set $S_j := \{(j, C), (j, C + p_j E_l)\}$. Calculate the corresponding $\Gamma(j, C)$.
Step 3 Output the state vector $(n, C) \in S_n$ that minimizes the makespan $C_{\max}$.
Step 4 Give the corresponding schedule of the original instance according to (n, C).

As every coordinate of $C = (c_1, c_2, \cdots, c_m)$ comes from the interval $[0, P_{sum}]$ where $P_{sum} = \sum_{j=1}^{n} p_j$, the cardinality of any S_k is upper bounded by $O(P_{sum}^m)$. So the running time is

$$O\big[n(p_{sum})^m\big].$$

Algorithm A is a pseudo-polynomial algorithm. As the cardinality of T_j is at most

$$\Big(\lceil log_{1+\varepsilon_0}^{np_{\max}} \rceil\Big)^m,$$

and the total cycle of the algorithm is n, so the total running time is

$$O\Big[n\big(\lceil log_{1+\varepsilon_0}^{np_{\max}} \rceil\big)^m\Big].$$

3.2 Algorithm MA and $FPTAS$ for $P_m|rej|C_{\max}/TCP$

In order to get an $FPTAS$, we have to trim the state space S_j into a new one whose cardinality is polynomial in the input size and the trimming cost should be small enough. Using T_j denotes the trimmed state space of S_j. Given any accuracy parameter $\varepsilon > 0$, let $\varepsilon_0 = \varepsilon/(2n)$. We divide the time interval $[0, p_{sum}]$ into t intervals $I_0 = [0, 1]$, $I_1 = ((1+\varepsilon_0)^0, (1+\varepsilon_0)^1]$, $I_2 = ((1+\varepsilon_0)^1, (1+\varepsilon_0)^2]$,$\cdots$, $I_t = ((1+\varepsilon_0)^{t-1}, (1+\varepsilon_0)^t]$, where $t = \lceil log_{1+\varepsilon_0}^{p_{sum}} \rceil$.

Algorithm MA

Step 1 Trim S_1 into T_1 according to the method of the following **Step 2**.

Step 2 For $j = 1$ to n, calculate S_j from T_{j-1} using the **Algorithm A**, and trim S_j to T_j according to the following rules:
For any $(j, C) \in S_j$, if c_t $(1 \le t \le m)$ falls into the time interval I_i $(i \ge 1)$, set $c'_t = (1+\varepsilon_0)^i$; if $c_t = 0$ or $c_t = 1$, then set $c'_t = c_t$.
Nextly, we set $T_j = \{(j, C') | (j, C) \in S_j\}$ and $\Gamma(j, C') = \Gamma(j, C)$, where $C' = (c'_1, c'_2, \cdots, c'_m)$.

Step 3 Output the state vector (n, C^*) from S_n such that $M(n, C') \le H$ and minimizing the makespan $C_{\max}$.

Step 4 Give the corresponding schedule of the original instance according to (n, C^*).

Theorem 2. *Algorithm MA yields a $(1+\varepsilon)$-factor approximation algorithm (FPTAS) for $P_m|rej|C_{\max}/TPC$, which runs in time $O[n(1+2n/\varepsilon)^m|I|^m]$. Where $|I|$ is the input size of I under the binary encoding.*

Proof. Time complexity: As the cardinality of T_j is at most $(\lceil log_{1+\varepsilon_0}^{p_{sum}} \rceil)^m$, and the total cycle of the algorithm is n, so the total running time is $O[n(\lceil log_{1+\varepsilon_0}^{p_{sum}} \rceil)^m]$. Note that

$$|I| \ge log_2^{p_{sum}} = \frac{ln(p_{sum})}{ln2},$$

$$\left\lceil log_{1+\varepsilon/(2n)}^{p_{sum}} \right\rceil = \left\lceil \frac{ln(p_{sum})}{ln\big(1+\varepsilon/(2n)\big)} \right\rceil \le \left\lceil \big(1+\frac{2n}{\varepsilon}\big) ln(p_{sum}) \right\rceil \le \left\lceil \big(1+\frac{2n}{\varepsilon}\big)|I| ln2 \right\rceil.$$

The second inequality is derived from $lnz \ge (z-1)/z$ $(\forall z \ge 1)$. And then the total running time of the algorithm is $O[n(1+2n/\varepsilon)^m|I|^m]$. Algorithm accuracy: Suppose π is the schedule get by Algorithm MA, we will prove that the makespan of π is at most $(1+\varepsilon)$ times that of the optimal schedule. We first establish the inequality $(1+x/k)^k \le 1+2x$, for any $0 \le x \le 1$ and any real $k \ge 1$. The left-hand side of the inequality is a convex function in x, and the right-hand side is a linear function in x. Moreover, the inequality holds at $x = 0$ and $x = 1$. Hence, it holds for any $0 \le x \le 1$. In particular, setting $x = \varepsilon/2$ and $k = n$, we have

$$\Big(1+\frac{\varepsilon}{2n}\Big)^n \le (1+\varepsilon).$$

Suppose $C = (c_1, c_2, \cdots, c_m)$ is the completion state of π on machines, and using (π, C) or (n, C) denotes the current state. Let (π, C') or (n, C') be the corresponding state of T_n, where $C' = (c'_1, c'_2, \cdots, c'_m)$. By **Step 2**, c_i may be smaller than c'_i. Let π^* be the optimal schedule for the original instance and the corresponding completion state on machines be $C^* = (c_1^*, c_2^*, \cdots, c_m^*)$. Similar, we let $(n, C^{*'})$ denote the corresponding state of T_n, where $C^{*'} = (c_1^{*'}, c_2^{*'}, \cdots, c_m^{*'})$. Suppose $p_{i_1}^*$ and $p_{i_s}^*$ denote the first and the last job's processing time on machine M_i in π^*. Although the load c_i^* also may be smaller than corresponding $c_i^{*'}$ on machine M_i, we can recursively stretch $p_{i_j}^*$ $(1 \le j \le s)$ as little as possible into

$p_{i_j}^{*'}$, such that $c_i^* = \sum_{j=1}^{s} p_{i_j}^*$ being an integer power of $1+\varepsilon_0$ and $(n, C^{*'}) \in T_n$. Applying

$$\begin{aligned} c_i^{*'} = \sum_{j=1}^{s} p_{i_j}^{*'} &\le \Big(1+\varepsilon_0\Big)\left(\sum_{j=1}^{s-1} p_{i_j}^{*'} + p_{i_s}^*\right) \\ &\le \Big(1+\varepsilon_0\Big)\Big(1+\varepsilon_0\Big)\left(\sum_{j=1}^{s-2} p_{i_j}^{*'} + p_{i_{s-1}}^* + p_{i_s}^*\right) \\ &= \Big(1+\varepsilon_0\Big)^2\left(\sum_{j=1}^{s-2} p_{i_j}^{*'} + p_{i_{s-1}}^* + p_{i_s}^*\right) \\ &\le \cdots \\ &\le \big(1+\varepsilon_0\big)^{(i_s - i_1 + 1)} \sum_{j=1}^{s} p_{i_j}^* \\ &\le \big(1+\varepsilon_0\big)^{n} \sum_{j=1}^{s} p_{i_j}^* \\ &\le \big(1+\varepsilon\big) c_i^*. \end{aligned}$$

That is for any i $(1 \le i \le m)$. It holds that $c_i^{*'} \le (1+\varepsilon_0) c_i^*$. And thus,

$$C_{\max}(\pi^*, C^{*'}) \le (1+\varepsilon) C_{\max}(\pi^*, C^*) \tag{1}$$

By the **Step 2** of the **Algorithm MA**

$$C_{\max}(\pi, C) \le C_{\max}(\pi, C') \tag{2}$$

By the **Step 3** of the **Algorithm MA**

$$C_{\max}(\pi, C') \le C_{\max}(\pi^*, C^{*'}) \tag{3}$$

Applying (1), (2) and (3), we get

$$C_{\max}(\pi, C) \le (1+\varepsilon) C_{\max}(\pi^*, C^*) \tag{4}$$

Where $C_{\max}(\pi, C)$ and $C_{\max}(\pi, C')$ are the makespan corresponding to the state (π, C) and (π, C'), respectively. The meaning of $C_{\max}(\pi^*, C^{*'})$ and $C_{\max}(\pi^*, C^*)$ are similar. □

3.3 Algorithm $MA(\varepsilon)$ for $P_m|rej|C_{\max}/TCP$

From the expression of the running time, the running time of the above algorithm MA becomes longer as the job's processing time increase. Next, we give another $FPTAS$ for $P_m|rej|C_{\max}/TCP$ whose running time isn't influenced by the job's processing time.

We renumber all the jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$. We denote A_i as the dynamic programming algorithm A which schedules jobs without processing times $\{p_1, p_2, \cdots, p_i\}$. We also denote A as algorithm A_0.

Algorithm $MA(\varepsilon)$

Step 1 Input an instance $I = \{J_j = (p_j, e_j) : 1 \leq j \leq n\}$.
Step 2 For $k = 0$ to $n-1$, set $M_k = \frac{\varepsilon}{n+1} p_{k+1}$.
Construct a rounded instance $I'_k = \{J_j = (p'_j, e_j) : 1 \leq j \leq n\}$ and go to **Step 3**, where $p'_j = M_k \lfloor p_j / M_k \rfloor$.
Step 3 Find out the optimal schedule $\tilde{\pi}_k$ of I'_k by calling algorithm A_i:
if $\tilde{\pi}_k$ exists, obtain the corresponding schedule π_k of $\tilde{\pi}_k$;
else set $s = k$ and go to **Step 4**.
Step 4 Find out π_{i_0} that minimizes the makespan from $\{\pi_k : 0 \leq k \leq s-1\}$, then π_{i_0} is a schedule of I.

Theorem 3. *Algorithm $MA(\varepsilon)$ yields a $(1+\varepsilon)$-factor approximation algorithm (FPTAS) for $P_m|rej|C_{\max}/TPC$, which runs in time $O[n^{m+2}\varepsilon^{-m}]$.*

Proof. Time complexity: As the jobs of $I'_k = \{J_j = (p'_j, e_j) : 1 \leq j \leq n\}$ have the common factor, we can delete the factor in executing algorithm A_i and get the corresponding schedule of I'_k. So, the running time of A_0 is:

$$O\Big[np'_{\max}\Big]^m = O\Big[n\Big\lfloor\frac{p_{\max}}{M_0}\Big\rfloor\Big]^m = O\Big[n^{m+1}\big(\frac{1}{\varepsilon}\big)^m\Big], \text{ where } P_{sum} = \sum_{j=1}^{n} p_j.$$

In the same way, the running time of A_i $(1 \leq i \leq s-1)$ is also $O[n^{m+1}\varepsilon^{-m}]$. Since the main running time of the algorithm $MA(\varepsilon)$ is calling algorithms A_i $(0 \leq i \leq s-1)$, the total running time of $MA(\varepsilon)$ is $O[n^{m+2}\varepsilon^{-m}]$.

Algorithm accuracy: For any schedule π, denote it's state as (π, C). Suppose π^* is the optimal schedule of the original instance and it's state is denoted as (π^*, C). Let the largest processing time of job's in π^* is p_t, then holds $p_t \geq p_k$, Let $\tilde{\pi}^*$ is the schedule corresponding to that after π^* is rounded. Obviously, the schedule $\tilde{\pi}^*$ is a feasible schedule of instance I'_{t-1}. We only need to prove: $C_{\max}(\pi_{i_0}) \leq (1+\varepsilon)C_{\max}(\pi^*)$.

Due to rounded, the processing time of job's in π_{t-1} may be longer than that of corresponding job's in $\tilde{\pi}_{t-1}$, but at most longer by a factor M_{t-1}.

So

$$\begin{aligned} C_{\max}(\pi_{t-1}) &\leq C_{\max}(\tilde{\pi}_{t-1}) + nM_{t-1} \\ &\leq C_{\max}(\pi^*) + \varepsilon p_t \\ &\leq C_{\max}(\pi^*) + \varepsilon C_{\max}(\pi^*) \\ &= (1+\varepsilon)C_{\max}(\pi^*). \end{aligned}$$

Applying $C_{\max}(\pi_{i_0}) \leq C_{\max}(\pi_{t-1})$, we get $C_{\max}(\pi_{i_0}) \leq (1+\varepsilon)C_{\max}(\pi^*)$. □

4 Dynamic Programming Algorithm and $FPTAS$ for the Problem $1|rej, r_j|C_{\max}/TPC$

For any set of jobs $\{J_j = (p_j, r_j, e_j) : 1 \leq j \leq n\}$ and an upper bound H about the total penalties (TPC) of rejected jobs. We first design a dynamic programming algorithm for $1|rej, r_j|C_{\max}/TPC$ and get a schedule with the minimum makespan, while it's TPC is at most H.

For any partial schedule for jobs $\{J_1, J_2, \cdots, J_j\}$, if it's makespan is P, we denote its state as (j, P). If (j, P) can be obtained by some partial schedule, we say it's feasible. Let S_j denote the jth state space, i.e., the set of all the feasible states obtained by partial schedules for jobs $\{J_1, J_2, \cdots, J_j\}$. For any $(j, C) \in S_j$, using $\Gamma(j, P)$ represents the minimum TPC of partial schedules whose states are (j, P).

We give the following dynamic programming algorithm for $1|rej, r_j|C_{\max}/TPC$.

Algorithm DP

Step 1 Initialize $S_1; = \{(1, 0), (1, p_1)\}$, $S_j := \phi$ $(j = 1, 2, \cdots, n)$, $\Gamma(1, 0) := e_1$ and $\Gamma(1, p_1) := 0$.

Step 2 For $j = 2$ to n, check every $(j-1, P) \in S_{j-1}$:

(i) If $P \geq r_j$ and $\Gamma(j-1, C) + e_j > H$, set $S_j := \{(j, P + p_j)\}$;
if $P \geq r_j$ and $\Gamma(j-1, C) + e_j \leq H$, set $S_j := \{(j, P + p_j), (j, P)\}$.

(ii) If $P < r_j$ and $\Gamma(j-1, C) + e_j > H$, set $S_j := \{(j, r_j + p_j)\}$;
if $P < r_j$ and $\Gamma(j-1, C) + e_j \leq H$, set $S_j := \{(j, r_j + p_j), (j, P)\}$.

Calculate the corresponding $\Gamma(j, C)$.

Step 3 Output the state vector (n, P^*) with $M(n, P) \leq H$ that minimizes the makespan $C_{\max}$.

Step 4 Give the corresponding schedule of the original instance according to (n, P^*).

It is straightforward that the running time is $O[n(p_{\max} + r_{\max})]$, where $P_{\max} = \max\{p_j : 1 \leq j \leq n\}$ and $r_{\max} = \max\{r_j : 1 \leq j \leq n\}$. Which is a pseudo-polynomial algorithm. In order to further get $FPTAS$ for $1|rej, r_j|C_{\max}/TCP$, we round down job's processing times and arrive times by the way offered in [1].

Lemma 4. [9] *With $1 + \varepsilon$ loss, we can assume that all processing times and arrive times are integer powers of $1 + \varepsilon$.*

From lemma 4, all processing time and arrive times of the rounded instance $I' = \{J_j = (p'_j, r'_j, e_j) : 1 \leq j \leq n\}$ are integer powers of $1 + \varepsilon$, and at most have $\lceil log_{1+\varepsilon}^{r_{\max}} \rceil$ different processing times and $\lceil log_{1+\varepsilon}^{p_{\max}} \rceil$ different arrive times. They are all the polynomial in the size of the corresponding instance.

Algorithm $MDP(\varepsilon)$

Step 1 Input instance $I=\{J_j=(p_j,r_j,e_j):1\le j\le n\}$;
Step 2 Construct the new instance $I'=\{J_j=(p'_j,r'_j,e_j):1\le j\le n\}$ by the way of lemma 4;
Step 3 Calling the algorithm DP obtains the optimal schedule $\tilde{\pi}$ of I';
Step 4 Give the corresponding schedule π of the original instance I according to $\tilde{\pi}$.

From the property of the algorithm DP and the above Lemma 4, the follows theorem can be proved easily.

Theorem 4. *Algorithm $MDP(\varepsilon)$ yields a $(1+\varepsilon)$-factor approximation algorithm (FPTAS) for $1|rej,r_j|C_{\max}/TCP$, which runs in time $\lceil log_{1+\varepsilon}^{np_{\max}r_{\max}}\rceil$.*

5 The Nonexistence of Infinite Worst Case Performance Ratio Approximation for $1|rej,on-line|C_{\max}/TCP$

The three models aforementioned are all off-line cases, i.e., for any instance of scheduling problems, all information (such as the number of jobs, processing times, compression cost etc) about the jobs are known before schedule. While in many applications, the jobs are arrive one by one and any information of job's are not to be known until it arrives. The jobs are scheduled with the passage of time and, at any point of time, the scheduler only has knowledge of those jobs that have already arrived. It is assumed that the scheduler's decision to assign and schedule a job is irrevocable. Under the circumstance, we call the corresponding scheduling as on-line scheduling (or scheduling over time). As the information about jobs' are released alone with the jobs' arriving, which increases the complexity.

Theorem 5. *There doesn't exist any on-line algorithm with a constant competitive ratio for $1|rej,on-line|C_{\max}/TCP$, even if the jobs only have two different release times.*

Proof. For any algorithm E, we only need to consider the following two cases.

Case 1. If by enforcing the algorithm E, the job $J_1=\{p_1,e_1\}$: $p_1=M$, $r_1=0$ is accepted, then another (also the last) job $J_2=\{p_2,e_2\}$: $p_2=\varepsilon$, $e_1+e_2>H$ arrives at the time $r=\varepsilon$. Obviously, by algorithm E, the job J_2 should be rejected. Thus $C_{\max}(E)=M$ and $C^{\star}_{\max}=2\varepsilon$, so

$$\frac{C_{\max}(E)}{C^{\star}_{\max}}=\frac{M}{2\varepsilon}\to+\infty\ \ (\varepsilon\to 0).$$

Case 2. If by enforcing the algorithm E, the job $J_1=\{p_1,e_1\}$: $p_1=M$ $(M>1)$, $r_1=0$ is rejected, then another (also the last) job $J_2=\{p_2,e_2\}$:

$p_2 = M^k$ and $e_1 + e_2 > H$ arrives at the time $r = \varepsilon$. Obviously, by algorithm E, the job J_2 should be accepted. Thus $C_{\max}(E) = \varepsilon + M^k$ and $C^{\star}_{\max} = M$, so

$$\frac{C_{\max}(E)}{C^{\star}_{\max}} = \frac{\varepsilon + M^k}{M} \to +\infty \quad (k \to +\infty).$$

The proof is then finished. □

References

1. Afrati, F., Bampis, E., Chekuri, C., Karger, D., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M.: Approximation Schemes for Minimizing Average Weighted Completion Time with Release Data. In: Proceedings of 40th FOCS, pp. 32–43 (1999)
2. Bartal, Y., Leonardi, S., Marchetti-Spaccamela, A., Sgall, J., Stougie, L.: Multi-Processor Scheduling with Rejection. SIAM Journal of Discrete Maths. 13, 64–78 (2000)
3. Cao, Z., Zhang, Y.: Scheduling with Rejection and Non-Identical Job Arrivals. Journal of System Science and Complexity 20, 529–535 (2007)
4. Engels, D.W., Karger, D.R., Kolliopoulos, S.G., Sengupta, S., Uma, R.N., Wein, J.: Techniques for Scheduling with Rejection. Journal of Algorithms 49(1), 175–191 (2003)
5. Epstein, L., Noga, J., Woeginger, G.J.: On-line Scheduling of Unit Time Jobs with Rejection: Minimizing the Total Completion Time. Operations Research Letters 30, 415–420 (2002)
6. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. Ann. Disc. Math. 5, 287–326 (1979)
7. He, Y., Min, X.: On-Line Uniform Machine Scheduling with Rejection. Computing 65, 1–12 (2000)
8. Seiden, S.S.: Preemptive Multiprocessor Scheduling with Rejection. Theoretical Computer Science 262, 437–458 (2001)
9. Sengupta, S.: Algorithms and Approximation Schemes for Minimizing Lateness/Tardiness Scheduling with Rejection. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 79–90. Springer, Heidelberg (2003)

Scheduling Problems in Cross Docking*

Rongjun Chen[1], Baoqiang Fan[2], and Guochun Tang[3]

[1] Department of Mathematics, Changzhou Institute of Technology
Changzhou 213002, China
chenrjecust@163.com
[2] Department of Mathematics and Information, Ludong University
Yantai 264025, China
baoqiangfan@163.com
[3] Institute of Management Engineering, Shanghai Second Polytechnic University
Shanghai 201209, China
gtang@sh163.net

Abstract. In this paper we study a cross-docking problem minimizing the total flow time of inbound and outbound jobs. Using the theories and methodologies of scheduling, we formulate the problem into a two-stage scheduling problem and propose heuristics with worst-case performance analysis under parallel, uniform and open-shop machines, respectively. For each of problems studied, some polynomially solvable special cases are also introduced.

Keywords: Scheduling; Cross docking; Heuristics.

1 Introduction

In the past decade, the cross-docking problem has led to relatively new logistics techniques in the retail, grocery and distribution industries. The idea is to transfer shipments directly from incoming to outgoing trailers without storage in between. Shipments typically spend less than 24 hours in a crossdock, sometimes less than an hour. In a traditional model, the warehouse maintains stock until a customer orders, then the product is picked, packed, and shipped. When replenishments arrive at the warehouse, they are stored until a customer is identified. In a crossdocking model, the customer is known before the product gets to the warehouse and there is no need to move it to storage. Clearly, cross docking will consolidate shipments from disparate sources and will help distributors to realize economies of scale in outbound transportation.

In fact cross docking system has been widely applied for many years to decayed or time sensitive product in many industries. Many famous companies such as Wal-Mart, Home Depot, Costco, Canadian Tire , Fedex Freight, etc, are now implementing cross docking system. Wal-mart is well know as a pioneer in implementing cross-docking operations. Due to the cross docking system, Wal-Mart essentially eliminates all inventory holding costs. At one distribution center

* Supported by the National Science Foundation of China (No. 70731160015) and the National Natural Science Foundation of Jiangsu (Grant No. yw06037).

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 421–429, 2009.

in California, direct freight accounts for a whopping 60% of all items shipped. Recently cross docking technique is also extensively implemented in most logistic companies in China.

In 2003, Ratliff et al. [10] studied the design of cross docking network. Motivated by vehicle loading, they determined the number of cross docks, locations and material assignment strategies through a mixed integer programming. Also in 2003, Donaldson et al. [3] studied the cross docking network design under timetable constraint. Based on vehicle dispatch constraint, they have proposed vehicle routes and the number of vehicles on each of the routes. Heragu et al. [6] further studied warehousing design with cross dock and goods assignment. They created a mathematical programming to minimize the total cost according to material flow in warehouse and proposed heuristics to solve the mathematical programming. In 2004, Mahnet Gumus [5] addressed the number of cross docks and its influence to the total cost. Based on the time window of delivery and order picking, Chen et al. [1] proposed for the cross docking network problem an integer programming to minimize the total cost of transportation and inventory. They proved the NP hardness of the problem and proposed a greedy algorithm and a tabu search algorithm. Xu et al. [11,12] studied the transshipment problem in cross docking system with stochastic demand in 2004. Ma Deliang [9] proposed an integrated storage and transportation model under VMI system.

In this paper, we study the efficiency of the cross-docking system from a scheduling point of view. In a cross-dock setting, there are inbound and outbound products. Each of inbound products needs to be downloaded and unpacked while outbound products need to be collected and packed. We formulate the cross docking problem with inbound and outbound products as a two-machine flow shop problem $F2||C_{\max}$, where an operation on the first machine is to download and unpack the inbound jobs, while collecting and packing those products with the same destination into an outbound trailer is considered as an operation on the second machine. Use the theories and methodologies of scheduling, we develop heuristics for the crossing docking problems. This paper is organized as follows. In Section 2, we mathematically define the problem and provide some optimality properties. In Section 3 a heuristic is presented for cross docking problem with parallel machines and some special cases that are polynomially solvable are addressed. Section 4 proposes heuristics for cross docking problem with uniform machines and open shop problems, respectively. Some polynomially solvable special cases are also introduced. Section 5 includes concluding remarks and further research directions.

2 Concepts and Notations

Consider a cross-dock setting in a third party logistics company in which we assume that there are n inbound jobs. Different jobs can be needed by same destination. Each inbound job needs to be downloaded and unpacked. In the meanwhile, there are m outbound trailers each carries several products with the same destination. We assume that due to space constraint, there is only one outbound trailer working at any time and it will work only if all products needed

to be loaded on to that trailer are ready. In this situation, the scheduling of inbound products download-and-unpack operations and that of outbound products collect-and-pack operations will affect the efficiency of the cross-docking system as well as that of the total supply chain system. Our purpose is to sequence the download-and-unpack operations for the inbound jobs as well as the collect-and-pack operations for the outbound jobs to minimize the makespan.

Based on the cross-dock setting mentioned above, we can formulate the cross docking problem as a two-machine flow shop problem $F2||C_{\max}$, where an operation on the first machine M_1 is to download and unpack the inbound jobs, while collecting and packing those products with the same destination into an outbound trailer is considered as an operation on the second machine M_2. Note that an operation on the second machine M_2 can not start unless the corresponding job has been downloaded and unpacked on the first machine M_1. But our problem studied here is more complicate than two-machine flow shop problem because of the following two reasons. (1)The processing facility at the first stage is not a single machine, but parallel, uniform or open-shop machine.(2)The number of the first stage operations could be different from that of the second stage operations.

We denote inbound jobs as $N^1 = \{J_{i1}, i = 1, 2..., n\}$, which need to be downloaded and unpacked. We refer these jobs as the first stage jobs. Job $J_{i1}(i = 1, 2..., n)$ requires a processing time of p_{i1}. In the meanwhile, we denote outbound jobs as $N^2 = \{J_{j2}, j = 1, 2, ..., m\}$, which need to be collected and packed. We call them the second stage jobs. The processing time of J_{j2} is p_{j2}. For each $J_{j2}, j = 1, 2, ..., m$, there is a corresponding subset of the first stage jobs, call it as S_j, such that J_{j2} can be processed only after all jobs in S_j have been completed on the first stage machine. Without loss of generality, assume $N^1 = \cup_{j=1}^{m} S_j$. We study the scheduling of all jobs at the first and second stage, such that the makespan $C_{\max}$, the time by which all of the first and second jobs are completed, is minimal. Assume that the first stage has a k parallel, uniform or open-shop machine problem and the second stage has a single machine problem. Use the three parameters notation, these problems are denoted by $(Pk, F2)|CD|C_{\max}$, $(Qk, F2)|CD|C_{\max}$ and $(Ok, F2)|CD|C_{\max}$, respectively, where k is the number of machines.

For clarity, we summarize the notations mentioned above in the following table.

inbound/outbound jobs	processing times
$N^1 = \{J_{i1}, i = 1, 2..., n\}$	$p_{i1}, i = 1, 2..., n$
$N^2 = \{J_{j2}, j = 1, 2..., m\}$	$p_{j1}, j = 1, 2..., m$

Due to the NP hardness of problems $Pk||C_{\max}$ [8] and $Om||C_{\max}$ [4], the following Lemma 1 holds.

Lemma 1. $(Pk, F2)|CD|C_{\max}, (Qk, F2)|CD|C_{\max}, (Ok, F2)|CD|C_{\max}$ *are all NP-hard.*

Property 1. *There exists an optimal schedule for problems* $(Pk, F2)|CD|C_{\max}$ *and* $(Qk, F2)|CD|C_{\max}$ *such that inbound jobs are processed consecutively on the first stage machines.*

Proof. If not, we can move forwardly jobs at the first stage such that inbound jobs are processed consecutively. The new solution is still feasible without increasing the makespan. □

Property 2. *There exists an optimal schedule for problems* $(Pk, F2)|CD|C_{\max}$, $(Qk, F2)|CD|C_{\max}$ *and* $(Ok, F2)|CD|C_{\max}$ *such that if the completion time of jobs in* S_j *is earlier than that of* S_k *on the first stage machines, then* J_{j2} *is processed before* J_{k2} *on the second stage machine.*

Proof. If not, i.e., J_{j2} is processed after J_{k2} at the second stage, we can move J_{j2} forwardly to start at the starting time of J_{k2} and move all jobs originally between J_{k2} and J_{j2} (including J_{k2}) to the right by p_{j2}. The new solution is still feasible without increasing the makespan. Continue the similar shift and we can obtain the required optimal schedule. □

Due to the proof of Property 2, we have

Property 3. *There exists an optimal schedule for problems* $(Pk, F2)|CD|C_{\max}$, $(Qk, F2)|CD|C_{\max}$ *and* $(Ok, F2)|CD|C_{\max}$ *such that* J_{k2} *is processed after* J_{j2} *at the second stage if* $S_j \subseteq S_k$ *for* $j, k \in N^2$.

Finally, we introduce Johnson's algorithm for $F2||C_{\max}$ problem which will be cited in the following two sections.

Johnson's Algorithm [7]
(1) Let the jobs be labelled as $1, 2, ..., n$ and construct sets X and Y as

$$X = \{j|\tau_{j1} < \tau_{j2}\}, Y = \{j|\tau_{j1} \geq \tau_{j2}\},$$

where τ_{j1}, τ_{j1} are the processing times of job j on the first and second machine, respectively.
(2) Arrange jobs in X in nondecreasing τ_{j1}-order. Arrange jobs in Y in nonincreasing τ_{j2}-order. Call these ordered sets $\hat{\text{X}}$ and $\hat{\text{Y}}$.
(3) Concatenate $\hat{\text{X}}$ and $\hat{\text{Y}}$ forming the permutation { $\hat{\text{X}}$ $\hat{\text{Y}}$ } which is the processing order for both machines.

3 $(Pk, F2)|CD|C_{\max}$

Firstly, we study $(Pk, F2)|CD|C_{\max}$ problem where the first stage has a k parallel machine problem and the second stage has a single machine problem. Obviously, when $m = 1$ or $k = 1$, the problem is equivalent to $Pk||C_{\max}$ or $F2|prec|C_{\max}$ [2], respectively. Both of the two problem are NP hard. The following is a polynomial approximation algorithm for $(Pk, F2)|CD|C_{\max}$.

Algorithm HP
(1) Let $u_i = \frac{p_{i1}}{k}, v_i = \sum_{j:J_{i1}\in S_j} \frac{p_{j2}}{|S_j|}, i = 1, 2, ..., n.$
(2) Apply Johnson's algorithm to the two-machine flow problem $F2||C_{\max}$ with n jobs and with processing times (u_i, v_i) and obtain its optimal solution $F = (\sigma^*, \sigma^*)$.
(3) At the first stage, schedule the first available job from the list σ^*, whenever any machine of the k parallel machines becomes available. At the second stage, process J_{j2} as early as possible after all jobs in S_j have been completed by the list of σ^* on k parallel machines.

Theorem 1. *Using Algorithm HP, an approximation solution to* $(Pk, F2)|CD|C_{\max}$ *can be obtained in polynomial time and the performance ratio is no more than* $2 - \frac{1}{\max\limits_{j=1,2,...,m}\{k,|S_j|\}}$.

Proof. Let τ be the last finished job at the first stage and C_{HP} be the makespan of schedule obtained by HP. We have

$$C_{HP} \le C_F + \sum_{j=1}^{m} p_{2j}\Big(1 - \frac{1}{|S_j|}\Big) + \Big(\frac{k-1}{k}\tau + \frac{\sum_{i=1}^{n} p_{i1}}{k}\Big) - \sum_{i=1}^{n}\frac{p_{i1}}{k}$$

$$\le C_F + \sum_{j=1}^{m} p_{2j}\Big(1 - \frac{1}{|S_j|}\Big) + \frac{k-1}{k}\tau$$

$$\le C_* + \sum_{j=1}^{m} p_{2j}\Big(1 - \frac{1}{\max\limits_{j=1,2,...,m} S_j}\Big) + \frac{k-1}{k}\max_{i=1,2,...,n} p_{i1}$$

$$\le C_* + \Big(\sum_{j=1}^{m} p_{2j} + \max_{i=1,2,...,n} p_{i1}\Big)\Big[1 - \frac{1}{\max\limits_{j=1,2,...,m}\{k, |S_j|\}}\Big]$$

$$\le C_*\Big(2 - \frac{1}{\max\limits_{j=1,2,...,m}\{k, |S_j|\}}\Big),$$

where C_F be the makespan of two-machine flow shop problem obtained at step 2 of HP Algorithm. □

Next we discuss some polynomially solvable cases of $(Pk, F2)|CD|C_{\max}$.

Property 4. *For* $(Pk, F2)|CD|C_{\max}$, *there exists an optimal schedule such that (1) if jobs in* N^2 *are processed on the second stage machine in the sequence* $(J_{12}, J_{22}, ..., J_{m2})$, *then the jobs of* N^1 *are processed on each of k parallel machines in the following order:* $S_1, S_2\backslash S_1, ..., S_m\backslash(\cup_{j=1}^{m-1} S_j)$. *(2)if* $S_j \cap S_k = \emptyset, \forall k(\neq j)$ *and a given* $J_{j2} \in N^2$, *the jobs in* S_j *are processed consecutively on each of k parallel machines.*

Proof. (1) If jobs in S_1 are not processed first on some machine of k parallel machines, we just on that machine move them to the first positions and move

other affected jobs to the right accordingly. After the move, the schedule is still feasible without increasing the makespan. We can then move jobs in $S_2 \backslash S_1$ to be processed immediately after S_1 and move other affected jobs to the right accordingly. After the move, the schedule is still feasible without increasing the makespan. Repeat the process eventually we reach our goal.

(2) If jobs in S_j are not consecutive on some machine, we just move all jobs in S_j on that machine to the right to make them be processed consecutively and finish at the same time as before. Those jobs affected and not in S_j will be moved to the left accordingly. The new solution is still feasible without increasing the makespan. □

Polynomially solvable case 1 of $n \leq k$.
In this situation, the problem can be transferred to $1|r_j|C_{\max}$ which is polynomially solvable.

Polynomially solvable case 2 of $k = 1, S_1 \subseteq S_2 \subseteq \cdots \subseteq S_m$.
By Properties 3 and 4, the optimal schedule is that jobs in N^1 are processed at the first stage in the order of $S_1, S_2 \backslash S_1, ..., S_m \backslash (\cup_{j=1}^{m-1} S_j)$ while jobs in N^2 are processed in the sequence $(J_{12}, J_{22}, ..., J_{m2})$.

Polynomially solvable case 3 of $k = 1, S_j \cap S_l = \emptyset, \forall j, l \in \{1, 2, ..., m\}$.

Algorithm P
(1) Let $u_j = \sum_{i:J_{i1} \in S_j} p_{i1}, v_j = p_{j2}, j = 1, 2, ..., m$.
(2) Apply Johnson's algorithm to the two-machine flow problem $F2||C_{\max}$ with n jobs and with processing times (u_i, v_i) and obtain its optimal solution $F = (\sigma^*, \sigma^*)$.
(3) At the second stage, jobs in N^2 are processed in the order of σ^*. At the first stage, the sequence of jobs in N^1 is ordered according to Property 4(1).

Theorem 2. *The optimal solution to Case 3 of* $(Pk, F2)|CD|C_{\max}$ *can be obtained by Algorithm P in polynomial time.*

Proof. By Property 4(2), there exists an optimal solution such that jobs in each $S_j (j = 1, 2, ..., m)$ are processed consecutively at the first stage. We thus can combine those jobs in S_j into a combined one with processing time u_j. Thus the problem is same as $F2||C_{\max}$. □

4 $(Qk, F2)|CD|C_{\max}$, $(Ok, F2)|CD|C_{\max}$

In this section we first study $(Qk, F2)|CD|C_{\max}$ problem with k uniform machines at the first stage. Assume machine speeds $s_i, i = 1, 2, ..., k$ satisfy that

$$s_1 \geq s_2 \geq ... \geq s_k.$$

Based on the HP algorithm, we first construct HQ to solve $(Qk, F2)|CD|C_{\max}$.

Algorithm HQ
(1) Let $u_i = \frac{p_{i1}}{ks_1}, v_i = \sum\limits_{j:J_{i1}\in S_j} \frac{p_{j2}}{|S_j|}, i = 1, 2, ..., n;$
(2) Apply Johnson's algorithm to the two-machine flow problem $F2||C_{\max}$ with n jobs and with processing times (u_i, v_i) and obtain its optimal solution $F = (\sigma^*, \sigma^*)$.
(3) At the first stage, schedule the first available job from the list σ^* on an available machine of the k uniform machines such that the job last selected finish as early as possible. At the second stage, process J_{j2} as early as possible after all jobs in S_j have been completed on k uniform machines.

Theorem 3. *Using Algorithm HQ, an approximation solution to the problem* $(Qk, F2)|CD|C_{\max}$ *can be obtained in polynomial time and the performance ratio is no more than* $3 - \frac{1}{\max\limits_{j=1,2,\ldots,m} |S_j|} - \frac{s_k}{s_1}$.

Proof. Let C_{HQ} be the makespan of schedule obtained by HQ. We have

$$C_{HQ} \le C_F + \sum_{j=1}^{m} p_{2j}\Big(1 - \frac{1}{|S_j|}\Big) + \Big(\frac{\sum_{i=1}^{n} p_{i1}}{ks_k} - \frac{\sum_{i=1}^{n} p_{i1}}{ks_1}\Big)$$

$$\le C_F + \sum_{j=1}^{m} p_{2j}\Big(1 - \frac{1}{\max_j |S_j|}\Big) + \frac{\sum_{i=1}^{m} p_{i1}}{ks_k}\Big(1 - \frac{s_k}{s_1}\Big)$$

$$\le C_* + C_*\Big(1 - \frac{1}{\max_{j=1,2,\ldots,m} S_j}\Big) + C_*\Big(1 - \frac{s_k}{s_1}\Big)$$

$$\le C_*\Big(3 - \frac{1}{\max_{j=1,2,\ldots,m} |S_j|} - \frac{s_k}{s_1}\Big),$$

where C_F be the makespan of two-machine flow shop problem obtained at step 2 of Algorithm HQ. □

Note. Problem $(Qk, F2)|CD|C_{\max}$ with $p_{i1} = 1, m = 1$, can be transformed into $Qk|p_i = 1|C_{\max}$ which is equivalent to the Assignment problem and thus can be solved Polynomially.

Next we study the problem $(Ok, F2)|CD|C_{\max}$ with k open shop machines at the first stage. Let $\{M_1, M_2, \cdots, M_k\}$ be the set of machines. Each job $J_{i1}(i = 1, 2, ..., n)$ consists of a set $\{O_{1i1}, \cdots, O_{ki1}\}$ of operations, and operation O_{ti1} has to be processed on machine M_t for p_{ti1} time units. Similarly based on HP algorithm, we construct HO algorithm to solve $(Ok, F2)|CD|C_{\max}$.

Algorithm HO
(1) Let $u_i = \frac{\sum\limits_{t=1}^{k} p_{ti1}}{k}, v_i = \sum\limits_{j:J_{i1}\in S_j} \frac{p_{j2}}{|S_j|}, i = 1, 2, ..., n;$
(2) Apply Johnson's algorithm to the two-machine flow problem $F2||C_{\max}$ with n jobs and with processing times (u_i, v_i) and obtain its optimal solution $F = (\sigma^*, \sigma^*)$.

(3) At the first stage, select the first available job from the list σ^* and construct dense schedule for open shop problem at the first stage by greedy algorithm. At the second stage, process J_{j2} on the second stage machine as early as possible after all jobs in S_j have been completed on k open shop machines.

Theorem 4. *Using Algorithm HO, an approximation solution to the problem* $(Ok, F2)|CD|C_{\max}$ *can be obtained in polynomial time and the performance ratio is no more than* $4 - \frac{1}{\max_{j=1,2,\ldots,m}|S_j|}$.

Proof. Let C_{HO} be the makespan of schedule obtained by HO. We have

$$C_{HO} \leq C_F + \sum_{j=1}^{m} p_{2j}\Big(1 - \frac{1}{|S_j|}\Big) + 2C^*_{Ok||C_{\max}}$$

$$\leq C_* + \sum_{j=1}^{m} p_{2j}\Big(1 - \frac{1}{\max\limits_{j=1,2,\ldots,m}|S_j|}\Big) + 2C_*$$

$$\leq C_*\Big(4 - \frac{1}{\max\limits_{j=1,2,\ldots,m}|S_j|}\Big),$$

where $C^*_{Ok||C_{\max}}$ be the optimal makespan of the problem $Ok||C_{\max}$ and C_F be the makespan of two-machine flow shop problem obtained at Step 2 of Algorithm HO. □

Two polynomially solvable cases to $(Ok, F2)|CD|C_{\max}$:

Polynomially solvable case 1 of $k = 2, m = 1$.
In this situation, the problem is equivalent to $O2||C_{\max}$ which is polynomial.

Polynomially solvable case 2 of $p_{ti1} = p_i, m = 1$.
The problem is equivalent to $Ok|p_{ij} = f_j|C_{\max}$ which is polynomial.

5 Conclusion

In this paper, we have formulated the cross-docking problems into two-stage scheduling problems and propose several heuristic algorithms to address the scheduling problems with performance analysis. Three different types of machines, parallel, uniform, and open-shop, are considered in the scheduling problem and in each case, besides heuristic algorithms and approximation ratio analysis, some special polynomially solvable cases are introduced. Future research direction includes developing good heuristics with better error bound and considering the model with other objective functions.

References

1. Chen, P., Guo, Y.S., Andrew, L.: Multiple Crossdocks with Inventory and Time Windows. Computers and Operations Research 33(1), 43–63 (2006)
2. Chen, F., Song, K.L.: Cross Docking Logistics Scheduling Problem and its Approximative and Precise Algorithms. Industrial Engineering and Management 6, 53–58 (2006) (in Chinese)

3. Donaldson, H., Johnson, E.L., Ratliff, H.D., Zhang, M.: Schedule-Driven Cross-Docking Networks. Technical Report, the Logistics Institute, Georgia Institute of Technology, Atlanta, GA (2003)
4. Gonzalez, T., Sahni, S.: Open Shop Scheduling to Minimize Finish Time. J. ACM 21, 665–679 (1976)
5. Gumus, M., James, H.B.: Cross-Docking and its Implications in Location-Distribution Systems. J. of Business Logistics 25(2), 221–232 (2004)
6. Heragu, S.S., Jason, C., Huang, S.: A Mathematical Model for Warehouse Design and Product Allocation. International Journal of Production Research 42(4), 122–131 (2003)
7. Johnson, S.M.: Optimal Two-and Three Stage Production Schedules with Setup Times Included. Naval Research Logistics Quarterly 1, 61–68 (1954)
8. Lenstra, J.K., Rinnooy, K.A.H.G., Brucker, P.: Complexity of Machine Scheduling Problems. ADM 1, 343–362 (1977)
9. Ma, D.L., Zhang, J.Y., Dai, H.L.: A Controlling Logistics Cost Optimal Model under VMI System. J. of Donghua University 6, 35–39 (2003)
10. Ratliff, H.D., Vate, J.V., Zhang, M.: Network Design for Load-Driven Cross-Docking Systems[EB/OL]. GIT Technical Report, the Logistics Institute, Georgia Institute of Technology, Atlanta, GA (2003)
11. Xu, T.L., Luo, S.Y.: The Expected Total Cost Method of Lateral Transshipment in a Cross-Docking System with Stochastic Demand. Industrial Engineering and Management 1, 27–32 (2004) (in Chinese)
12. Xu, T.L., Xiong, H.: The Method of Searching the Best Time for One-Off Transshipment in a Cross-Docking System with Stochastic Demand. Systems Engineering 9, 23–27 (2004) (in Chinese)

Makespan Minimization with Machine Availability Constraints

Bin Fu[1,⋆], Yumei Huo[2,⋆⋆], and Hairong Zhao[3]

[1] Department of Computer Science
University of Texas–Pan American, Edinburg, TX 78539, USA
binfu@cs.panam.edu
[2] Department of Computer Science
College of Staten Island, CUNY, Staten Island, New York 10314, USA
huo@mail.csi.cuny.edu
[3] Department of Mathematics, Computer Science & Statistics
Purdue University Calumet, Hammond, IN 46323, USA
hairong@calumet.purdue.edu

Abstract. We investigate the problems of scheduling n jobs to $m = m_1 + m_2$ identical machines where m_1 machines are always available, m_2 machines have some specified unavailable intervals. The objective is to minimize the makespan. We assume that if a job is interrupted by the unavailable interval, it can be resumed after the machine becomes available.

We show that if at least one machine is always available, i.e. $m_1 > 0$, then the PTAS for Multiple Subset Sum problem given by Kellerer [3] can be applied to get a PTAS; otherwise, $m = m_2$, every machine has some unavailable intervals, we show that if $(m-1)$ machines each of which has unavailable intervals with total length bounded by $\alpha(n) \cdot P_{sum}/m$ where P_{sum} is the total processing time of all jobs and $\alpha(n)$ can be any non-negative function, we can develop a $(1 + \alpha(n) + \epsilon)-$approximation algorithm for any constant $0 < \epsilon < 1$; finally we show that there does not exist any polynomial time $(1 + \alpha(n) - o(1))-$approximation unless P=NP.

Keywords: makespan, machine availability constraints,inapproximation, PTAS.

1 Introduction

Scheduling problems with machine availability constraints have received considerable attention from researchers in the last two decades. These models reflect the real-world situations where the machines have unavailable intervals for processing jobs due to breakdown, preventive maintenance or processing unfinished jobs from a previous planning horizon. Various criteria and machine environments have been studied, see for example [1,2,10,11,12,13,16,20,21], etc. More

⋆ This author is supported by National Science Foundation Early Career Award 0845376.

⋆⋆ This work is partially supported by PSC-CUNY research grant.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 430–437, 2009.

information can be found in the surveys by Saidy et. al [17], Schmidt ([19]), Lee ([14]) and the references therein.

In this paper we study the problems of scheduling n jobs to m identical machines. We use $J = \{J_1, J_2, \cdots, J_n\}$ to denote the job set. Each job J_i has a processing time p_i. Let $P_{sum} = \sum_{j=1}^{n} p_j$. Let $M = \{M_1, M_2, \cdots, M_{m_1}, M_{m_1+1}, \cdots, M_{m_1+m_2}\}$ be a set of $m = m_1 + m_2$ identical machines, where machines $M_1, M_2, \cdots, M_{m_1}$ are always available and machines $M_{m_1+1}, \cdots, M_{m_1+m_2}$ have some unavailable intervals. We assume that the unavailable intervals are known beforehand and all jobs are available to process from the beginning. Given a schedule S, the completion time of job J_i in S is denoted by $C_i(S)$. If S is clear from the context, we use C_i for short. The goal is to schedule the set of n jobs to m identical machines so as to minimize the makespan, $C_{max} = \max\{C_i\}$. Two cases have been considered in the literature with regard to the jobs' resumability. A job is nonresumable ($nr - a$) if interrupted by the unavailable interval, a job has to be restarted after the interval; and a job is resumable ($r - a$) if it can be resumed after the interval. By extending the 3-field notation, the problems can be denoted by $P_{m_1,m_2} \mid nr - a \mid C_{max}$ and $P_{m_1,m_2} \mid r - a \mid C_{max}$ for the non-resumable and resumable cases, respectively. In this paper, we consider the resumable case.

Literature Review. When the machines are always available, the parallel machine scheduling problem, denoted as $P \mid\mid C_{max}$, is strongly NP-Hard; see Garey and Johnson [7]. Hochbaum and Shmoys [8] have given a PTAS for this problem. For more results about this problem, please see the survey paper by Chen et al. [4].

With the constraint of limited machine availability, some research has been done on the problem of minimizing makespan. Lee [15] and Kellerer [10] gave constant approximation algorithms for the special case of machine availability where each machine M_i has a release time r_i, i.e the machine is not available until time r_i. Note that the technique of [8] can be used to obtain a PTAS for this problem. Lee [13] studied the problem $P_{1,m} \mid nr - a \mid C_{max}$ with the constraint that each machine has at most one unavailable period and analyzed the performance of LPT rule. Hwang et. al [9] also studied the same problem with additional constraint that at most $\lambda \in [m-1]$ machines are permitted to be unavailable simultaneously. They proved a tight bound of the LPT rule. In [18], Scharbrodt et al. give approximation schemes and inapproximation results for the problems such that some machines are not available for some periods due to the fixed jobs scheduled there. Recently, Diedrich et al. [5] show that no constant approximation exists for $P_{0,m} \mid nr - a \mid C_{max}$ when $m \geq 2$ and develop a PTAS for $P_{1,m} \mid nr - a \mid C_{max}$ based on PTAS for multiple subset sum problems given by Kellerer [3].

When the jobs are resumable, Lee [13] studied the problem $P_{1,m} \mid r - a \mid C_{max}$ with the constraint that each machine has at most one unavailable period. He showed that LPT rule can yield an arbitrarily large performance ratio while a modified LPT rule has a tight worst case performance of $\frac{3}{2} - \frac{1}{2(m+1)}$.

New Contributions. In this paper, we study the resumable scheduling problem $P_{m_1,m_2} \mid r-a \mid C_{max}$. We first study the case $m_1 \geq 1$, i.e. at least one machine is always available. We show that the PTAS for Multiple subset sum problem given by Kellerer [3] can be applied to obtain a PTAS. Otherwise $m_1 = 0$. We consider the case that all but one machine each has unavailable intervals whose total length is bounded by $\alpha(n)P_{sum}/m$. For convenience, we denote our problem as $P_{0,m}, \alpha(n) \mid r-a \mid C_{max}$. We show that there is a $(1+\alpha(n)+\epsilon)-$approximation algorithm for any constant $0 < \epsilon < 1$. and we show that there does not exist any polynomial time $(1+\alpha(n)-o(1))-$approximation unless P=NP.

Organization. Our paper is organized as follows. In Section 2 and 3, we study the problems $P_{m_1\geq 1,m_2} \mid r-a \mid C_{max}$ and $P_{0,m}, \alpha(n) \mid r-a \mid C_{max}$, respectively. In Section 4, we draw the conclusion.

2 $P_{m_1\geq 1,m_2} \mid r-a \mid C_{max}$

In this section, we study the problem such that at least one machine is always available. We will show that PTAS for Multiple Subset Sum Problem (MSSP) with different knapsack capacities given by Kellerer [3] can be applied to solve this problem. The MSSP with different knapsack capacities is the problem of assigning items from a given ground set to a given number of knapsacks such that the sum of the item weights in every knapsack does not exceed its capacity and the total sum of the weights of the packed items is as large as possible.

For any instance of $P_{m_1\geq 1,m_2} \mid r-a \mid C_{max}$, if we knew the value of the optimal makespan $C^*_{\max}$, then we can apply Kellerer's PTAS for MSSP to obtain a schedule whose makespan is at most $(1+\epsilon)C^*_{\max}$ as follows: for each job J_i, create an item i whose weight is the same as the job's processing time; for each machine M_i, $1 \leq i \leq m_1+m_2$, we create a knapsack with the capacity $(C^*_{\max} - \overline{A}_i(C^*_{\max}))$ where $\overline{A}_i(C^*_{\max})$ is the total length of unavailable intervals before time $C^*_{\max}$ on M_i; then by applying Kellerer's PTAS, a set of items with total weight of $(1-\epsilon/m)P_{sum}$ can be selected and assigned to these m knapsacks; finally we schedule the jobs basing on the assignment of the items to the knapsacks, that is, if item i is assigned to a knapsack corresponding to machine M_i, then we schedule the job J_i to M_i; if item i is not assigned to any knapsack, then we schedule job J_i to machine M_1 which is always available by our assumption.

Since all jobs are resumable and the jobs assigned to M_i, $i \neq 1$, correspond to the items packed in the knapsack with capacity $(C^*_{\max} - \overline{A}_i(C^*_{\max}))$, it is easy to see that the completion time of the last job on machine M_i, is at most $C^*_{\max}$. On the other hand, the completion time of the last job on machine M_1, is at most $C^*_{\max}$ plus the total weight of the jobs corresponding to the unselected items, which is at most $\epsilon P_{sum}/m$. Since P_{sum}/m is a lower bound of $C^*_{\max}$, therefore the makespan of the produced schedule is at most $(1+\epsilon)C^*_{\max}$.

To find the optimal $C^*_{\max}$, we operate a binary search on the range of $[P_{sum}/m, P_{sum}]$.

Theorem 1. *There is a PTAS for* $P_{m_1\geq 1,m_2} \mid r-a \mid C_{max}$.

3 $P_{0,m}, \alpha(n) \mid r - a \mid C_{max}$

In this section, we study the problem $P_{0,m}, \alpha(n) \mid r - a \mid C_{max}$. In this case, there are m machines. We consider the case that machine M_i, $1 \leq i \leq m-1$, each has unavailable intervals whose total length is bounded by $\alpha(n)P_{sum}/(m)$, where $\alpha(n)$ can be any non-negative function; machine M_m may have arbitrary unavailable intervals. We first show that for any constant $0 < \epsilon < 1$, there is a $(1 + \alpha(n) + \epsilon)$-approximation algorithm. Then we will show that there does not exist any polynomial time $(1 + \alpha(n) - o(1))-$approximation unless P=NP.

3.1 Approximation Algorithm

Our algorithm is similar to the one in [6] which is used to minimize the total weighted completion time.

A feasible schedule S partitions the jobs in $J = \{J_1, J_2, \cdots, J_n\}$, into m sets, $X_1, \cdots, X_m$, where X_i $(1 \leq i \leq m)$ contains the jobs allocated to machine M_i. Without loss of generality, one can assume that S does not contain any idle time between the jobs on each machine. Since the order of the jobs on each machine does not have any effect on the makespan, we consider two schedules to be same if they have the same set of jobs on each machine. In this way, there is a one-to-one correspondence between a schedule S of the jobs and a tuple $(X_1, \cdots, X_m)$ that partitions the jobs into m sets. Our algorithm is described below.

Main-Algorithm(I,ϵ)

Input: I, an instance of $P_{0,m}, \alpha(n) \mid r-a \mid C_{max}$; ϵ: a constant with $0 < \epsilon < 1$

1. Let $f = (1 + \frac{\epsilon}{2 \log n})$
2. Let L be the list of tuples returned by Sub-Algorithm(I, f)
3. Return the schedule that corresponds to a tuple in L and has minimum makesapn.

End of Main-Algorithm

Sub-Algorithm(I,f)

Input: I, an instance of $P_{0,m}, \alpha(n) \mid r - a \mid C_{max}$; f, the error control parameter which is a positive constant

1. If $n = 1$, there is only a single job J_1, return a set of m tuples $L = ((J_1, \emptyset, \cdots, \emptyset, \emptyset), (\emptyset, J_1, \cdots, \emptyset, \emptyset), \cdots, (\emptyset, \emptyset, \cdots, \emptyset, J_1)$
2. Let I_1 and I_2 be the instances derived from I that contain jobs $J^1 = \{J_1, J_2, \cdots, J_{\lfloor \frac{n}{2} \rfloor}\}$, and $J^2 = \{J_{\lfloor \frac{n}{2} \rfloor + 1}, \cdots, J_n\}$, respectively.
3. Let L_1=Sub-Algorithm(I_1, f).
4. Let L_2=Sub-Algorithm(I_2, f).
5. Merge L_1, L_2 into a single set L:
   For each tuple $u \in L_1$, $u = (u_1, \cdots, u_m)$
   for each tuple $v \in L_2$, $v = (v_1, \cdots, v_m)$
   generate a tuple $(u_1 \cup v_1, \cdots, u_m \cup v_m)$, insert into L.
6. Return Tuple-Prune(L, f) (described below)

End of Sub-Algorithm

To complete the algorithm, we show how tuples are pruned so that the size of L is not large while one of the tuple in L still corresponds to a good approximation of the optimal solution. Given the error control parameter f, we divide the interval $[0, P_{sum}]$ into segments at points f^i, $0 \leq i \leq \lfloor \log_f P_{sum} \rfloor$. We use I_0, I_1, ..., to denote the segments. For a subset of jobs Q, we define $P(Q) = \sum_{J_i \in Q} p_i$. For the purpose of prune, we associate each tuple $(u_1, \cdots, u_m)$ with a vector in R^m, $(P(u_1), \cdots, P(u_m))$. We say two tuples $(u_1, \cdots, u_m)$ and $(v_1, \cdots, v_m)$ are in the same region if there exist segments $I_{k_1}, I_{k_2}, \cdots, I_{k_m}$ such that the vectors associated with these two tuples are both in $I_{k_1} \times I_{k_2} \times \cdots \times I_{k_m}$. It is obvious that there are at most $O((\log_f P_{sum})^m)$ different regions.

Given a list L of tuples and the error control parameter f, **Tuple-Prune**(L, f), selects for each region the tuple $(u_1, \cdots u_{m-1}, u_m)$ such that $P(u_m)$ is the least among all tuples in the region.

Next we analyze the Sub-Algorithm. For convenience, we assume that $n = 2^i$ for some integer $i \geq 0$. Otherwise, we can always append some dummy jobs with $p_j = 0$, which does not change the makespan of any schedule.

Lemma 1. *For any schedule $S = (X_1, \cdots, X_m)$, there exists a tuple $(u_1, \cdots, u_m)$ in the list returned by Sub-Algorithm such that the following conditions hold:*

(1) $P(u_j) \leq f^i \cdot P(X_j)$ *for* $1 \leq j \leq m-1$
(2) $P(u_m) \leq P(X_m)$.

Proof. We prove the lemma by induction based on i. When $i = 0$, the proof is trivial. Assuming that the claim is true for $i-1$, we now verify the hypothesis for i.

The Sub-Algorithm divides the $n = 2^i$ jobs into J^1 and J^2 and call itself recursively on J^1 and J^2. For any two sets of jobs, A and B, we define $A[B]$ to be the jobs in $A \bigcap B$. By inductive hypothesis, in L_1, there is a tuple $(u_1^{[1]}, \cdots, u_m^{[1]})$ that partitions the jobs in J^1 such that the two conditions hold for $(u_1^{[1]}, \cdots, u_m^{[1]})$ and $(X_1[J^1], \cdots, X_m[J^1])$. Similarly, in L_2, there is a tuple $(u_1^{[2]}, \cdots, u_m^{[2]})$ that partitions jobs in J^2 such that the two conditions hold for $(u_1^{[2]}, \cdots, u_m^{[2]})$ and $(X_1[J^2], \cdots, X_m[J^2])$.

Consider the tuple$(u_1^{\#}, \cdots, u_m^{\#})$ obtained by merging $(u_1^{[1]}, \cdots, u_m^{[1]})$ and $(u_1^{[2]}, \cdots, u_m^{[2]})$. If it is not in the list returned by Sub-Algorithm, then there must exist a tuple $(u_1, \cdots, u_m)$ that is in the list returned by Sub-Algorithm and is in the same region as $(u_1^{\#}, \cdots, u_m^{\#})$. Furthermore, according to the Tuple-Prune process, we must have $P(u_m) \leq P(u_m^{\#})$ and $P(u_j) \leq f \cdot P(u_j^{\#})$ for $1 \leq j \leq m-1$. By using the inductive hypothesis, we have for $1 \leq j \leq m-1$,

$$\begin{aligned} P(u_j) \leq f P(u_j^{\#}) &= f(P(u_j^{[1]}) + P(u_j^{[2]})) \\ &\leq f(f^{i-1} \cdot P(X_j[J^1]) + f^{i-1} \cdot P(X_j[J^2])) \\ &= f^i \cdot (P(X_j[J^1]) + P(X_j[J^2])) = f^i \cdot P(X_j) \ , \end{aligned}$$

and

$$\begin{aligned} P(u_m) &\leq P(u_m^{\#}) = P(u_m^{[1]}) + P(u_m^{[2]}) \\ &\leq P(X_m[J^1]) + P(X_m[J^2]) \leq P(X_m). \end{aligned}$$

The lemma is then proved. □

For the time complexity, one can easily show that the running time of the Sub-Algorithm(I, f) is $O(mn(\log_f P_{sum})^{2m})$.

Theorem 2. *For any function $\alpha(n)$ and $0 < \epsilon < 1$, the Main-Algorithm returns a $(1+\alpha(n)+\epsilon)$-approximation for the scheduling problem $P_{0,m}, \alpha(n) \mid r-a \mid C_{max}$ and the algorithm runs in $O(mn(\frac{\log P_{sum}}{\epsilon})^{2m})$-time.*

Proof. Let I be any instance of the problem $P_{0,m}, \alpha(n) \mid r-a \mid C_{max}$. Assume that the optimal schedule for I is $S^* = (X_1^*, \cdots, X_m^*)$. For all $1 \le i \le m$, let $J_{[i^*]}$ be the job with the largest completion time on M_i in S^*.

We apply the Main-Algorithm with $f = (1 + \frac{\epsilon}{2\log n})$ to the instance I. By Lemma 1, we know that there is a tuple $S = (X_1, \cdots, X_m)$ returned by the Sub-Algorithm, such that (1)$P(X_j) \le f^i \cdot P(X_j^*)$, $1 \le j \le m-1$ and (2) $P(X_m) \le P(X_m^*)$.

Use the two mathematical facts, (a) For any $y \ge 1$, $(1 + \frac{1}{y})^y < e$, where e is the base of the natural logarithm; (b)For $0 < x < 1$, $e^x < 1 + x + x^2$, we get $f^i = (1 + \frac{\epsilon}{2\log n})^{\log n} < (1+\epsilon)$. Thus $P(X_j) \le (1+\epsilon)P(X_j^*)$ for $1 \le j \le m-1$.

Let $J_{[i]}, i = 1, \cdots, m$, be the jobs with the largest completion time on M_i in S. Then we must have $C_{[m]}(S) \le C_{[(m)^*]}(S^*)$ since $P(X_m) \le P(X_m^*)$. Let $\overline{A}_i$ be the total length of the unavailable intervals on machine M_i, then by assumption we have $\overline{A}_i \le \alpha(n)P_{sum}/m \le \alpha(n)C^*_{\max}$. For $i = 1, \ldots m-1$, we have

$$C_{[i]}(S) \le P(X_i) + \overline{A}_i \le (1+\epsilon)P(X_i^*) + \overline{A}_i \le (1+\epsilon)C_{[i^*]}(S^*) + \alpha(n)C^*_{\max} \ .$$

Therefore, the makespan of S is

$$\begin{aligned} C_{\max}(S) &\le \max\Big(\max_{1\le i\le m-1}(1+\epsilon)C_{[i^*]}(S^*) + \alpha(n)C^*_{\max}, C_{[m]}(S)\Big) \\ &\le \big((1+\epsilon) + \alpha(n)\big)C^*_{\max} \ . \end{aligned}$$

Let S' be the schedule returned by the Main-Algorithm, then $C_{\max}(S') \le C_{\max}(S) \le ((1+\epsilon) + \alpha(n))C^*_{\max}$.

For the running time, plugging $f = 1 + \frac{\epsilon}{2\log n}$ into the running time of Sub-Algorithm and using the fact that

$$\log_f P_{sum} = \frac{\ln P_{sum}}{\ln f} \text{ and } \frac{x}{2} \le \ln(1+x) < x, \text{ for } 0 < x < 1,$$

we get the running time

$$O\big(mn(\frac{1}{\epsilon}\log n \log P_{sum})^m\big).$$

The lemma is then proved. □

3.2 Lower Bounds of Approximation

Define the k-partition problem to be the problem of partitioning a set of integers $a_1, \cdots, a_n$ into k subsets $X_1, \cdots, X_k$ so that $\sum_{a_j \in X_i} a_j = A$ for all $1 \leq i \leq k$, where $A = \frac{1}{k}\sum_{i=1}^{n} a_i$. Without loss of generality, we assume that $a_i > 0$ and A is an integer. It is obvious that for any fixed integer $k > 1$, k-partition problem is NP-complete. Furthermore, one can reduce the m-partition problem to the problem of finding a $(1+\alpha(n)$ -approximation for the scheduling problem $P_{0,m}, \alpha(n) \mid r - a \mid C_{max}$. Therefore, we have the following theorem.

Theorem 3. *For each function $\alpha(n)$, the problem $P_{0,m}, \alpha(n) \mid r - a \mid C_{max}$ has no polynomial time $(1 + \alpha(n) - o(1))$-approximation unless $P = NP$.*

Theorems 2 and 3 together imply the following dense hierarchy for polynomial time approximation for NP-hard optimization problems.

Theorem 4. *For every constant $c > 1$ and any small positive δ, there exists a NP-hard problem that has a polynomial time c-approximation, but has no polynomial time $c - \delta$-approximation unless P=NP.*

4 Conclusions

In this paper, we study two problems: $P_{m_1 \geq 1, m_2} \mid r - a \mid C_{max}$ and $P_{0,m}, \alpha(n) \mid r - a \mid C_{max}$. For the former problem we show that the PTAS for Multiple subset sum problem given by Kellerer [3] can be applied to obtain a PTAS. For the latter problem, we show that there is a $(1 + \alpha(n) + \epsilon)-$approximation algorithm for any constant $0 < \epsilon < 1$, and we show that there does not exist any polynomial time $(1 + \alpha(n) - o(1))-$approximation unless P=NP. Thus we derive a tight polynomial approximation algorithm for this problem.

References

1. Baewicz, J., Drozdowski, M., Formanowicz, P., Kubiak, W., Schmidt, G.: Scheduling Preemptable Tasks on Parallel Processors with Limited Availability. Parallel Computing 26(9), 1195–1211 (2000)
2. Breit, J., Schmidt, G., Strusevich, V.A.: Non-Preemptive Two-Machine Open Shop Scheduling with Non-Availability Constraints. Mathematical Methods of Operations Research 57, 217–234 (2003)
3. Caprara, A., Kellerer, H., Pferschy, U.: A PTAS for the Multiple Subset Sum Problem with Different Knapsack Capacities. Information Processing Letter 73(3-4), 111–118 (2000)
4. Chen, B., Potts, C., Woeginger, G.: A Review of Machine Scheduling: Complexity, Algorithms and Approximability. In: Du, D.Z., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, pp. 21–169. Kluwer, Boston (1998)
5. Diedrich, F., Jansen, K., Pascual, F., Trystram, D.: Approximation Algorithms for Scheduling with Reservations. In: Aluru, S., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2007. LNCS, vol. 4873, pp. 297–307. Springer, Heidelberg (2007)

6. Fu, B., Huo, Y., Zhao, H.: Exponential Inapproximability and FPTAS for Scheduling with Availability Constraints. Theoretical Computer Science (to appear)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, CA (1979)
8. Hochbaum, D.S., Shmoys, D.B.: Using Dual Approximation Algorithms for Scheduling Problems: Practical and Theoretical Results. Journal of ACM 34, 144–162 (1987)
9. Hwang, H.-C., Lee, K., Chang, S.Y.: The Effect of Machine Availability on the Worst-Case Performance of LPT. Disc. App. Math. 148(1), 49–61 (2005)
10. Kellerer, H.: Algorithm for Multiprocessor Scheduling with Machine Release Time. IIE Transactions 30, 991–999 (1998)
11. Kubiak, W., Blazewic, J., Formanowicz, P., Breit, J., Schmidt, G.: Two-machine Flow Shops with Limited Machine Availability. European Journal of Operational Research 136, 528–540 (2002)
12. Kubzin, M.A., Potts, C.N., Strusevich, V.A.: Approximation Results for Flow Shop Scheduling Problems with Machine Availability Constraints. Computers & Operations Research 36(2), 379–390 (2009)
13. Lee, C.Y.: Machine Scheduling with Availability Constraint. Journal of Global Optimization 9, 395–416 (1996)
14. Lee, C.Y.: Machine Scheduling with Availability Constraints. In: Leung, J.Y.-T. (ed.) Handboof of Scheduing, pp. 22.1–22.13. CRC Press, Boca Raton (2004)
15. Lee, C.-Y.: Parallel Machines Scheduling with Non-Simultaneous Machine Available Time. Discrete Applied Mathematics 30, 53–61 (1991)
16. Liao, L.-W., Sheen, G.-J.: Parallel Machine Scheduling with Machine Availability and Eligibility Constraints. European Journal of Operational Research 184(2), 458–467 (2008)
17. Saidy, H., Taghvi-Fard, M.: Study of Scheduling Problems with Machine Availability Constraint. Journal of Industrial and Systems Engineering 1(4), 360–383 (2008)
18. Scharbrodt, M., Steger, A., Weisser, H.: Approximation of Scheduling with Fixed Jobs. Journal of Scheduling 2, 267–284 (1999)
19. Schmidt, G.: Scheduling with Limited Machine Availability. European Journal of Operational Research 121, 1–15 (2000)
20. Sheen, G.-J., Liao, L.-W.: Scheduling Machine-Dependent Jobs to Minimize Lateness on Machines with Identical Speed under Availability Constraints. Computers & Operations Research 34(8), 2266–2278 (2007)
21. Xie, J., Wang, X.: Complexity and Algorithms for Two-Stage Flexible Flowshop Scheduling with Availability Constraints. Computers & Mathematics with Applications 50(10-12), 1629–1638 (2005)

A Mathematical Programming Approach for Online Hierarchical Scheduling

Zhiyi Tan* and An Zhang

Department of Mathematics, State Key Lab of CAD & CG
Zhejiang University, Hangzhou 310027, China
tanzy@zju.edu.cn

Abstract. This paper considers an online hierarchical scheduling problem on parallel identical machines. We are given a set of m machines and a sequence of jobs. Each machine has a different hierarchy, and each job also has a hierarchy associated with it. A job can be assigned to a machine only if its hierarchy is no less than that of the machine. The objective is to minimize makespan. Two models are studied in the paper. For the fractional model, we present an improved algorithm and lower bounds. Both algorithm and lower bounds are based on solutions of mathematical programming. For any given m, our algorithm is optimal by numerical calculation. For the integral model, we present both a general algorithm for any m, and an improved algorithm with better competitive ratios of 2.333 and 2.610 for $m = 4$ and 5, respectively.

1 Introduction

In this paper, we considers an online hierarchical scheduling problem on parallel machines. We are given a set of m parallel machines $\{M_1, M_2, \cdots, M_m\}$. Machine M_i has hierarchy i and speed s_i, $i = 1, 2, \cdots, m$. A sequence of jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ arrive one by one over list. Job J_j has hierarchy $g_j \in \{1, 2, \cdots, m\}$ and size p_j, $j = 1, 2, \cdots, n$. Job J_j can be assigned to the machine M_i only if $g_j \geq i$, and the time needed for M_i to process J_j is p_j/s_i. The objective is to minimize makespan, i.e., the maximum load of all machines. Here the *load* of a machine is the completion time of any part of jobs assigned to it.

The performance of an algorithm A for online problem is often evaluated by its competitive ratio, which is defined as the smallest number γ such that for any job sequence $\mathcal{J}$, $C^A(\mathcal{J}) \leq \gamma C^*(\mathcal{J})$, where $C^A(\mathcal{J})$ (or in short C^A) denotes the makespan produced by A and $C^*(\mathcal{J})$ (or in short C^*) denotes the optimal makespan in an offline version. An online problem has a *lower bound* ρ if no online algorithm has a competitive ratio smaller than ρ. An online algorithm is called *optimal* if its competitive ratio matches the lower bound.

The online hierarchical scheduling problem was first studied by [3], and has many applications came from service industry, computer system, hierarchical

* Corresponding author. Supported by Natural Science Foundation of China (10671177, 60021201) and Zhejiang Provincial Natural Science Foundation of China (Y607079).

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 438–450, 2009.

databases, etc. Two models have been studied in the literature. The first is *fractional* model, where each job can be arbitrarily split between the machines, and parts of the same job can be processed on different machines in parallel. The second is *integral* model, where each job should be assigned completely to one machine. For the fractional model on *identical* machines, where all machines have the same speed 1, Bar-Noy et al. [3] presented a e-competitive algorithm. The algorithm can be modified to solve the integral model with competitive ratio $e+1$ (see also [5]). They also showed a lower bound e for both fractional and integral model when the number of machines tends to infinity.

When the number of machines is small, optimal algorithms with better competitive ratios can be obtained, even when machines have different speeds. Park et al. [11] and Jiang et al. [9] independently presented an optimal algorithm with a competitive ratio of $5/3$ for the integral model on two identical machines. When two machines have different speeds, Chassid and Epstein [4] proved the optimal algorithm has a competitive ratio $\frac{1+2s+s^2}{1+s+s^2}$ for the fractional model, Tan and Zhang [12] proved the optimal algorithm has a competitive ratio of

$$\begin{cases} \min\{1+s, \frac{2+2s+s^2}{1+s+s^2}\}, 0 < s \leq 1, \\ \\ \min\{\frac{1+s}{s}, \frac{1+3s+s^2}{1+s+s^2}\}, \quad 1 \leq s < \infty, \end{cases}$$

for the integral model, here $s = \frac{s_2}{s_1}$ is the ratio of speed of two machines. Recently, Zhang et al. [13] proposed an algorithm with a competitive ratio of 2 for the integral model on three identical machines, and showed that this ratio is optimal.

A strongly related problem is online hierarchical scheduling problem on m identical machines with two hierarchy. In this problem, machines and jobs have hierarchy either *high* or *low*. Jobs with high hierarchy can only be assigned to the machine with high hierarchy, while jobs with low hierarchy can be assigned to all machines. Jiang [10] proposed an online algorithm with a competitive ratio of $(12+4\sqrt{2})/7 \approx 2.522$ for the integral model. The result was improved to

$$1 + \frac{m^2 - m}{m^2 - km + k^2} < \frac{7}{3}$$

by Zhang et al. [14], where k is the number of machines with high hierarchy. They also proved a lower bound of 2 when $k \geq 3$ and $m \geq \frac{3}{2}(k+1)$. For the fractional model, Zhang et al. [13] proposed an optimal algorithm with competitive ratio $m^2/(m^2 - km + k^2)$.

A more general problem is the online *restricted assignment* model [2], where each job may be processed on a subset of the machines. Azar et al. presented an online algorithm with a competitive ratio of $\log_2 2m$ for any machine number m. On the contrary, if all jobs have the same hierarchy, online hierarchical scheduling reduces to the classical online scheduling problem, which has been well studied during the last half century ([7,1,6]).

In this paper, we consider both fractional and integral models on identical machines. For the fractional model, we present an online algorithm and lower

bounds. Both algorithm and lower bounds are based on solutions of mathematical programming. For any given m, our algorithm is better than that in [3] and is optimal by numerical calculation. For the integral model, we first use a method similar as that in [3] to obtain a general algorithm for any m with better competitive ratio. Then an improved algorithms are given with competitive ratios 2.333 and 2.610 for $m = 4$ and 5, respectively.

The rest of the paper is organized as follows. Section 2 gives some useful lemmas. Section 3 considers online algorithm and lower bounds for the fractional model. Section 4 is dedicated to the integral model.

2 Preliminaries

Denote by T_i the total size of jobs with hierarchy i, $i = 1, \cdots, m$. We have the following results about the optimal makespan of two models, respectively.

Lemma 1. [3] *For the fractional model, the optimal makespan*

$$C_f^* = \max_{i=1,\cdots,m} \frac{1}{i} \sum_{l=1}^{i} T_l.$$

For the integral model, the optimal makespan

$$C_g^* \geq \max \left\{ \max_{i=1,\cdots,m} \frac{1}{i} \sum_{l=1}^{i} T_l, \max_{j=1,\cdots,n} p_j \right\}.$$

The following technical lemma will be used frequently in the rest of the paper.

Lemma 2. *Let* $\mathbf{1} = (1, 1, \cdots, 1)^T$, $\mathbf{x} = (x_1, x_2, \cdots, x_q)^T$ *be* $q \times 1$ *matrices and* $\mathbf{c} = (c_1, c_2, \cdots, c_q)$ *be a* $1 \times q$ *matrix.* $\mathbf{A} = (a_{ij})_{q\times q}$ *is an invertible matrix, and the* i*-th row vector of* $\mathbf{A}$ *is denoted as* α_i*. If* $\mathbf{cA^{-1}} \geq \mathbf{0}$*, then for any* $\mathbf{x}$

$$\mathbf{cx} \leq (\mathbf{cA^{-1}1}) \max\{\alpha_1\mathbf{x}, \alpha_2\mathbf{x}, \cdots, \alpha_q\mathbf{x}\}.$$

Proof. As $\mathbf{A}$ is invertible, it's clear that $\mathbf{cx} = (\mathbf{cA^{-1}})(\mathbf{Ax})$. Moreover, by $\mathbf{cA^{-1}} \geq \mathbf{0}$, we can conclude that

$$\mathbf{cx} = (\mathbf{cA^{-1}})(\alpha_1\mathbf{x}, \alpha_2\mathbf{x}, \cdots, \alpha_q\mathbf{x})^T \leq (\mathbf{cA^{-1}1}) \max\{\alpha_1\mathbf{x}, \alpha_2\mathbf{x}, \cdots, \alpha_q\mathbf{x}\}. \qquad \square$$

3 Fractional Model

3.1 LP-Based Algorithm

In this subsection, we give an online algorithm for the fractional model. The algorithm is based on an optimal solution of a linear programming, and performs better than that of Bar-Noy et al. [3] for any given m. Besides, the algorithm has very simple structure. The proportion of jobs assigned to each permitted

machine only depends on the hierarchy of the job and machine, regardless the size of the job.

For any fixed $m \geq 2$, consider the following linear programming:

$$
\begin{aligned}
& \min \gamma \\
& \text{s.t.} \sum_{i=1}^{j} x_{ij} = 1, \quad j = 1, \cdots, m, && (1) \\
\text{LP}(m) \qquad & i x_{ii} + \sum_{j=i+1}^{m} x_{ij} \leq \gamma, \quad i = 1, \cdots, m, && (2) \\
& x_{ij} \geq x_{i,j+1}, \quad j = i, \cdots, m-1, \quad i = 1, \cdots, m, && (3) \\
& x_{ij} \geq 0, \quad j = i, \cdots, m, \quad i = 1, \cdots, m.
\end{aligned}
$$

Note that LP(m) has $1 + (m^2 + m)/2$ nonnegative decision variables and $(m^2 + 3m)/2$ constraints. Since

$$\{x_{ii} = 1, i = 1, 2, \cdots, m, x_{ij} = 0, i < j, i, j = 1, 2, \cdots, m, \gamma = m\}$$

is clearly a feasible solution of $LP(m)$ with objective value m, LP(m) always has optimal solution with bounded objective value. Denote by

$$\{x_{ij}^{(m)}, i \leq j, i, j = 1, 2, \cdots, m, \gamma^{(m)}\}$$

the optimal solution of $LP(m)$ with optimal objective value $\gamma^{(m)}$.

Algorithm *LP*

For the fractional model of online hierarchical scheduling on m identical machines, when a job J_k with size p_k and hierarchy $g_k = j$ arrives, assign part of J_k with size of $x_{ij}^{(m)} p_k$ to M_i, $i = 1, \cdots, j$.

Theorem 1. *For the fractional model of online hierarchical scheduling on m identical machines, algorithm LP has a competitive ratio of $\gamma^{(m)}$.*

Proof. Note that (1) ensures the feasibility of algorithm LP, since parts of J_k are assigned to permitted machines $M_1, \cdots M_j$, and total size of parts assigned equals

$$\sum_{i=1}^{j} x_{ij}^{(m)} p_k = p_k \sum_{i=1}^{j} x_{ij}^{(m)} = p_k.$$

Let $\overline{L}_i$ be the load of M_i when algorithm terminates, $i = 1, \cdots, m$. Since only jobs of hierarchy j, $j \geq i$ will be assigned to M_i, and the proportion of jobs assigned to each machine is independent of its size, we have

$$\overline{L}_i = \sum_{j=i}^{m} x_{ij}^{(m)} T_j, i = 1, \cdots, m. \qquad (4)$$

Suppose $C^{LP} = \overline{L}_i$ for some i. Let $\mathbf{x}_{(i)} = (T_i, T_{i+1}, \cdots, T_m)^T$ and $\mathbf{c}_{(i)} = (x_{ii}^{(m)}, x_{i,i+1}^{(m)}, \cdots, x_{im}^{(m)})$, then $C^{LP} = \mathbf{c}_{(i)}\mathbf{x}_{(i)}$. Let

$$\mathbf{A}_{(i)} = \begin{pmatrix} \frac{1}{i} & 0 & 0 & \cdots & 0 \\ \frac{1}{i+1} & \frac{1}{i+1} & 0 & \cdots & 0 \\ & \cdots & & \cdots & \\ \frac{1}{m} & \frac{1}{m} & \frac{1}{m} & \cdots & \frac{1}{m} \end{pmatrix}$$

be an $(m-i+1) \times (m-i+1)$ matrix, and the inventory matrix of $\mathbf{A}_{(i)}$ is

$$\mathbf{A}_{(i)}^{-1} = \begin{pmatrix} i & 0 & 0 & \cdots & 0 & 0 \\ -i & i+1 & 0 & \cdots & 0 & 0 \\ 0 & -(i+1) & i+2 & \cdots & 0 & 0 \\ & \cdots & & \cdots & \cdots & \\ 0 & 0 & 0 & \cdots & m-1 & 0 \\ 0 & 0 & 0 & \cdots & -(m-1) & m \end{pmatrix}.$$

Since

$$\mathbf{c}_{(i)}\mathbf{A}_{(i)}^{-1} = (i(x_{ii}^{(m)} - x_{i,i+1}^{(m)}), (i+1)(x_{i,i+1}^{(m)} - x_{i,i+2}^{(m)}), \cdots, (m-1)(x_{i,m-1}^{(m)} - x_{im}^{(m)}), m x_{im}^{(m)}),$$

we have $\mathbf{c}_{(i)}\mathbf{A}_{(i)}^{-1} \geq \mathbf{0}$ by constraint (3) of LP(m). On the other hand, by Lemma 1, we have

$$C_f^* = \max_{j=1,\cdots,m} \frac{1}{j} \sum_{l=1}^{j} T_l \geq \max_{j=i,\cdots,m} \frac{1}{j} \sum_{l=i}^{j} T_l = \max_{j=1,\cdots,m-i+1} \alpha_{ij}\mathbf{x}_{(i)},$$

where α_{ij} is the j-th row vector of $\mathbf{A}_{(i)}$. Hence, by Lemma 2, we have

$$C^{LP} = \mathbf{c}_{(i)}\mathbf{x}_{(i)} \leq \left(\mathbf{c}_{(i)}\mathbf{A}_{(i)}^{-1}\mathbf{1}\right) \max_{j=1,\cdots,m-i+1} \alpha_{ij}\mathbf{x}_{(i)} \leq \left(\mathbf{c}_{(i)}\mathbf{A}_{(i)}^{-1}\mathbf{1}\right) C_f^*.$$

Since for any i,

$$\mathbf{c}_{(i)}\mathbf{A}_{(i)}^{-1}\mathbf{1} = i x_{ii}^{(m)} + x_{i,i+1}^{(m)} + \cdots + x_{im}^{(m)} \leq \gamma^{(m)}$$

by constraint (2) of LP(m), we have $C^{LP} \leq \gamma^{(m)} C_f^*$, and thus the competitive ratio of LP is $\gamma^{(m)}$. □

3.2 Lower Bounds

Theorem 2. *Any algorithm for the fractional model of online hierarchical scheduling on $m \geq 2$ identical machines has a competitive ratio of at least $\rho^{(m)}$, where $\rho^{(m)}$ is the optimal objective value of the following nonlinear programming NLP(m):*

$$\begin{aligned} & \max \rho \\ \text{NLP}(m) \quad & \text{s.t.} \quad p_i + q_{i+1} - q_i \geq \rho \max_{j=i,\cdots,m} \frac{1}{j} \sum_{l=i}^{j} p_l, i = 1, \cdots, m, \qquad (5) \\ & \rho \geq 1, \quad p_i \geq 0, \quad i = 1, \cdots m, \\ & q_1 = q_{m+1} = 0, \quad q_i \geq 0, \quad i = 2, \cdots, m. \end{aligned}$$

Proof. Since

$$\{p_i = 1, \quad i = 1, \cdots, m, \quad q_i = 0, \quad i = 1, \cdots, m+1, \quad \rho = 1\}$$

is a feasible solution of $NLP(m)$, the feasible region of NLP(m) will not be empty. Denote by

$$\{p_i^{(m)}, \ i = 1, \cdots, m, q_i^{(m)}, \ i = 1, \cdots, m+1, \rho^{(m)}\}$$

an optimal solution of $NLP(m)$. To derive the lower bound for m machines, consider the job sequence with m jobs. The i-th job has size $p_{m-i+1}^{(m)}$ and hierarchy $m-i+1$, $i = 1, \cdots, m$. By Lemma 1, the optimal makespan of the first i jobs is

$$\begin{aligned} C_f^* &= \max_{k=1,\cdots,i} \frac{1}{m-k+1} \sum_{l=m-i+1}^{m-k+1} p_l^{(m)} \\ &= \max_{j=m-i+1,\cdots,m} \frac{1}{j} \sum_{l=m-i+1}^{j} p_l^{(m)}, \quad i = 1, \cdots, m. \end{aligned} \tag{6}$$

We will prove by induction that for any i, $1 \le i \le m-1$, in order to be $\rho^{(m)}$-competitive, A must assign parts of the first i jobs with total size more than $q_{m-i+1}^{(m)}$ to the first $m-i$ machines $M_1, \cdots, M_{m-i}$.

If any algorithm A assigns part of the first job with size no more than $q_m^{(m)}$ to all machines except M_m, then the sequence terminates. The load of M_m should be no less than $p_m^{(m)} - q_m^{(m)}$. Thus

$$\frac{C^A}{C_f^*} \ge \frac{p_m^{(m)} - q_m^{(m)}}{\frac{p_m^{(m)}}{m}} \ge \rho^{(m)},$$

where the last inequality is due to (5). The claim is true for $i = 1$.

Suppose the claim is true for i, then the $(i+1)$-th job with size $p_{m-i}^{(m)}$ and hierarchy $m-i$ comes. Parts of the first i jobs that should be assigned to the first $m-i$ machines and the $(i+1)$-th job, has a total size of $p_{m-i}^{(m)} + q_{m-i+1}^{(m)}$. If total size assigned to to the first $m-i-1$ machines $M_1, \cdots, M_{m-i-1}$ is no more than $q_{m-i}^{(m)}$, then the sequence terminates. The load of M_{m-i} should be no less than $p_{m-i}^{(m)} + q_{m-i+1}^{(m)} - q_{m-i}^{(m)}$. Thus by (5) and (6),

$$C^A \ge p_{m-i}^{(m)} + q_{m-i+1}^{(m)} - q_{m-i}^{(m)} \ge \rho^{(m)} \max_{j=m-i,\cdots,m} \left(\frac{1}{j} \sum_{l=m-i}^{j} p_l^{(m)} \right) = \rho^{(m)} C_f^*.$$

Hence, A must assign parts of the first $i+1$ jobs with total size more than $q_{m-i}^{(m)}$ to the first $m-i-1$ machines $M_1, \cdots, M_{m-i-1}$. The claim is true for $i+1$.

When the last job with size $p_1^{(m)}$ and hierarchy 1 comes, total size of parts of all jobs should be assigned to M_1 is at least $p_1^{(m)} + q_2^{(m)}$. Hence

$$C^A \geq p_1^{(m)} + q_2^{(m)} \geq \rho^{(m)} \max_{j=1,\cdots,m} \left\{ \frac{1}{j} \sum_{l=1}^{j} p_l^{(m)} \right\} = \rho^{(m)} C_f^*$$

by (5) and (6), the desired results follow. □

3.3 Discussion on Optimality of *LP*

For any fixed $m \geq 2$, $LP(m)$ and $NLP(m)$ can be solved numerically. For example, the optimal solutions of $LP(4)$ and $NLP(4)$ are

$$\left\{ x_{11}^{(4)} = 1, x_{12}^{(4)} = \frac{14}{27}, x_{13}^{(4)} = \frac{1}{9}, x_{14}^{(4)} = 0, x_{22}^{(4)} = x_{23}^{(4)} = \frac{13}{27}, x_{24}^{(4)} = \frac{5}{27}, \right.$$

$$\left. x_{33}^{(4)} = x_{34}^{(4)} = x_{44}^{(4)} = \frac{11}{27}, \gamma^{(4)} = \frac{44}{27} \right\}$$

and

$$\left\{ p_1^{(4)} = p_2^{(4)} = p_3^{(4)} = \frac{3}{2}, p_4^{(4)} = 1, q_2^{(4)} = \frac{17}{18}, q_3^{(4)} = \frac{29}{27}, q_4^{(4)} = \frac{16}{27}, \rho^{(4)} = \frac{44}{27} \right\}$$

respectively. It is interesting to note that the optimal objective values of $LP(m)$ and $NLP(m)$ always coincide for any m that we have examined by computer, which implies that algorithm LP is optimal for these values of m. Some selected value of m and corresponding optimal bounds are listed in the following table. Moreover, the bounds tends to e when m tends to ∞, which coincides with the result in [3]. Therefore, though we have not give analytical solution of $LP(m)$ or $NLP(m)$, or prove the optimal objective value of $LP(m)$ and $NLP(m)$ are identical theoretically, we still conjecture that LP is an optimal algorithm for any m.

m	2	3	4	5	6	7	8	9	10	20	50	100
bound	$\frac{4}{3}$	$\frac{3}{2}$	$\frac{44}{27}$	$\frac{245}{143}$	$\frac{16}{9}$	$\frac{1071}{586}$	$\frac{1168}{625}$	$\frac{383}{201}$	$\frac{3676}{1899}$	$\frac{74811780}{35531293}$		
$\approx$	1.333	1.500	1.630	1.713	1.778	1.828	1.869	1.905	1.936	2.106	2.265	2.351

4 Integral Model

4.1 De-fractional *LP* Algorithm

In this subsection, we propose an de-fractional LP algorithm for the integral model. The competitive ratio of the algorithm for the integral model is always larger than that of the fractional model by one. The idea was first suggested by Bar-Noy et al. ([3]). However, our algorithm is based on a more effective algorithm LP for the fractional model, and hence it performs better than that in [3] for any given m.

Algorithm De-fractional LP (DF for short)
Denote by L_k^{j-1} the current load of machine M_k just before job J_j arrives, $k = 1, \cdots, m$. Let $\overline{L}_k^j$ be the load of M_k in the schedule generated by algorithm LP for the job sequence containing first j jobs, $1 \le k \le m$. Let $k_j = \max\{k | L_k^{j-1} \le \overline{L}_k^j, \text{for some } 1 \le k \le g_j\}$, assign J_j entirely to M_{k_j}.

The feasibility of algorithm DF is based on the following lemma. We omit the detailed proof due to lack of space.

Lemma 3. *For any j, $j = 1, \cdots, n$,*
(i) $L_1^{j-1} \le \overline{L}_1^j$.
(ii) *Let $B_i^j = \sum_{k=1}^{i} L_k^j$ and $A_i^j = \sum_{k=1}^{i} \overline{L}_k^j$, then $B_i^j \le A_i^j$ for any $i = 1, \cdots, m$.*

Theorem 3. *For the integral model of online hierarchical scheduling problem on m identical machines, algorithm DF has a competitive ratio of $\gamma^{(m)} + 1$.*

Proof. Note that (i) of Lemma 3 implies that k_j always exists for any j, then DF is well-defined. Without loss of generality, suppose the last job J_n determines makespan. Hence, $C^{DF} = L_{k_n}^{n-1} + p_n$. By the definition of k_n, we have $L_{k_n}^{n-1} \le \overline{L}_{k_n}^n$. Combining with Theorem 1 and Lemma 1, it follows that

$$C^{DF} \le \overline{L}_{k_n}^n + p_n \le C^{LP} + p_n \le \gamma^* C_f^* + p_n \le (\gamma^{(m)} + 1) C_g^*. \qquad \square$$

4.2 Improved Algorithm for Small m

Though LP have good performance for the fractional model, the competitive ratio of DF seems a bit large, especially for small m. In fact, current results show that optimal algorithms for the integral model of online hierarchical scheduling on two and three machines have competitive ratios of 5/3 and 2, respectively ([11,9,13]), both smaller than that of DF. Hence, in this subsection, we propose an improved algorithm which has better performance than DF for $m = 4, 5$. Though the algorithm can be used to the cases with more machines as well, the analysis is extremely difficult and it may not beat DF in such cases.

We first generalize result of Lemma 1. Denote by $V_{ji} = \{k | 1 \le k \le j \text{ and } g_k = i\}$ and $T_{ji} = \sum_{l \in V_{ji}} p_l$ for any $i = 1, \cdots, m$. Let

$$LB_j = \max\left\{ \max_{i=1,\cdots,m} \frac{1}{i} \sum_{l=1}^{i} T_{jl}, \max_{l=1,\cdots,j} p_l \right\}. \tag{7}$$

Clearly, LB_j is a nondecreasing function of j.

Lemma 4. *For the job sequence containing first j jobs $J_1, J_2, \cdots, J_j$, the optimal makespan of the integral model is at least LB_j.*

Algorithm *Hierarchial Threshold*$(m; \alpha)$ (HT for short)
Let J_j be the currently arriving job, L_i^{j-1} be the current load of M_i just before J_j arrives, $i = 1, \cdots, m$.

1. If $g_j = 1$ or $g_j = 2$, assign J_j to M_1.
2. If $g_j \geq 4$, then let $I_j = \{i | p_j + L_i^{j-1} \leq (1+\alpha)LB_j \text{ and } 4 \leq i \leq g_j\}$.
   If $I_j \neq \emptyset$, assign J_j to M_k, where $k = \arg\max I_j$. Otherwise, goto Step 3.
3. If $L_2^{j-1} \leq L_3^{j-1}$, assign J_j to M_2. Otherwise, assign J_j to M_3.

In fact, HT use three different methods according to hierarchy of jobs. M_1 is specialized for processing jobs of hierarchy 1 or 2. The load of M_1 will not be too large since the expected competitive ratio is greater than 2. The assignment of jobs with hierarchy no less than 4 is according to dual greedy idea. Jobs are assigned to the machine with hierarchy as large as possible, unless such assignment will either is not permitted due to hierarchy constraint, or will cause the load of that machine exceed a certain threshold, which is determined by Lemma 4. Since jobs with hierarchy 2 are already scheduled on M_1, we use M_2 and M_3 to process jobs with hierarchy 3, and all remaining jobs which can not be assigned to machines with larger hierarchy. Since no hierarchy constraints should be concerned now, we assign these jobs using primal greedy idea. As we will see in the following, combination of these methods makes the loads of all machines as evenly as we can, and greatly simplify case by case analysis.

Theorem 4. *For $m = 4$ and $\alpha = 4/3$, $HT(4;\alpha)$ is no more than $(1+\alpha)$-competitive.*

Theorem 5. *For $m = 5$ and $\alpha \approx 1.610$, which is the smaller positive solution of equation $2x^3 + 11x^2 - 85x + 100 = 0$, $HT(5;\alpha)$ is no more than $(1+\alpha)$-competitive.*

Proof. Without loss of generality, suppose the last job J_n determines makespan. If it is assigned to M_5 or M_4, then clearly we have

$$C^{HT} = p_n + L_i^{n-1} \leq (1+\alpha)LB_n \leq (1+\alpha)C_g^*, \quad i = 4, 5,$$

where the last inequality is due to Lemma 1. If the algorithm assigns J_n to M_1, then we have

$$C^{HT} = L_1^n = T_{n1} + T_{n2} = 2\frac{T_{n1} + T_{n2}}{2} \leq 2C_g^*,$$

since jobs assigned to M_1 are either of hierarchy 1 or 2. Now suppose J_n is assigned to M_2 or M_3 by the algorithm. Since HT always assigns jobs to one of the two machines which has a smaller load, we have

$$C^{HT} \leq \frac{L_2^{n-1} + L_3^{n-1}}{2} + p_n. \tag{8}$$

If there are only jobs with hierarchy 3 assigned to M_2 and M_3, then $T_{n3} \geq L_2^{n-1} + L_3^{n-1} + p_n$. Together with Lemma 1, it follows

$$C_g^* \geq \max\left\{\frac{\sum_{i=1}^3 T_{ni}}{3}, p_n\right\} \geq \max\left\{\frac{L_2^{n-1} + L_3^{n-1} + p_n}{4}, p_n\right\}.$$

Let $\mathbf{c} = (1, 1)$,

$$\mathbf{x} = \left(\frac{L_2^{n-1} + L_3^{n-1}}{2}, p_n\right)^T \text{ and } \mathbf{A} = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ 0 & 1 \end{pmatrix}.$$

By (8), it is easy to verify that

$$C^{HT} \leq \mathbf{cx}, C_g^* \geq \max\{\alpha_\mathbf{1}\mathbf{x}, \alpha_\mathbf{2}\mathbf{x}\}, \mathbf{cA^{-1}} = (1, 1)\begin{pmatrix} 2 & -\frac{1}{2} \\ 0 & 1 \end{pmatrix} = \left(2, \frac{1}{2}\right) \geq \mathbf{0},$$

By Lemma 2, we can get

$$C^{HT} \leq \mathbf{cA^{-1}1}C_g^* = \frac{5}{2}C_g^* < (1 + \alpha)C_g^*.$$

Therefore, we assume that there exist jobs with hierarchy greater than 3 assigned to M_2 or M_3 in the following. Denote by J_j the last one from the sequence which is assigned to M_2 or M_3 and does not have hierarchy 3. In other words, each job assigned to M_2 and M_3 and arrived after J_j must be of hierarchy 3. Hence, due to Lemma 1,

$$C_g^* \geq \frac{\sum_{i=1}^{3} T_{ni}}{3} \geq \frac{T_{n3}}{3} \geq \frac{1}{3}\left(L_2^{n-1} + L_3^{n-1} + p_n\right) - \frac{1}{3}\left(L_2^{j-1} + L_3^{j-1} + p_j\right). \quad (9)$$

We distinguish two cases according to the value of g_j.

Case 1. $g_j = 5$. Since J_j cannot be assigned on M_4 or M_5 by the algorithm, we must have $L_4^{j-1} + p_j > (1 + \alpha)LB_j$ and $L_5^{j-1} + p_j > (1 + \alpha)LB_j$. By (7), we have $LB_j \geq p_j$ and

$$LB_j \geq \frac{\sum_{i=1}^{5} T_{ji}}{5} = \frac{\sum_{i=1}^{5} L_i^{j-1} + p_j}{5} \geq \frac{(L_2^{j-1} + L_3^{j-1} + p_j) + (L_4^{j-1} + L_5^{j-1})}{5}.$$

It follows that $L_4^{j-1} + L_5^{j-1} > 2\alpha LB_j$ and thus

$$L_2^{j-1} + L_3^{j-1} + p_j \leq (5 - 2\alpha)LB_j < \frac{5 - 2\alpha}{2\alpha}\left(L_4^{j-1} + L_5^{j-1}\right).$$

By Lemma 1, we have

$$\begin{aligned} C_g^* &\geq \frac{\sum_{i=1}^{5} T_{ni}}{5} \geq \frac{(L_2^{n-1} + L_3^{n-1}) + (L_4^{j-1} + L_5^{j-1}) + p_n}{5} \\ &\geq \frac{1}{5}p_n + \frac{1}{5}(L_2^{n-1} + L_3^{n-1}) + \frac{2\alpha}{5(5 - 2\alpha)}\left(L_2^{j-1} + L_3^{j-1} + p_j\right) \end{aligned} \quad (10)$$

Let $y_1 = p_n$, $y_2 = (L_2^{n-1} + L_3^{n-1}) - (L_2^{j-1} + L_3^{j-1} + p_j)$ and $y_3 = (L_2^{j-1} + L_3^{j-1} + p_j)$. Combining with Lemma 1 and (9), (10), we have

$$C_g^* \geq \max\left\{y_1, \frac{1}{3}y_1 + \frac{1}{3}y_2, \frac{1}{5}y_1 + \frac{1}{5}y_2 + \frac{1}{5 - 2\alpha}y_3\right\}. \quad (11)$$

Let $\mathbf{c} = (1, \frac{1}{2}, \frac{1}{2})$, $\mathbf{x} = (y_1, y_2, y_3)^T$ and

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5-2\alpha} \end{pmatrix}$$

it can be verified from (8) and (11) that $C^{HT} \leq \mathbf{cx}$, $C_g^* \geq \max\{\alpha_1\mathbf{x}, \alpha_2\mathbf{x}, \alpha_3\mathbf{x}\}$ and

$$\mathbf{cA^{-1}} = \left(1, \frac{1}{2}, \frac{1}{2}\right) \begin{pmatrix} 1 & 0 & 0 \\ -1 & 3 & 0 \\ 0 & \frac{6\alpha}{5} - 3 & 5 - 2\alpha \end{pmatrix} = \left(\frac{1}{2}, \frac{3\alpha}{5}, \frac{5-2\alpha}{2}\right) \geq \mathbf{0}.$$

Thus by Lemma 2, we have $C^{HT} \leq \mathbf{cA^{-1}1}C_g^* = (3 - \frac{2}{5}\alpha)C_g^* < (1+\alpha)C_g^*$.

Case 2. $g_j = 4$. By Algorithm HT and the definition of LB_j, it follows that $p_j + L_4^{j-1} > (1+\alpha)LB_j \geq p_j + \alpha LB_j$ and

$$LB_j \geq \frac{L_2^{j-1} + L_3^{j-1} + L_4^{j-1} + p_j}{5} \geq \frac{L_2^{j-1} + L_3^{j-1} + \alpha LB_j + p_j}{5},$$

which lead to

$$L_4^{j-1} > \alpha LB_j > \frac{\alpha}{5-\alpha}(L_2^{j-1} + L_3^{j-1} + p_j). \tag{12}$$

If no job assigned to M_2 or M_3 before the arrival of J_j has hierarchy 5, we must have $T_{n3} + T_{n4} \geq L_2^{n-1} + L_3^{n-1} + p_n$. By Lemma 1,

$$C_g^* \geq \max\left\{p_n, \frac{\sum_{i=1}^4 T_{ni}}{4}\right\} \geq \max\left\{p_n, \frac{L_2^{n-1} + L_3^{n-1} + p_n}{4}\right\}.$$

By the same analysis as before, we can get $C^{HT} \leq \frac{5}{2}C_g^*$.

Hence, suppose there are some jobs with hierarchy 5 which arrive before J_j and is assigned to M_2 or M_3. Let J_k be the one among them which arrived latest. In other words, all jobs assigned to M_2 or M_3 arrived after J_k must have hierarchy 3 or 4. Thus we have

$$C_g^* \geq \frac{\sum_{i=1}^4 T_{ni}}{4} \geq \frac{(L_2^{n-1} + L_3^{n-1} + p_n) - (L_2^{k-1} + L_3^{k-1} + p_k)}{4}. \tag{13}$$

Note that algorithm prefer to assign jobs with hierarchy 5 to M_5 or M_4, it follows that

$$p_k + L_5^{k-1} > (1+\alpha)LB_k \geq \alpha LB_k + p_k$$

and

$$p_k + L_4^{k-1} > (1+\alpha)LB_k \geq \alpha LB_k + p_k.$$

Hence,

$$LB_k \geq \frac{(L_2^{k-1} + L_3^{k-1} + p_k) + (L_4^{k-1} + L_5^{k-1})}{5} \geq \frac{(L_2^{k-1} + L_3^{k-1} + p_k) + 2\alpha LB_k}{5},$$

which further results in

$$L_5^{k-1} > \alpha LB_k > \frac{\alpha}{5-2\alpha}\left(p_k + L_2^{k-1} + L_3^{k-1}\right).$$

Combining it with Lemma 1 and (12), we have

$$\begin{aligned} C_g^* &\geq \frac{\sum_{i=1}^{5} T_{ni}}{5} \geq \frac{p_n + L_2^{n-1} + L_3^{n-1} + L_4^{j-1} + L_5^{k-1}}{5} \\ &\geq \frac{p_n + L_2^{n-1} + L_3^{n-1}}{5} + \frac{\alpha}{5(5-\alpha)}\left(p_j + L_2^{j-1} + L_3^{j-1}\right) \\ &\quad + \frac{\alpha}{5(5-2\alpha)}\left(p_k + L_2^{k-1} + L_3^{k-1}\right). \end{aligned} \tag{14}$$

Let $z_1 = p_n$, $z_2 = (L_2^{n-1} + L_3^{n-1}) - (p_j + L_2^{j-1} + L_3^{j-1})$, $z_3 = (p_j + L_2^{j-1} + L_3^{j-1}) - (p_k + L_2^{k-1} + L_3^{k-1})$ and $z_4 = p_k + L_2^{k-1} + L_3^{k-1}$. By Lemma 1 and (9), (13) (14), we can conclude

$$C_g^* \geq \max\left\{z_1, \frac{z_1+z_2}{3}, \frac{z_1+z_2+z_3}{4}, \frac{z_1+z_2}{5} + \frac{z_3}{5-\alpha} + \frac{25-5\alpha-\alpha^2}{5(5-\alpha)(5-2\alpha)} z_4\right\}. \tag{15}$$

Let

$$\mathbf{c} = \left(1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right), \mathbf{x} = (z_1, z_2, z_3, z_4)^T, \mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5-\alpha} & \frac{25-5\alpha-\alpha^2}{5(5-\alpha)(5-2\alpha)} \end{pmatrix},$$

it can be verified from (8) and (15) that $C^{HT} \leq \mathbf{cx}$, $C_g^* \geq \max\{\alpha_1\mathbf{x}, \alpha_2\mathbf{x}, \alpha_3\mathbf{x}, \alpha_4\mathbf{x}\}$ and

$$\begin{aligned} \mathbf{cA^{-1}} &= \left(1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right) \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 3 & 0 & 0 \\ 0 & -3 & 4 & 0 \\ 0 & \frac{15\alpha-6\alpha^2}{25-5\alpha-\alpha^2} & \frac{40\alpha-100}{25-5\alpha-\alpha^2} & \frac{125-75\alpha+10\alpha^2}{25-5\alpha-\alpha^2} \end{pmatrix} \\ &= \left(\frac{1}{2}, \frac{3\alpha(5-2\alpha)}{50-10\alpha-2\alpha^2}, \frac{2\alpha(5-\alpha)}{25-5\alpha-\alpha^2}, \frac{125-75\alpha+10\alpha^2}{50-10\alpha-2\alpha^2}\right) \geq \mathbf{0}. \end{aligned}$$

Thus by Lemma 2, we have

$$C^{HT} \leq \mathbf{cA^{-1}1}C_g^* = \frac{150-45\alpha-\alpha^2}{50-10\alpha-2\alpha^2} C_g^* = (1+\alpha)C_g^*,$$

where the last equality holds by the definition of α. □

References

1. Albers, S.: Better Bounds for Online Scheduling. SIAM J. Comput. 29, 459–473 (1999)
2. Azar, Y., Naor, A., Rom, R.: The Competitiveness of On-line Assignments. J. Alg. 18, 221–237 (1995)

3. Bar-Noy, A., Freund, A., Naor, J.: On-line Load Balancing in a Hierarchical Server Topology. SIAM J. Comput. 31, 527–549 (2001)
4. Chassid, O., Epstein, L.: The Hierarchical Model for Load Balancing on Two Machines. J. Comb. Optim. 15, 305–314 (2008)
5. Crescenzi, P., Gambosi, G., Penna, P.: On-line Algorithms for the Channel Assignment Problem in Cellular Networks. Dis. Appl. Math. 137, 237–266 (2004)
6. Fleischer, R., Wahl, M.: On-line Scheduling Revisited. J. Scheduling 3, 343–353 (2000)
7. Graham, R.L.: Bounds for Certain Multiprocessing Anomalies. Bell Sys. Tech. J. 45, 1563–1581 (1966)
8. Hwang, H., Chang, S., Lee, K.: Parallel Machine Scheduling under a Grade of Service Provition. Comput. Oper. Res. 31, 2055–2061 (2004)
9. Jiang, Y.W., He, Y., Tang, C.M.: Optimal Online Algorithms for Scheduling on Two Identical Machines under a Grade of Service. J. Zhejiang Univ. Sci. 7A, 309–314 (2006)
10. Jiang, Y.W.: Online Scheduling on Parallel Machines with Two GoS Levels. J. Comb. Optim. 16, 28–38 (2008)
11. Park, J., Chang, S.Y., Lee, K.: Online and Semi-online Scheduling of Two Machines under a Grade of Service Provision. Oper. Res. Lett. 34, 692–696 (2006)
12. Tan, Z.Y., Zhang, A.: A Note on Hierarchical Scheduling on Two Uniform Machines. J. Comb. Optim. (to appear)
13. Zhang, A., Jiang, Y.W., Tan, Z.Y.: Optimal Algorithms for Online Hierarchical Scheduling on Parallel Machines. Technical report, Zhejiang University (2008)
14. Zhang, A., Jiang, Y.W., Tan, Z.Y.: Online Parallel Machines Scheduling with Two Hierarchies. Technical report, Zhejiang University (2008)

Recoverable Robust Timetables on Trees*

Gianlorenzo D'Angelo[1], Gabriele Di Stefano[1],
Alfredo Navarra[2], and Cristina M. Pinotti[2]

[1] Department of Electrical and Information Engineering, University of L'Aquila
{gianlorenzo.dangelo,gabriele.distefano}@univaq.it
[2] Department of Mathematics and Computer Science, University of Perugia
{navarra,pinotti}@dmi.unipg.it

Abstract. In the context of scheduling and timetabling, we study a challenging combinatorial problem which is very interesting for both practical and theoretical points of view. The motivation behind it is to cope with scheduled activities which might be subject to unavoidable disruptions, such as delays, occurring during the operational phase. The idea is to preventively plan some extra time for the scheduled activities in order to be "prepared" if a delay occurs, and absorb it without the necessity of re-scheduling all the activities from scratch. This realizes the concept of designing *robust timetables*. During the planning phase, one should also consider recovery features that might be applied at runtime if disruptions occur. This leads to the concept of *recoverable robust timetables*. In this new concept, it is assumed that recovery capabilities are given as input along with the possible disruptions that must be considered. The main objective is the minimization of the overall needed time. We show that finding an optimal solution for this problem is NP-hard even though the topology of the network, which models dependencies among activities, is restricted to trees. However, we manage to design a pseudo-polynomial time algorithm based on dynamic programming.

1 Introduction

In many real world applications, the design of a solution is divided in two main phases: a *strategic planning* phase and an *operational planning* phase. The two planning phases differ in the time in which they are applied. The strategic planning phase aims to plan how to optimize the use of the available resources according to some objective function *before the system starts operating*. The operational planning phase aims to have immediate reaction to disturbing events that can occur *when the system is running*. In general, the objectives of strategic and operational planning might be in conflict with each other. As disturbing events are unavoidable in large and complex systems, it is fundamental to understand the interaction between the objectives of the two phases. An example of real world systems, where this interaction is important, is the *timetable planning* in railways systems. It arises in the strategic planning phase, and requires to compute a

* This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 451–462, 2009.

timetable for passenger trains that determines minimal passenger waiting times. However, many disturbing events might occur during the operational phase, and they might completely change the scheduled activities. The main effect of the disturbing events is the arising of delays. The conflicting objectives of strategic against operational planning are evident in timetable optimization. In fact, a train schedule that lets trains sit in stations for some time will not suffer from small delays of arriving trains, because delayed passengers can still catch potential connecting trains. On the other hand, large delays can cause passengers to lose trains and hence imply extra traveling time. The problem of deciding when to guarantee connections from a delayed train to a connecting train is known as *delay management* [2,8,14,15,10,11]. Although its natural formalization, the problem turns out to be very complicated to be optimally solved. In fact, it is NP-hard in the general case, while it is polynomial in some particular cases (see [2,14,15,10,11]).

To cope with the management of delays, we follow the recent *recoverable robustness* approach provided in [1,13], continuing the recent studying in robust optimization. Our aim is the design of timetables in the strategic planning phase in order to be "prepared" to react against possible disruptions. If a disruption (delay) occurs, the designed timetable should guarantee to recover the scheduled events by means of allowed operations represented by given recovery algorithms. Events and dependencies among events are modeled by means of an *event activity network* (see [2,15]). This is a directed graph where the nodes represent events (e.g., arrival or departure of trains) and arcs represent activities occurring between events (e.g., waiting in a train, driving between stations or changing to another train). We assume that only one delay of at most α time might occur at a generic activity of the scheduled event activity network. An activity may absorb the delay if it is associated with a so called *slack time*. A slack time assigned to an activity represents some extra available time that can be used to absorb limited delays. Clearly, if we associate a slack time of at least α to each activity, every delay can be locally absorbed. However, this approach is not practical as the overall duration time of the scheduled events would increase too much. We plan timetables able to absorb the possible occurring delay in a fixed amount of steps, Δ. This means that, if a delay occurs, it is not required that the delay is immediately absorbed (unless $\Delta = 0$) but it can propagate to a limited number of activities in the network. Namely, the propagation might involve at most Δ activities. The objective function is then to minimize the total time required by the events in order to serve all the scheduled activities and to be robust with respect to one possible delay. The challenging combinatorial problem arising by those restrictions is of its own interest. We restrict our attention to event activity networks whose topology is a tree. In fact, in [2], the authors show that the described problem is NP-hard when the event activity network topology is a DAG, and they provide approximation algorithms which cope with the case of $\Delta = 0$. In [3], these algorithms have been extended to $\Delta \geq 0$. In [4], the authors provide polynomial time algorithms for $\Delta \geq 0$ when the network is a path. In [6], practical motivations to the use of trees in real world scenarios have

been provided. In the same paper, the algorithms proposed in [7], which have $O(n^{\Delta+1})$ time complexity, have been applied.

In this paper, we study event activity networks which have a tree topology. Surprisingly, the described problem turns out to be NP-hard even in this restricted case. Then we present algorithmic results. We provide an algorithm that solves the problem for $\Delta \geq 1$ in $O(\Delta^2 n)$ time and $O(\Delta n)$ space where n is the number of events in the input event activity network. While it requires $O(n)$ time and space when $\Delta = 0$. The result implies that the problem can be solved in pseudo-polynomial time. The algorithm exploits the tree topology in order to choose which arc must be associated with some slack time. Intuitively, on trees, we prove that the choice to carefully postpone the assignment of a slack time to descendent activities as much as possible leads to cheapest solutions. Due to space restrictions, omitted proofs can be found in [5].

2 Recoverable Robustness Model

In this section, we summarize the model of recoverable robustness given in [1]. Such a model describes how an optimization problem P can be turned into a *robustness problem* $\mathcal{P}$. Hence, concepts like *robust solution*, *robust algorithm* for $\mathcal{P}$ and *price of robustness* are defined. In the remainder, an optimization problem P is characterized by the following parameters. A set I of instances of P; a function F, that associates to any instance $i \in I$ the set of all feasible solutions for i; and an objective function $f\colon S \to \mathbb{R}$, where $S = \bigcup_{i \in I} F(i)$ is the set of all feasible solutions for P. Without loss of generality, from now on we consider minimization problems. Additional concepts to introduce robustness requirements for a minimization problem P are needed:

- $M : I \to 2^I$ – a *modification* function for instances of P. Let $i \in I$ be the considered input to the problem P. A *disruption* is meant as a modification to the input i. Hence, $M(i)$ represents the set of disruptions of the input of P that can be obtained by applying all possible modifications to I.
- $\mathbb{A}$ – a class of *recovery algorithms* for P. Algorithms in $\mathbb{A}$ represent the capability of recovering against disruptions. An element $A_{rec} \in \mathbb{A}$ works as follows: given $(i, \pi) \in I \times S$, an instance/solution pair for P, and $j \in M(i)$, a disruption of the current instance i, then $A_{rec}(i, \pi, j) = \pi'$, where $\pi' \in F(j)$ represents the recovered solution for P.

Definition 1. *A* recoverable robustness problem $\mathcal{P}$ *is defined by the triple* $(P, M, \mathbb{A})$. *All the recoverable robustness problems form the class* RRP.

Definition 2. *Let* $\mathcal{P} = (P, M, \mathbb{A}) \in$ RRP. *Given an instance* $i \in I$ *for* P, *an element* $\pi \in F(i)$ *is a* feasible solution *for* i *with respect to* $\mathcal{P}$ *if and only if the following relationship holds:* $\exists A_{rec} \in \mathbb{A} : \forall j \in M(i),\ A_{rec}(i, \pi, j) \in F(j)$.

In other words, $\pi \in F(i)$ is feasible for i with respect to $\mathcal{P}$ if it can be *recovered* by applying some algorithm $A_{rec} \in \mathbb{A}$ for each possible disruption $j \in M(i)$. The solution π is called a *robust solution* for i with respect to problem P. The quality of a robust solution is measured by the *price of robustness*.

Definition 3. *Let* $\mathcal{P} = (P, M, \mathbb{A}) \in$ RRP. *A* robust algorithm *for* $\mathcal{P}$ *is an algorithm* A_{rob} *such that, for each* $i \in I$, $A_{rob}(i)$ *is a robust solution for* i *wrt* P.

Definition 4. *Let* $\mathcal{P} \in$ RRP *and let* A_{rob} *be a robust algorithm for* $\mathcal{P}$. *The* price of robustness of A_{rob} *is:* $P_{rob}(\mathcal{P}, A_{rob}) = \max_{i \in I} \left\{ \frac{f(A_{rob}(i))}{\min\{f(\pi):\pi \in F(i)\}} \right\}$. *The* price of robustness of $\mathcal{P}$ *is:* $P_{rob}(\mathcal{P}) = \min\{P_{rob}(\mathcal{P}, A_{rob}) : A_{rob}$ *is robust for* $\mathcal{P}\}$. A_{rob} *is* $\mathcal{P}$-optimal *if* $P_{rob}(\mathcal{P}, A_{rob}) = P_{rob}(\mathcal{P})$. *A robust solution* π *for* $i \in I$ *is* $\mathcal{P}$*-optimal if* $f(\pi) = \min\{f(\pi') \ : \ \pi'$*is feasible for* $i\}$.

3 Robust Timetabling Problem

In this section, we turn a particular timetable problem into a recoverable robustness problem, the *Robust Timetabling* problem ($\mathcal{RTT}$).

Given a DAG $G = (V, A)$, where the nodes represent events and the arcs represent the activities, the *timetabling problem* consists in assigning a time to each event in such a way that all the constraints provided by the set of activities are respected. Specifically, given a function $L : A \to \mathbb{N}$ that assigns the minimal duration time to each activity, a solution $\pi \in \mathbb{R}_{\geq 0}^{|V|}$ for the timetable problem on G is found by assigning a time $\pi(u)$ to each event $u \in V$ such that $\pi(v) - \pi(u) \geq L(a)$, for all $a = (u, v) \in A$.

Given a function $w : V \to \mathbb{R}_{\geq 0}$ that assigns a weight to each event, an optimal solution for the timetabling problem minimizes the total weighted time for all events. Formally, the *timetabling problem* TT is defined as follows.

TT	
GIVEN:	A DAG $G = (V, A)$, functions $L : A \to \mathbb{N}$ and $w : V \to \mathbb{R}_{\geq 0}$.
PROB.:	Find a function $\pi : V \to \mathbb{R}_{\geq 0}$ such that $\pi(v) - \pi(u) \geq L(a)$ for all $a = (u, v) \in A$ and $f(\pi) = \sum_{v \in V} w(v)\pi(v)$ is minimal.

Then, an instance i of TT is specified by a triple (G, L, w), where G is a DAG, L associates a minimal duration time to each activity, and w associates a weight to each event. The set of feasible solutions for i is: $F(i) = \{\pi : \pi(u) \in \mathbb{R}_{\geq 0}, \ \forall u \in V$ and $\pi(v) - \pi(u) \geq L(a), \ \forall a = (u, v) \in A\}$.

A feasible solution for TT may induce a positive *slack time* $s_\pi(a) = \pi(v) - \pi(u) - L(a)$ for each $a \in A$. That is, the planned duration $\pi(v) - \pi(u)$ of an activity $a = (u, v)$ is greater than the minimal duration time $L(a)$.

When in TT the DAG is an out-tree $T = (V, A)$, any feasible solution satisfies, for each $v \in V$, $\pi(v) \geq \pi(r) + \sum_{a \in P(r,v)} L(a)$, where r is the root of T and $P(r, v)$ the directed path from r to v in T. Moreover, without loss of generality, we can fix our attention only on instances of TT with $L(a) = 1$, $\forall a \in A$. Indeed, as proved below, the cost $f(\pi)$ of a feasible solution π for an instance of TT with an arbitrary function L easily derives from the cost $f(\pi')$ of a feasible solution π' for the same instance of TT with $L'(a) = 1$, $\forall a \in A$.

Lemma 1. *Given a tree* $T = (V, A)$ *and an instance* $i = (T, L, w)$ *of* TT, *if* $i' = (T, L', w)$ *where* $L'(a) = 1$, $\forall a \in A$, *then for any feasible solution*

π of i, there exists a feasible solution π' of i' such that $f(\pi) = f(\pi') + \sum_{v \in V} \sum_{a \in P(r,v)} w(v)(L(a) - 1)$.

TT can be solved in linear time by assigning the minimal possible time to each event (i.e. by using the Critical Path Method [12]). However, such a solution cannot always cope with possible delays occurring at running time to the activities. Recovery (on-line) strategies might be necessary. For this reason, let now transform TT into a recoverable robustness problem $\mathcal{RTT} = (TT, M, \mathbb{A})$, according to Section 2. Given an instance $i = (G, L, w)$ for TT, and a constant $\alpha \in G$, we limit the modifications on i by admitting a single delay of at most α time. We model it as an increase on the minimal duration time of the delayed activity. Formally, $M(i)$ is defined as follows: $M(i) = \{(G, L', w) : \exists\, \bar{a} \in A : L(\bar{a}) \leq L'(\bar{a}) \leq L(\bar{a}) + \alpha,\ L'(a) = L(a)\ \forall a \neq \bar{a}\}$.

We define the class of recovery algorithms $\mathbb{A}$ for TT by introducing the concept of *events affected by one delay* as follows.

Definition 5. *Given a DAG $G = (V, A)$, a function $s : A \to \mathbb{R}_{\geq 0}$, and a number $\alpha \in \mathbb{R}_{\geq 0}$, a node x is* α-affected *by $a = (u, v) \in A$ (or a* α-affects *x) if there exists a path $p = (u \equiv v_0, v \equiv v_1, \ldots, v_k \equiv x)$ in G, such that $\sum_{i=1}^{k} s((v_{i-1}, v_i)) < \alpha$. The set of nodes α-affected by an arc $a = (u, v)$ is denoted as* $\mathrm{Aff}(a)$.

In the following, given a feasible solution π for TT, we will use the slack times s_π defined by π as the function s in the previous definition. Thus, an event x is affected by a delay α occurring on the arc $a = (u, v)$ if the sum of the slack times assigned by the function π to the events on the path from u to x is smaller than α. That is, the planned duration of the activities are not able to absorb the delay α, which cannot be hidden from x .

We assume that the recovery capabilities allow us to change the time of at most Δ events. Formally, each algorithm in $\mathbb{A}$ is able to compute a solution $\pi' \in F(j)$ if, for each $a \in A$, $|\mathrm{Aff}(a)| \leq \Delta$, where $\Delta \in \mathbb{N}$. This implies that a robust solution for $\mathcal{RTT}$ must guarantee that, if a delay of at most α time occurs, then it affects at most Δ events. Note that, $\mathcal{RTT}$ only requires to find a feasible solution. Nevertheless, it is worth to find a solution that minimizes the objective function of TT. Moreover, we focus on $\mathcal{RTT}$, where the DAG is an out-tree. The above optimization problem, denoted by $\mathcal{RTT}_{opt}$, is defined as follows:

$\mathcal{RTT}_{opt}$	
GIVEN:	A tree $T = (V, A)$, a function $w : V \to \mathbb{R}_{\geq 0}$, and $\alpha, \Delta \in \mathbb{N}$.
PROB.:	Find a function $\pi : V \to \mathbb{R}_{\geq 0}$ s. t. each arc in A α-affects at most Δ nodes, according to the function $s_\pi : A \to \mathbb{R}_{\geq 0}$ defined as $s_\pi(a = (i, j)) = \pi(j) - \pi(i) - 1$, and s. t. $f(\pi) = \sum_{v \in V} \pi(v) w(v)$ is minimal

In the next lemma, we prove that, when the modifications are confined to a single delay of at most α time, there exists a solution for the $\mathcal{RTT}_{opt}$ problem which assigns only slack times equal to α.

Lemma 2. *Given an instance i of $\mathcal{RTT}$, for each feasible solution π for i there exists a solution π' for i such that $f(\pi') \leq f(\pi)$ and, for each arc $a = (x, y)$, either $\pi'(y) = \pi'(x) + 1$ or $\pi'(y) = \pi'(x) + 1 + \alpha$.*

4 Complexity

We call $\mathcal{RIT}_{dec}$ the decision problem corresponding to $\mathcal{RIT}_{opt}$.

$\mathcal{RIT}_{dec}$	
GIVEN:	A tree $T = (V, A)$, function $w : V \to \mathbb{R}_{\geq 0}$, $\alpha, \Delta \in \mathbb{N}$, and $K' \in \mathbb{R}_{\geq 0}$.
PROB.:	Is there a function $\pi : V \to \mathbb{R}_{\geq 0}$ such that each arc in A α-affects at most Δ nodes, according to the function $s_\pi : A \to \mathbb{R}_{\geq 0}$ defined as $s_\pi(a = (i, j)) = \pi(j) - \pi(i) - 1$, and such that $\sum_{v \in V} \pi(v)w(v) \leq K'$?

In the next theorem, we show that $\mathcal{RIT}_{dec}$ is NP-complete by a transformation from *Knapsack* [9].

Knapsack	
GIVEN:	A finite set U, for each $u \in U$ a size $S(u) \in Z^+$, a value $v(u) \in Z^+$, and positive integers $B, K \in Z^+$.
PROB.:	Is there a subset $U' \subseteq U$ s. t. $\sum_{u \in U'} S(u) \leq B$ and $\sum_{u \in U'} v(u) \geq K$?

Theorem 1. *$\mathcal{RIT}_{dec}$ is NP-complete.*

Proof. A solution for $\mathcal{RIT}_{dec}$ can be verified in polynomial time, as we only need to find out whether there exists a subtree of size bigger than Δ where no slack time α has been added. This can be performed by counting for each node v, how many descending nodes are affected if a delay of α is assumed to occur at the in-arc of x. Starting from the leaves of the tree and moving up until the root, the procedure needs a simple visit of the tree, hence $\mathcal{RIT}_{dec}$ is in NP.

Given an instance I of *Knapsack* where $U = \{u_1, u_2, \ldots, u_{|U|}\}$, we define an instance I' of $\mathcal{RIT}_{dec}$. See Fig. 1 in [5] for a visualization of I'. Without loss of generality, we assume that $S(u_i) \leq B$, for each $u_i \in U$. The set of nodes is made of: nodes r, r' and the sets of nodes $X_i = \{x_i^1, x_i^2, \ldots, x_i^{S(u_i)} \equiv z_i\}$, for each $u_i \in U$. The set of arcs A is made of: arc (r, r'); arcs (r', x_i^1), for each $i = 1, 2, \ldots, |U|$; and for each u_i such that $S(u_i) > 1$, arcs (x_i^j, x_i^{j+1}), for each $i = 1, 2, \ldots, |U|$ and for each $j = 1, 2, \ldots, S(u_i) - 1$.

The weight of each node in V is 0 except for nodes z_i, $i = 1, 2, \ldots, |U|$, where $w(z_i) = v(u_i)$. Finally, $\Delta = B + 1$, $\alpha = 1$ and $K' = \sum_{u_i \in U} v(u_i)(S(u_i) + 2) - K$. It is worth noting that the particular tree construction preserves the size of the *Knapsack* instance, as each path of nodes of the same weight w can be compacted and implicitly be represented by two numbers, namely, the number of nodes of the path and the weight w of each node. This implies that each path $(x_i^1, x_i^2, \ldots, x_i^{S(u_i)-1})$ of our construction will be represented by the pair $(S(u_i) - 1, 0)$. Also the operations performed on such paths do not need the explicit representation of the tree. In particular, the check needed to verify whether a solution is feasible can be done efficiently with respect to the compacted instance.

Now we show that, if there exists a "yes" solution for *Knapsack*, then there exists a "yes" solution for $\mathcal{RIT}_{dec}$.

Given a set $U' \subseteq U$ for I, we define the following solution π for I': $\pi(r) = 0$; $\pi(r') = 1$; $\pi(x_i^1) = 2$ for each $u_i \in U'$ and $\pi(x_i^1) = 3$ for each $u_i \notin U'$; $\pi(x_i^j) = \pi(x_i^1) + j - 1$, for each $i = 1, 2, \ldots, |U|$ and for each $j = 2, 3, \ldots, S(u_i)$.

Note that, for each $u_i \in U'$, $\pi(z_i) = S(u_i) + 1$ while for each $u_i \notin U'$, $\pi(z_i) = S(u_i) + 2$. As the weight of each node in V is 0 except for nodes z_i, then $f(\pi) = \sum_{i=1}^{|U|} \pi(z_i) \cdot w(z_i) = \sum_{u_i \in U'} (S(u_i)+1) \cdot v(u_i) + \sum_{u_i \notin U'} (S(u_i)+2) \cdot v(u_i) = \sum_{u_i \in U} (S(u_i)+2) \cdot v(u_i) - \sum_{u_i \in U'} v(u_i) \leq \sum_{u_i \in U} (S(u_i)+2) \cdot v(u_i) - K = K'$. We have to show that each arc in A α-affects at most Δ nodes. As $\Delta = B+1 > S(u_i)$, for each $u_i \in U$, then all the arcs but (r, r') do not α-affect more than Δ nodes. Moreover, for each $u_i \notin U'$, $\pi(x_i^1) - \pi(r') = 1 + \alpha$. Hence, the arc (r, r') α-affects r' and nodes x_i^j for each $u_i \in U'$ and for each $j = 1, 2, \ldots, S(u_i)$. Then, the overall number of affected nodes is $1 + \sum_{u_i \in U'} S(u_i) \leq 1 + B = \Delta$.

Now we show that, if there exists a "yes" solution for $\mathcal{RIT}_{dec}$, then there exists a "yes" solution for *Knapsack*. Given a "yes" solution π' for I', we define a "yes" solution π that assigns slack times only to arcs (r', x_i^1), $i = 1, 2, \ldots, |U|$ as follows: $\pi(r) = 0$; $\pi(r') = 1$; $\pi(x_i^1) = 3$ for each i such that π' assigns at least a slack time in the path from r' to z_i, i.e. $\pi'(z_i) - \pi'(r') \geq S(u_i) + \alpha$; $\pi(x_i^1) = 2$ for each i such that $\pi'(z_i) - \pi'(r') < S(u_i) + \alpha$; $\pi(x_i^j) = \pi(x_i^1) + j - 1$, for each $i = 1, 2, \ldots, |U|$ and for each $j = 2, 3, \ldots, S(u_i)$. Note that, if π' is a "yes" solution for I' then π is also a "yes" solution for I'. In fact, $\pi(z_i) \leq \pi'(z_i)$ and then $f(\pi) \leq f(\pi') \leq K'$. Moreover, the number of nodes α-affected by (r, r') in π is at less than or equal to the number of nodes α-affected by (r, r') in π'.

We define a solution for I as $U' = \{u_i : \pi(x_i^1) = 2\}$. We have to show that $\sum_{u \in U'} S(u) \leq B$ and $\sum_{u \in U'} v(u) \geq K$. As the number of nodes α-affected by (r, r') in the solution π is less than or equal to Δ, then $\Delta \geq 1 + \sum_{i:\pi(x_i^1)=2} |X_i| = 1 + \sum_{u_i \in U'} S(u_i)$. Hence $\sum_{u_i \in U'} S(u_i) \leq \Delta - 1 = B$. As the weight of each node in V is 0 except for nodes z_i, then $f(\pi) = \sum_{i=1}^{|U|} \pi(z_i) \cdot w(z_i) = \sum_{i:\pi(x_i^1)=2} (S(u_i)+1) \cdot w(z_i) + \sum_{i:\pi(x_i^1)=3} (S(u_i)+2) \cdot w(z_i) = \sum_{i:\pi(x_i^1)=2} (S(u_i)+1) \cdot v(u_i) + \sum_{i:\pi(x_i^1)=3} (S(u_i)+2) \cdot v(u_i) = \sum_{i=1}^{|U|} (S(u_i)+2) \cdot v(u_i) - \sum_{i:\pi(x_i^1)=2} v(u_i) = \sum_{u_i \in U} (S(u_i)+2) \cdot v(u_i) - \sum_{u_i \in U'} v(u_i) \leq K' = \sum_{u_i \in U} (S(u_i)+2) \cdot v(u_i) - K$. Hence $\sum_{u_i \in U'} v(u_i) \geq K$. □

Corollary 1. *$\mathcal{RIT}_{opt}$ and computing $P_{rob}(\mathcal{RIT})$ are NP-hard.*

5 Pseudo-polynomial Time Algorithm

Based on the dynamic programming techniques, in this section we devise a pseudo-polynomial time algorithm for $\mathcal{RIT}_{opt}$.

Let us introduce further notation. Note that, $\mathcal{RIT}$-optimal denote a solution for $\mathcal{RIT}_{opt}$. Let $T = (V, A)$ be an arbitrarily ordered rooted tree, i.e. for each node v we can distinguish its children denoted as $N_o(v)$ as an arbitrarily ordered set $\{v_1, v_2, \ldots, v_{|N_o(v)|}\}$. For an arbitrary subtree $S(v)$ rooted at $v \in V$, let $N_o(S(v))$ denote the set of nodes y such that $(x, y) \in A$, $x \in S(v)$ and $y \notin S(v)$. Clearly, when $S(v) = \{v\}$, $N_o(S(v)) \equiv N_o(v)$. In addition, for each node $v \in T$, let $T(v)$ be the full subtree of T rooted in v and let $T_i(v)$ denote the full subtree of T rooted at v limited to the first (according to the initially chosen order) i children of v. Moreover, recalling that T is a weighted tree, let $c(v) = \sum_{x \in T(v)} w(x)$ be the sum of the weights of the nodes in $T(v)$ and let $|T(v)|$ be the number of

nodes belonging to $T(v)$. Note that, the values $|T(v)|$ and $c(v)$ for all $v \in V$ can be computed in linear time by visiting $T(r)$ where r is the root of T.

Lemma 3. *Given a tree T, consider two feasible solutions π' and π such that, for an arc $\bar{a} = (u, v)$, $s_{\pi'}(\bar{a}) = s_\pi(\bar{a}) + \alpha$ and $s_\pi(a) = s_{\pi'}(a)$, for each $a \neq \bar{a}$. Then, $f(\pi') = f(\pi) - \alpha c(v)$.*

Definition 6. *Given a feasible solution π for the $\mathcal{RIT}$ problem and a node $v \in V$, a* ball $B_\pi(v)$ *is the maximal subtree rooted in v such that each node in $B_\pi(v)$ has its incoming arc a with $s_\pi(a) = 0$.*

In other words, $B_\pi(v)$ represents the set of nodes in solution π which are affected by the delay occurring at $a = (u, v)$. Due to the feasibility of π, no more than Δ nodes can belong to $B_\pi(v)$. Note that, if $s_\pi(a) = \alpha$, then $B_\pi(v)$ is empty.

Lemma 4. *For each instance of $\mathcal{RIT}$, an $\mathcal{RIT}$-optimal solution π is such that, for each $v \in V$, $B_\pi(v)$ cannot be extended by adding any node in $N_o(B_\pi(v))$, while keeping feasibility.*

Lemma 5. *For each instance of $\mathcal{RIT}$, with $\Delta \geq 1$, there exists a $\mathcal{RIT}$-optimal solution such that at most one of two consecutive arcs has a slack time of α.*

Let us now consider an optimal solution π for the $\mathcal{RIT}$ problem on T. Since the root r of T has no incoming arcs, it cannot be affected by any delay. In other words, r can be considered as a node reached by an arc associated with a slack time of α, hence, by Lemma 5, for each arc a outgoing from the root r, $s_\pi(a) = 0$. By Lemma 4, for each $v \in N_o(r)$, π induces a ball $B_\pi(v)$ of size $\min\{|T(v)|, \Delta\}$.

In order to compute an $\mathcal{RIT}$-optimal solution π, by applying dynamic programming techniques, we obtain an algorithm which requires $O(\Delta^2 n)$ time complexity and $O(\Delta n)$ space. The algorithm makes use of two procedures, called `SA-DP` and `BUILD` illustrated in Fig. 2 and 3, respectively. Algorithm `SA-DP` considers an arbitrarily ordered tree T in input and performs a visit of T. It uses for each node $v \in T$ two matrices G_v and SOL_v, both of size $(N_o(v)+1) \times (\Delta+1)$.

Let f^* denote the cost of an $\mathcal{RIT}$-optimal solution on T when $\Delta = 0$, that is, the cost of the solution which associates to each arc a slack time of α. The `SA-DP` algorithm stores in each entry $G_v[i, j]$ the maximum gain with respect to f^* achievable with a solution π by constructing a ball $B_\pi(v)$ of size at most j when considering only the first i children of v, i.e. $|B_\pi(v) \cap T_i(v)| \leq j$. As $G_v[i, j]$ must be the maximum gain, solution π must be optimal with respect to $T_i(v)$. Formally, we denote as $f_i^v(\pi)$ the value of the objective function computed only on the nodes in $T_i(v)$, i.e. $f_i^v(\pi) = \sum_{u \in T_i(v)} \pi(u) w(u)$ and by π^* an $\mathcal{RIT}$-optimal solution on T when $\Delta = 0$. When the objective function is computed on the nodes in the whole $T(v)$, we simply write f^v. Then, $G_v[i, j]$ is defined as: $G_v[i, j] = \max_\pi \{f_i^v(\pi^*) - f_i^v(\pi) \ : \ |B_\pi(v)| \leq j\}$. Note that, nodes not in $T_i(v)$ are not considered in the computation of f_i^v and hence, there always exists a solution π which gives the gain $G_v[i, j]$ and $B_\pi(v) \subseteq T_i(v)$.

In practice, for each node v and each integer $j \leq \Delta$, we have to decide the most profitable way of building a ball of size j, rooted at v. Since any node v

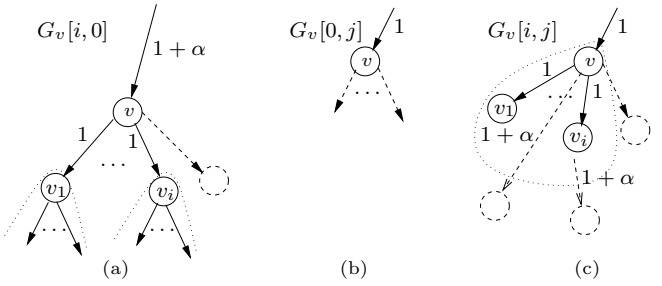


Fig. 1. The three configurations that must be considered when computing matrix G_v

belonging to the ball gains a profit of $\alpha c(v)$, independent of the shape of the ball itself, each ball can be easily partitioned in maximal sub-balls, possibly empty, rooted at the children $v_1, v_2, \ldots, v_{|N_o(v)|}$ of v. Moreover, note that, by Lemma 4, if $j \geq |T(v)|$, the maximum gain is obtained by setting $B_\pi(v) \equiv T(v)$.

The following lemma shows sub-optimality properties of matrices G_v needed by `SA-DP`. For a node v in T, Fig. 1 shows the possible configurations to consider when evaluating $G_v[i,j]$.

Lemma 6. *For each instance of $\mathcal{RTT}$ and for each node v,*

1. $G_v[i,0] = \sum_{\ell=1}^{i} G_{v_\ell}[|N_o(v_\ell)|, \Delta]$, *for* $0 \leq i \leq |N_o(v)|$;
2. $G_v[0,j] = \alpha c(v)$, *for* $1 \leq j \leq \Delta$;
3. $G_v[i,j] = \max_{0 \leq s < j} \{G_v[i-1, j-s] + G_{v_i}[|N_o(v_i)|, s]\}$, *otherwise.*

Proof. 1. By definition, $G_v[i,0] = \max_\pi \{f_i^v(\pi^*) - f_i^v(\pi) \ : \ |B_\pi(v)| = 0\}$. As $|B_\pi(v)| = 0$, by definition of ball, the incoming arc of v has a slack time of α and by Lemma 4, nodes v_ℓ in $N_o(v)$ have a ball of size $\min\{\Delta, T(v_\ell)\}$. It follows that $G_v[i,0] = \sum_{\ell=1}^{i} \max_\pi \{f^{v_\ell}(\pi^*) - f^{v_\ell}(\pi) \ : \ |B_\pi(v_\ell)| \leq \Delta\} = \sum_{\ell=1}^{i} G_{v_\ell}[|N_o(v_e ll)|, \Delta]$.

2. It directly follows from Lemma 3. In fact, as $i = 0$, and $j > 0$, $B_\pi(v)$ is composed only by v. Then, the required solution π differs from π^* only for the slack time removed from the incoming arc to v.

3. By definition, $\max_{0 \leq s < j} \{G_v[i-1, j-s] + G_{v_i}[|N_o(v_i)|, s]\} = \max_{0 \leq s < j} \{$

$$\underbrace{\max_\pi \left\{ f_{i-1}^v(\pi^*) - f_{i-1}^v(\pi) : |B_\pi(v)| \leq j - s \right\}}_{(1)} + \underbrace{\max_\pi \left\{ f^{v_i}(\pi^*) - f^{v_i}(\pi) : |B_\pi(v_i)| \leq s \right\}}_{(2)}.$$

Let us denote as π' and π'' two solutions which maximize (1) and (2), respectively and such that $\pi'(u) = \pi''(u) = \pi^*(u)$, for each $u \notin T(v)$. Note that, the set of nodes where functions f_{i-1}^v and f^{v_i} are defined are disjoint as the former is defined in $T_{i-1}(v)$ and the latter is defined in $T(v_i)$. Without loss of generality, we can assume that $\pi'(u) = \pi^*(u)$, for each $u \in T(v) \setminus T_{i-1}(v)$, that is $B_{\pi'}(v) \subseteq T_{i-1}(v)$. In fact, nodes in $T(v) \setminus T_{i-1}(v)$ are not considered in the computation of function f_{i-1}^v and then there always exists a timetable which fulfills such a condition and maximizes (1). Hence, we define a solution π^s as follows:

$$\pi^s(u) = \begin{cases} \pi'(u) & \text{if } u \in T_{i-1}(v) \\ \pi''(u) & \text{if } u \in T(v_i) \\ \pi^*(u) & \text{otherwise,} \end{cases}$$

for each node u in T. It follows that, the value of (1) + (2) is given by $f_i^v(\pi^*) - f_i^v(\pi^s)$ and that π^s induces a ball $B_{\pi^s}(v)$ such that $B_{\pi^s}(v) \cap T_{i-1}(v) \leq j - s$ and $B_{\pi^s}(v) \cap T(v_i) \leq s$, and then $|B_{\pi^s}(v)| \leq j$. Therefore, $\max_{0 \leq s < j} \{G_v[i-1, j-s] + G_{v_i}[|N_o(v_i)|, s]\} = \max_{0 \leq s < j} \{f_i^v(\pi^*) - f_i^v(\pi^s)\} = \max_{\pi} \{f_i^v(\pi^*) - f_i^v(\pi) \ : \ |B_\pi(v)| \leq j\} = G_v[i, j]$. □

Informally, if $j = 0$ (Fig. 1(a) and Item 1 of Lemma 6), then $G_v[i, 0]$ is defined as the maximum gain of a solution π wrt f^* achievable by an empty ball $B_\pi(v)$. Thus, solution π must have the slack time on the in-arc of v set to α. Therefore, by Lemma 5, all the out-arcs of v have a slack time equal to 0, and so the optimal solution π wrt $T_i(v)$ in each subtree rooted at v_ℓ, $1 \leq \ell \leq i$, of v has a ball $B_\pi(v_\ell)$ of maximal size, as proved in Lemma 4. Specifically, $B_\pi(v_\ell)$ has size $\min\{\Delta, |T(v_\ell)|\}$. Hence, the overall gain $G_v[i, 0]$ wrt f^* is recursively computed as the sum of the gains achieved at the first i children of v, that is, $G_v[i, 0] = \sum_{\ell=1}^{i} G_{v_\ell}[|N_o(v_\ell)|, \Delta]$.

If $i = 0$ and $j > 0$ (Fig. 1(b) and Item 2 of Lemma 6), then $G_v[0, j]$ is defined as the maximum gain of a solution π wrt f^* achievable by constructing a ball $B_\pi(v)$ of size j without considering the children of v, that is $|B_\pi(v)| = 1$ independently from j. By Lemma 3, $G_v[0, j] = \alpha c(v)$, for each $1 \leq j \leq \Delta$.

If $i > 0$ and $j > 0$ (Fig. 1(c) and Item 3 of Lemma 6), then $G_v[i, j]$ takes its most general meaning. Since $j > 0$, the considered solution π sets the slack time incoming into v to 0. Exploiting the sub-optimality property, $B_\pi(v)$ is built recursively on the subtrees $T_{i-1}(v)$ and $T(v_i)$ where two balls, whose sizes sum up to j, are considered. For a chosen size s of $B_\pi(v_i)$, the gain is recursively computed as $G_v[i-1, j-s] + G_{v_i}[|N_o(v_i)|, s]$. Thus, $G_v[i, j]$ is determined by considering all the possible sizes s, with $0 \leq s \leq j-1$, of $B_\pi(v_i)$. Note that, since $j > 0$, the in-arc of v has the slack time set to 0, and hence $B_\pi(v_i)$ cannot have size larger than $j - 1$. Finally, $G_v[i, j] = \max_{0 \leq s < j} \{G_v[i-1, j-s] + G_{v_i}[|N_o(v_i)|, s]\}$.

For each node v, the `SA-DP` algorithm also memorizes in SOL_v the choices that lead to the optimal gains computed in the corresponding matrix G_v (Lines 9 and 11 of the algorithm). All matrices SOL_v, $v \in T$, are first initialized by assigning 0 to each entry. Then, if $i, j > 0$, the entry $SOL_v[i, j]$ stores the size s of the ball $B_\pi(v_i)$ rooted at the i-th child v_i of v which gives the maximum gain when evaluating $G_v[i, j]$. Basically, if $SOL_v[i, j] \neq 0$, then the arc (v, v_i) has no slack time. Viceversa, if $SOL_v[i, j] = 0$, either the arc (v, v_i) has a slack time of α (i.e., $|B_\pi(v_i)| = 0$ that is $j = 0$) or such an arc does not exist (i.e., $i = 0$).

Now, we have to show how to construct the optimal solution π from matrices SOL_v, i.e. how to assign to each $v \in T$ the value $\pi(v)$. Recall first that, by Lemma 5, $\pi(r) = 0$ and, for each $v_\ell \in N_o(r)$, $\pi(v_\ell) = \pi(r) + 1 = 1$. Moreover, a ball $B_\pi(v_\ell)$ of size $\min = \{|T(v_\ell)|, \Delta\}$ is rooted at each $v_\ell \in N_o(r)$. An optimal solution is computed by calling, for each child $v_\ell \in N_o(r)$, the procedure `BUILD`$(v_\ell, |N_o(v_\ell)|, \Delta)$, depicted in Fig. 3. Algorithm `BUILD`(v, i, j) recursively

Algorithm SA-DP
Input: $v \in V$
Output: SOL_v, for each $v \in V$
1. **for** $i = 1$ to k
2. SA-DP(v_i)
3. $G_v[i,0] = \sum_{\ell=1}^{i} G_{v_\ell}[|N_o(v_\ell)|, \Delta]$, for $0 \le i \le N_o(v)$
4. $G_v[0,j] = \alpha c(v)$, for $1 \le j \le \Delta$
5. **for** $i = 1$ **to** k
6. **for** $j = 1$ **to** Δ
7. **if** $j \le |T_i(v)|$ **then**
8. $G_v[i,j] = \max_{0 \le s < j} \{G_v[i-1, j-s] + G_{v_i}[|N_o(v_i)|, s]\}$
9. let s^* be the index determining the maximum at Line 8, $SOL_v[i,j] = s^*$
10. **else** $G_v[i,j] = G_v[i,j-1]$
11. $SOL_v[i,j] = SOL_v[i,j-1]$

Fig. 2.

Algorithm BUILD
Input: $v \in V$, $i \in [0, |N_o(v)|]$, $j \in [0, \Delta]$
Output: π
1. **if** $i > 0$ **then**
2. **if** $SOL_v[i,j] > 0$ **then**
3. $s := SOL_v[i,j]$; $\pi(v_i) := \pi(v) + 1$
4. BUILD$(v_i, |N_o(v_i)|, s)$
5. BUILD$(v, i-1, j-s)$
6. **else**
7. $\pi(v_i) := \pi(v) + 1 + \alpha$
8. **for each** $w \in N_o(v_i)$
9. $\pi(w) := \pi(v_i) + 1$
10. BUILD$(w, |N_o(w)|, \Delta)$
11. BUILD$(v, i-1, j)$

Fig. 3.

builds the most profitable ball $B_\pi(v)$ of size j with respect to $T_i(v)$. At first, it determines the slack time on the arc (v, v_i) depending on $SOL_v[i,j]$.

If $SOL_v[i,j] > 0$, then the slack time on the arc (v, v_i) is equal to 0, and the ball $B_\pi(v)$ consists of two sub-balls: one of size $SOL_v[i,j]$ in $T(v_i)$ and one of size $j - SOL_v[i,j]$ in $T_{i-1}(v)$. Whereas, if $SOL_v[i,j] = 0$, the ball of size j in the subtree $T(v)$ does not extend over $T(v_i)$ and it completely belongs to $T_{i-1}(v)$. However, since the slack time on the arc (v, v_i) is equal to α, by Lemma 5, all the arcs outgoing from v_i, say (v_i, w_k) for $1 \le k \le |N_o(v_i)|$, have slack time equal to 0. The balls of size at most Δ rooted at the nodes $w \in N_o(v_i)$, are recursively built by invoking $|N_o(v_i)|$ times Algorithm BUILD, one for each child of v_i.

Finally, we define the Algorithm SA-DP_BUILD which outputs a robust timetable π by performing first SA-DP(r), and then BUILD$(v_\ell, |N_o(v_\ell)|, \Delta)$, for each $v_\ell \in N_o(r)$.

Theorem 2. *SA-DP_BUILD is $\mathcal{RTT}$-optimal.*
For $\Delta \geq 1$, $P_{rob}(\mathcal{RTT}, \texttt{SA-DP\_BUILD}) \leq 1 + \frac{\alpha}{2}$ and $P_{rob}(\mathcal{RTT}) \geq 1 + \frac{\alpha}{\Delta+1}$.

Theorem 3. *For $\Delta \geq 1$, SA-DP_BUILD requires $O(\Delta^2 n)$ time and $O(\Delta n)$ space. For $\Delta = 0$, SA-DP_BUILD requires $O(n)$ time and $O(n)$ space.*

References

1. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Robust Algorithms and Price of Robustness in Shunting Problems. In: Proc. of 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, ATMOS, pp. 175–190 (2007)
2. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Delay Management Problem: Complexity Results and Robust Algorithms. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 458–468. Springer, Heidelberg (2008)
3. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Recoverable Robust Timetabling: Complexity Results and Algorithms. ARRIVAL-TR-0172, ARRIVAL Project (2008)
4. Cicerone, S., Di Stefano, G., Schachtebeck, M., Schöbel, A.: Dynamic Algorithms for Recoverable Robustness Problems. In: The 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, ATMOS (2008)
5. D'Angelo, G., Di Stefano, G., Navarra, A.: A Dynamic Programming Approach for Recoverable Robust Timetables on Trees. ARRIVAL-TR-0217, ARRIVAL Project (2009)
6. D'Angelo, G., Di Stefano, G., Navarra, A.: Recoverable-Robust Timetables for Trains on Single-Line Corridors. In: The 3rd International Seminar on Railway Operations Modelling and Analysis, RailZurich (2009)
7. D'Angelo, G., Di Stefano, G., Navarra, A.: Recoverable Robust Timetables on Trees. ARRIVAL-TR-0163, ARRIVAL Project (2008)
8. De Giovanni, L., Heilporn, G., Labbé, M.: Optimization Models for the Delay Management Problem in Public Transportation. European Journal of Operational Research 189(3), 762–774 (2007)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
10. Gatto, M., Jacob, R., Peeters, L., Widmayer, P.: Online Delay Management on a Single Train Line. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 306–320. Springer, Heidelberg (2007)
11. Ginkel, A., Schöbel, A.: The Bicriteria Delay Management Problem. Transportation Science 4(41), 527–538 (2007)
12. Levy, F.K., Thompson, G.L., Wies, J.D.: The ABCs of the Critical Path Method. Graduate School of Business Administration, Harvard University (1963)
13. Liebchen, C., Lübbecke, M., Möhring, R.H., Stiller, S.: Recoverable Robustness. ARRIVAL-TR-0066, ARRIVAL Project (2007)
14. Schöbel, A.: A Model for the Delay Management Problem Based on Mixed Integer Programming. ENTCS 50(1), 1–10 (2004)
15. Schöbel, A.: Integer Programming Approaches for Solving the Delay Management. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 145–170. Springer, Heidelberg (2007)

Roulette Wheel Graph Colouring for Solving Examination Timetabling Problems

Nasser R. Sabar[1], Masri Ayob[1], Graham Kendall[2], and Rong Qu[2]

[1] Jabatan Sains Komputer, Fakulti Teknologi dan Sains Maklumat
Universiti Kebangsaan Malaysia, 43600 UKM, Bangi Selangor
naserdolayme@yahoo.com, masri@ftsm.ukm.my
[2] ASAP Research Group, School of Computer Science
The University of Nottingham, Nottingham NG8 1BB, UK
{gxk,rxq}@cs.nott.ac.uk

Abstract. This work presents a simple graph based heuristic that employs a roulette wheel selection mechanism for solving exam timetabling problems. We arrange exams in non-increasing order of the number of conflicts (degree) that they have with other exams. The difficulty of each exam to be scheduled is estimated based on the degree of exams in conflict. The degree determines the size of a segment in a roulette wheel, with a larger degree giving a larger segment. The roulette wheel selection mechanism selects an exam if the generated random number falls within the exam's segment. This overcomes the problem of repeatedly choosing and scheduling the same sequence of exams. We utilise the proposed Roulette Wheel Graph Colouring heuristic on the un-capacitated Carter's benchmark datasets. Results showed that this simple heuristic is capable of producing feasible solutions for all 13 instances.

Keywords: scheduling, examination timetabling, graph colouring heuristics, roulette wheel selection.

1 Introduction

Examination timetabling problems deal with assigning a given set of exams into a given set of timeslots, subject to sets of hard and soft constraints [15]. A timetable is feasible if all exams have been assigned to timeslots and all required hard constraints are satisfied. Soft constraints are represent features that we wish to avoid but we can violate them if necessary. However, violations of soft constraints should be minimized as much as possible, and it is the minimization of the soft constraints that indicates the quality of the generated timetable.

Various approaches have been used to construct examination timetables such as graph colouring [15,10], fuzzy logic [2], ant algorithms [23] and neural networks [18]. Graph colouring heuristics represent exams as vertices and conflicts between exams as edges. The aim is to colour adjacent vertices with a different colour. Every colour represents a time period in the timetable. In the literature, various heuristics have been used to construct a conflict free timetable by using graph

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 463–470, 2009.

colouring models [15,20,26,13,3]. These heuristics prioritize exams according to the level of scheduling difficulty. The rationale behind this is to make sure the most difficult exams are scheduled first. Examples of the most common graph colouring/timetabling heuristics are:

- Largest Degree First (LD): The exams are ordered according to the number of conflicts, in decreasing order.
- Least Saturation Degree First (LS): Exams are dynamically ordered by the number of available timeslots, in ascending order.
- Largest Enrolment First (LE): The exams are ordered according to the number of students enrolled, in decreasing order.

Asmuni et al. [2] used fuzzy logic to order exams based on graph colouring heuristics. When ordering the exams, by their degree of difficulty, fuzzy functions were introduced to evaluate the degree of difficulty. Results show that the fuzzy approach is capable of producing good quality solutions (when tested on Carter's benchmark datasets). However, they noted that different fuzzy functions need to be used on different problems to obtain the best results.

Corr et al. [18] utilized a neural network to determine the level of difficulty of assigning exams, with the most difficult exam being scheduled first. The neural network was built by storing feature vectors using three graph heuristics. The research demonstrated the feasibility of employing neural network based methods as an adaptive and generally applicable technique to timetabling problems.

More recently, Eley [23] applied an ant algorithm to simultaneously construct and improve the timetables. He used two ant colony approaches; MMAS-ET that is based on Max-Min Ant System (MMAS) (used by Socha et al. [27] on course timetabling) and ANTCOL-ET which is a modified version of ANTCOL (originally used by Costa and Hertz [19] to solve graph colouring problems). Both ant algorithms were hybridized with a hill climber and found that the simple ant system, ANTCOL, outperformed the more complex Max-Min algorithm. It was concluded that the performance of ant systems can be improved by adjusting the algorithm parameters.

Meta-heuristic approaches have also been used to improve solutions. Examples include tabu search [21,24,29], simulated annealing [11,28], genetic algorithms [7], memetic algorithms [12,9] great deluge algorithms [5], ant Colony [22], particle swarm optimization [17] and hybridizations of different techniques [14,25,1,6].

There are a number of survey papers on examination timetabling, which is essential reading for those new to the area [8,26,13].

In this work, we propose the Roulette Wheel Graph coloring heuristic (RWG) to construct examination timetables. We utilize the Largest Degree First heuristic to compute the difficulty of exams to be scheduled. Then, the degree of exams in conflict is used to determine the size of a segment in a *roulette wheel*. Finally, a roulette wheel selection mechanism is used select the next exam to be scheduled, in order to generate a feasible timetable.

The aim of this work is to investigate the capability of using a probability selection mechanism to construct examination timetables. The overall idea is to overcome the problem of repeatedly choosing the same sequence of exams to be

scheduled, which is a common problem when using a deterministic graph coloring heuristic. In order to demonstrate the efficiency of the proposed heuristic, test it on the un-capacitated Carter's benchmark examination timetable dataset [16] (variant b, type I, see [26]) using the standard evaluation function given in [16].

2 Roulette Wheel Graph Coloring

In 1987 Baker [4] presented a simple selection schema called roulette wheel selection. This is a stochastic algorithm and is described as follows:

1. The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness.
2. A random number is generated and the individual whose segment spans the random number is selected.
3. The process is repeated until the desired number of individuals is obtained.

This technique is analogous to a roulette wheel with each segment proportional to its fitness. However, in this work, we compute the segment area by using Eq. 1:

$$S(i) = S(i-1) + \frac{d_i}{\sum_{i=1}^{n} d(i)} \text{ for all } i \in (1, \cdots, n), S(i-1) = 0 \text{ if } i = 1, \quad (1)$$

where $S(i)$ is the segment area and $d(i)$ is the number of exams in conflict. To see how roulette wheel selection mechanism works, we give the following example. Suppose we have the exams as shown in Table 1, the number of exams in conflict is shown in column 3.

Then, the area of Segment can be calculated by using Eq.1. For example, the Segment size for exams A and B in Table 1 are calculated as follows:

$$d(1) = 10, d(2) = 9 \text{ and } \sum_{i=1}^{n} d = 72.$$

So the sector size for A and B will be $S(1) = S(1-1) + 10/72 = 0 + 0.13 = 0.13$; $S(2) = S(2-1) + 9/72 = 0.13 + 0.12 = 0.25$, respectively.

Indeed, based on Table 1, exam C is the fittest one and occupies the largest interval (biggest segment); whereas exam number B is the least fit exam and has

Table 1. An illustrative example of roulette wheel selection

Exam No	Exams	Number of exams in conflicts	Segment Size	Segment Area (using Eq. 1)
1	A	10	0.13	0.00-0.13
2	B	9	0.12	0.14-0.25
3	C	20	0.27	0.26-0.53
4	D	15	0.20	0.54-0.73
5	E	18	0.25	0.74-1.00

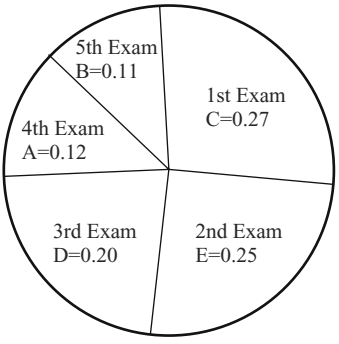


Fig. 1. Sector sizes of exams calculated using Eq. 1 for the example in Table 1

the smallest interval on the line. Based on the sector size in calculated in Table 1. Fig. 1 shows the exams sector order. The pseudo-code of the Roulette Wheel Graph coloring heuristic is presented in Fig. 2.

Step 1 Initialization
- Calculate the degree of difficulty for each unscheduled exam e in a non-increasing order of the number of conflicts they have with other exams in unscheduled exam list E
- Calculate the maximum span for each unscheduled exam e using Eq. (1).

Step 2 Schedule exams into timeslots

While ($E\emptyset$)

{
- Generate a random number r between [0, 1].
- Select the exam e where r falls within its segment span.
- Calculate the number of available clash free timeslots for the chosen exam e.
- If number of available clash free timeslots > 0 then
  {
    - Schedule e to the minimum penalty period.
    - Remove e from Unscheduled list E.
  }
- Else Break; // Terminate the procedure.

} // end of while $E\emptyset$

If $E = \emptyset$ then return the solution

Fig. 2. Pseudo-code of the Roulette Wheel Graph colouring heuristic

In the initialization step, all exams in E are sorted on a decreasing order of the number of conflict they have with other exams. Then, we calculate the segment size for all exams based on Eq. 1. In the second step, we generate a random number r between [0, 1]. The exam whose segment span the random number

is selected. Next, we calculate the number of available clash free timeslots for the selected exam e. If the number of available clash frees timeslots is greater than zero, allocate exam e to the minimum penalty clash free timeslot and remove exam e from unscheduled list E. If there is more than one timeslot that has the same minimum penalty, the timeslot is randomly selected from the set of minimum cost timeslots. If there is no available clash free timeslots for the current exam terminate the procedure and returns an infeasible solution.

3 Results on Benchmark Examination Timetabling Dataset

The Roulette Wheel Graph colouring heuristic was tested on Carter's un-capacitated examination timetabling benchmark datasets [16] (variant b, type I, [26]) which contains 13 problem instances. These datasets have been used by many researchers in the literature since 1996 [26]. Table 2 shows the number of timeslots, number of exams and the number of student enrolments for these datasets.

The results (out of 20 runs for each instance) are shown in Table 3, comparing the results against other constructive heuristics. Results indicate that this simple heuristic is able to produce feasible initial solutions for all instances. Note that all the other algorithms being compared in Table 3 employed more advanced techniques i.e. fuzzy techniques, neural networks, ant algorithms (constructive and improvement heuristics) and tabu search hyper-heuristic. Our constructive heuristic is simple and generally applicable to produce feasible solutions for all 13 instances being studied.

It is known from the literature that for the benchmark dataset tested here, by employing the mostly used graph colouring heuristics such as largest degree

Table 2. Un-capacitated standard Carter benchmark exam timetabling dataset

Data sets	Number of timeslots	Number of examinations	Number of Students
Car-f-92	32	543	18419
Car-s-91	35	682	16925
Ear-f-83	24	190	1125
Hec-s-92	18	81	2823
Kfu-s-93	20	461	5349
Lse-f-91	18	381	2726
Pur-s-93	43	2419	30032
Rye-s-93	23	486	11483
Sta-f-83	13	139	611
Tre-s-92	23	261	4360
Uta-s-92	35	622	21267
Ute-s-92	10	184	2750
Yor-f-83	21	181	941

and saturation degree, etc, alone cannot obtain feasible solutions for some of the difficult instances in Table 2. Other intelligent mechanisms need to be employed, or the graph colouring heuristics are repeated to obtain feasible solutions for advanced meta-heuristics [26]. Our new graph colouring heuristic with roulette wheel selection presents an effective and simple construction heuristic, and may be subsequently be improved by using a wide range of meta-heuristics.

Table 3. Results obtained from Roulette wheel Graph Colouring compared to constructive heuristics in the literature

Data sets	**Best**	**% feasible**	**Standard deviation**	Asmuni et al. [2]	Corr et al. [18]	Eley [23]	Burke et al. [10]
Car-f-92	6.21	80	0.26	4.56	6.24	4.8	**4.53**
Car-s-91	7.01	90	0.11	**5.29**	7.21	5.7	5.36
Ear-f-83	42.81	85	0.35	37.02	49.44	**36.8**	37.92
Hec-s-92	12.90	70	0.08	11.87	13.57	**11.3**	12.25
Kfu-s-93	18.47	90	0.17	15.81	19.9	**15.0**	15.2
Lse-f-91	15.62	95	0.16	12.09	14.99	12.1	**11.33**
Pur-s-93	7.92	30	0.34	-	-	5.4	-
Rye-s-93	10.5	60	0.29	10.35	-	**10.2**	-
Sta-f-83	161.00	100	0.14	160.42	159.28	**157.2**	158.19
Tre-s-92	10.98	100	0.31	8.67	10.77	**8.8**	8.92
Uta-s-92	4.76	85	0.10	3.57	4.48	3.8	**2.88**
Ute-s-92	29.69	80	0.50	27.78	31.25	**27.7**	28.01
Yor-f-83	39.83	100	0.35	40.66	-	**39.6**	41.37

4 Conclusion

We have proposed a Roulette wheel Graph coloring heuristic for solving examination timetabling problems. We first utilize the Largest Degree First graph coloring heuristic to order exams. Then, based on the degree of exams in conflict, we determine the size of its segment in a roulette wheel. An exam is then selected if a generated random number falls within the exam segment. This overcomes the problem of repeatedly choosing the same sequence of exams to be scheduled. The current constructive heuristics shown in the literature quite often lead to low quality timetables or infeasible timetable especially for the more difficult problem instances in the benchmark dataset considered here. Our results show that the Roulette wheel Graph coloring heuristic is capable of producing feasible solutions for all problem instances.

References

1. Abdullah, S., Burke, E.K.: A Multi-Start Large Neighbourhood Search Approach with Local Search Methods for Examination Timetabling. In: Long, D., Smith, S.F., Borrajo, D., McCluskey, L. (eds.) ICAPS 2006, Cumbria, UK, pp. 334–337 (2006)

2. Asmuni, H., Burke, E.K., Garibaldi, J.M., McCollum, B.: Fuzzy multiple heuristic orderings for examination timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 334–353. Springer, Heidelberg (2005)
3. Ayob, M., Malik, A.M.A., Abdullah, S., Hamdan, A.R., Kendall, G., Qu, R.: Solving a Practical Examination Timetabling Problem: A Case Study. In: Gervasi, O., Gavrilova, M.L. (eds.) ICCSA 2007, Part III. LNCS, vol. 4707, pp. 611–624. Springer, Heidelberg (2007)
4. Baker, J.: Reducing Bias and Inefficiency in the Selection Algorithm. In: Proceeding of the Second International Conference on Genetic Algorithms (ICGA 1987), pp. 14–21. L. Erlbaum Associates Inc., Hillsdale (1987)
5. Burke, E.K., Bykov, Y., Newall, J.P., Petrovic, S.: A Time-Predefined Local Search Approach to Exam Timetabling Problems. IIE Transactions on Operations Engineering 36, 509–528 (2004)
6. Burke, E.K., Eckersley, A.J., McCollum, B., Petrovic, S., Qu, R.: Hybrid Variable Neighbourhood Approaches to University Exam Timetabling. European Journal of Operational Research (to appear)
7. Burke, E.K., Elliman, D.G., Weare, R.F.: A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems. In: Proceedings of the 6th International Conference on Genetic Algorithms (ICGA 1995), pp. 605–610. Morgan Kaufmann, San Francisco (1995)
8. Burke, E.K., Kingston, J., de Werra, D.: Applications to timetabling. In: Gross, J., Yellen, J. (eds.) Handbook of Graph Theory, pp. 445–474. Chapman Hall/CRC Press, Boca Raton (2004)
9. Burke, E.K., Landa Silva, J.D.: The Design of Memetic Algorithms for Scheduling and Timetabling Problems. In: Hart, W.E., Krasnogor, N., Smith, J.E. (eds.) Studies in Fuzziness and Soft Computing: Recent Advances in Memetic Algorithms and Related Search Technologies, vol. 166, pp. 289–312. Springer, Heidelberg (2004)
10. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A Graph-Based Hyper-Heuristic for Educational Timetabling Problems. European Journal of Operational Research 176, 177–192 (2007)
11. Burke, E.K., Newall, J.: Enhancing timetable solutions with local search methods. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 195–206. Springer, Heidelberg (2003)
12. Burke, E.K., Newall, J.P., Weare, R.F.: A Memetic Algorithm for University Exam Timetabling. In: Burke, E.K., Ross, P. (eds.) PATAT 1995. LNCS, vol. 1153, pp. 241–250. Springer, Heidelberg (1996)
13. Burke, E.K., Petrovic, S.: Recent Research Directions in Automated Timetabling. European Journal of Operational Research 140, 266–280 (2002)
14. Caramia, M., Dell'Olmo, P., Italiano, G.F.: New Algorithms for Examination Timetabling. In: Näher, S., Wagner, D. (eds.) WAE 2000. LNCS, vol. 1982, pp. 230–241. Springer, Heidelberg (2001)
15. Carter, M.W., Laporte, G.: Recent Developments in Practical Examination Timetabling. In: Burke, E.K., Ross, P. (eds.) PATAT 1995. LNCS, vol. 1153, pp. 3–21. Springer, Heidelberg (1996)
16. Carter, M.W., Laporte, G., Lee, S.Y.: Examination Timetabling: Algorithmic Strategies and Applications. Journal of Operational Research Society 47(3), 373–383 (1996)
17. Chu, S.C., Chen, Y.T., Ho, J.H.: Timetable Scheduling Using Particle Swarm Optimization. In: Proceedings of the 1st International Conference on Innovation computing, Information and Control, vol. 3, pp. 324–327 (2006)

18. Corr, P.H., McCollum, B., McGreevy, M.A.J., McMullan, P.: A New Neural Network Based Construction Heuristic for the Examination Timetabling Problem. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 392–401. Springer, Heidelberg (2006)
19. Costa, D., Hertz, A.: Ant can Colour Graphs. Journal of Operational Research Society 48, 295–305 (1997)
20. de Werra, D.: An Introduction to Timetabling. European Journal of Operational Research 19, 151–162 (1985)
21. Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 104–117. Springer, Heidelberg (2001)
22. Dowsland, K., Thompson, J.: Ant Colony Optimisation for the Examination Scheduling Problem. Journal of the Operational Research Society 56, 426–439 (2005)
23. Eley, M.: Ant Algorithms for the Exam Timetabling Problem. In: Burke, E.K., Rudová, H. (eds.) PATAT 2007. LNCS, vol. 3867, pp. 364–382. Springer, Heidelberg (2007)
24. Kendall, G., Hussin, N.M.: Tabu Search Hyper-Heuristic Approach to the Examination Timetabling Problem at University Technology MARA. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 199–218. Springer, Heidelberg (2005)
25. Merlot, L.T.G., Borland, N., Hughes, B.D., Stuckey, P.J.: A Hybrid Algorithm for the Examination Timetabling Problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 207–231. Springer, Heidelberg (2003)
26. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G., Lee, S.Y.: A Survey of Search Methodologies and Automated System Development for Examination Timetabling. Journal of Scheduling 12(1), 55–89 (2009)
27. Socha, K., Sampels, M., Manfrin, M.: Ant Algorithms for the University Course Timetabling Problem with Regard to State-of-the-Art. In: Proceedings of the 3rd European Workshop on Evolutionary Computation in Combinatorial Optimisation, Essex, UK, pp. 334–345 (2003)
28. Thompson, J., Dowsland, K.: A Robust Simulated Annealing Based Examination Timetabling System. Computers Operations Research 25, 637–648 (1998)
29. White, G.M., Xie, B.S.: Examination Timetables and Tabu Search with Longer-Term Memory. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 85–103. Springer, Heidelberg (2001)

Integrated Production and Delivery Scheduling with Disjoint Windows

Yumei Huo[1,⋆], Joseph Y.-T. Leung[2,⋆⋆], and Xin Wang[3]

[1] Department of Computer Science
College of Staten Island, CUNY, Staten Island, New York 10314, USA
huo@mail.csi.cuny.edu
[2] Department of Computer Science
New Jersey Institute of Technology, Newark, NJ 07102, USA
leung@oak.njit.edu
[3] R&D Department, Bloomberg L.P.
731 Lexington ave., New York, NY 10022, USA
xwang39@bloomberg.net

Abstract. Consider a company that manufactures perishable goods. The company relies on a third party to deliver goods, which picks up delivery products at regular times. At each delivery time, there is a time window that products can be produced to be delivered at that delivery time. Suppose we have a set of jobs with each job specifying its delivery time, processing time and profit. The company can earn the profit of the job if the job is produced and delivered at its specified delivery time. From the company point of view, we are interested in picking a subset of jobs to produce and deliver so as to maximize the total profit. The jobs that are not picked will be discarded without penalty. We consider both the single machine case and the parallel and identical machines case.

In this article we consider two kinds of profits: (1) arbitrary profit, (2) profit proportional to its processing time. In the first case, we give a fully polynomial time approximation scheme (FPTAS) for a single machine with running time $O(\frac{n^3}{\epsilon})$. In the second case, we give a faster FPTAS for a single machine with running time $O(\frac{n^2}{\epsilon})$. All of our algorithms can be extended to parallel and identical machines with a degradation of performance ratios.

Keywords: Perishable goods, Single machine, Parallel and identical machines, NP-hard and strong NP-hard, Fully polynomial time approximation schemes.

1 Introduction

Consider a company that produces perishable goods. The company relies on a third party to deliver goods, which picks up delivery products at regular times

⋆ This author is supported in part by PSC-CUNY research grant.
⋆⋆ This author is supported by NSF grant DMI-0556010.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 471–482, 2009.

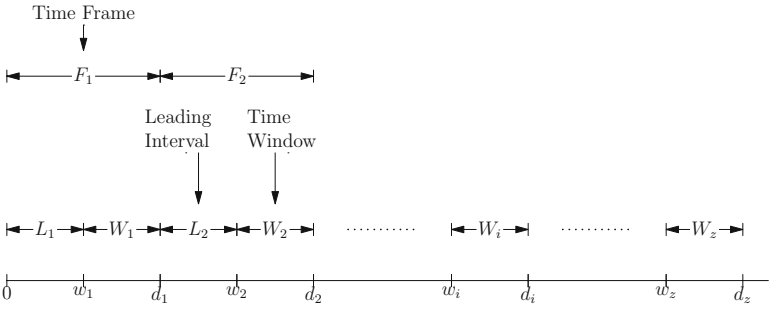


Fig. 1. Illustrating the Definitions of Time Window, Leading Interval, and Time Frame

(for example, at 10:00am every morning). Because the goods are perishable, it is infeasible to produce the goods far in advance of the delivery time. Thus, at each delivery time, there is a time window that the goods can be produced and delivered at that delivery time. Consider a planning horizon T. We have a set of jobs with each job specifying its delivery time, processing time and profit. The company can earn the profit of the job if the job is produced and delivered at its specified delivery time. From the company point of view, we are interested in picking a subset of jobs to produce and deliver so as to maximize the total profit. The jobs that are not picked will be discarded without any penalty, which is realistic since the manufacture may reject some orders quickly without incurring any penalty.

Formally, we have a planning horizon $T = \{d_1, d_2, \cdots, d_z\}$, consisting of z delivery times. For each delivery time d_j, there is a time instant $w_j < d_j$ such that a job must be completed in the time window $[w_j, d_j]$ if it were to be delivered at time d_j. We denote the time window $[w_j, d_j]$ by W_j. The time windows are assumed to be disjoint. Thus, we have $w_1 < d_1 < w_2 < d_2 < \cdots < w_z < d_z$. Let $d_0 = 0$ be a dummy delivery time. Preceding each time window W_j is the *leading interval* $L_j = (d_{j-1}, w_j)$. We call a time window together with its leading interval a *time frame*, and it is denoted by $F_j = L_j \cup W_j$. Figure 1 depicts the definitions of time window, leading interval and time frame. Let W be the length of a time window and L be the length of a leading interval. We assume that all time windows have the same length W, and all leading intervals have the same length L, although our algorithms still work when this restriction is relaxed.

Within the planning horizon, there is a set of jobs $J = \{J_1, J_2, \cdots, J_n\}$. Associated with each job J_i is a processing time p_i, a delivery time $d_j \in T$, $1 \leq j \leq z$, and a profit pf_i. The job J_i is supposed to be delivered at the delivery time d_i, its processing time is p_i, and the company can earn a profit pf_i if the job can be finished producing in the time window W_i and delivered at the delivery time d_i. From the company point of view, we are interested in picking a subset of jobs to produce and deliver so as to maximize the total profit. The jobs that are not produced and delivered will be discarded without any penalty. We assume that all job information is known in advance, preemption is not allowed, and there is no vehicle limitation at any delivery date.

In the past, production scheduling have focused on the issue of how machines are allocated to jobs in the production process so as to obtain an optimal or near-optimal schedule with respect to some objective functions. In the last two decades, integrated production and delivery scheduling problems have received considerable interest. However, most of the research for this model is done at the strategic and tactical levels (see [2,3,4,6,10,13,14] for examples), while very little is known at the operational scheduling level. Chen [4] classified the model at the operational scheduling level into four classes: (1) Models with individual and immediate delivery; (2) Models with batch delivery to a single customer; (3) Models with batch delivery to multiple customers; and (4) Models with fixed delivery departure date. In the models with individual and immediate delivery, Garcia and Lozano [7] is the only paper that studies a model with delivery time windows. They gave a tabu-search solution procedure for the problem and their objective function is the maximum number of jobs that can be processed. In the models with individual and immediate delivery, problems with fixed delivery date are also studied in [8,9]. In the models with fixed delivery departure date, no time window constraint is considered and the focus is on the delivery cost.

Bar-Noy et al. [1] considered a more general version of our scheduling problem. There are given n jobs to be scheduled nonpreemptively on m machines. Associated with each job is a release time, a deadline, a weight (or profit), and a processing time on each of the machines. The goal is to find a nonpreemptive schedule that maximizes the weight (or profit) of jobs that meet their respective deadlines. (Note that in the problem studied by Bar-Noy et al. [1], the intervals in which the jobs can execute can overlap in an arbitrary manner.) This problem is known to be strongly NP-hard, even for a single machine. They obtained the following results [1]:

- For identical job weights and unrelated machines: a greedy 2-approximation algorithm.
- For identical job weights and m identical machines: the same greedy algorithm achieves a tight $\frac{(1+1/m)^m}{(1+1/m)^m-1}$ approximation factor.
- For arbitrary job weights and a single machine: an LP formulation achieves a 2-approximation for polynomially bounded integral input and a 3-approximation for arbitrary input. For unrelated machines, the factors are 3 and 4, respectively.
- For arbitrary jobs weights and m identical machines: the LP-based algorithm applied repeatedly achieves a $\frac{(1+1/m)^m}{(1+1/m)^m-1}$ approximation factor for polynomially bounded integral input and a $\frac{(1+1/2m)^m}{(1+1/2m)^m-1}$ approximation factor for arbitrary input.
- For arbitrary job weights and unrelated machines: a combinatorial $(3+2\sqrt{2})$-approximation algorithm.

In this article we consider two kinds of profits: (1) arbitrary profit, (2) profit proportional to its processing time. In the first case, we give a pseudo-polynomial time algorithm to find an optimal schedule on a single machine. Based on the pseudo-polynomial time algorithm, we develop a fully polynomial time

approximation scheme (FPTAS) with running time $O(\frac{n^3}{\epsilon})$. It should be noted that this FPTAS still work when the window sizes are not equal and windows are not disjoint. In the second case, we give a FPTAS with an improved running time, $O(\frac{n^2}{\epsilon})$ versus $O(\frac{n^3}{\epsilon})$. And this FPTAS still work when the window sizes are not equal. All of our algorithms can be extended to parallel and identical machines with a degradation of performance ratios.

The article is organized as follows. In the next section we consider the arbitrary profit case. We first give the pseudo-polynomial time algorithm and the FPTAS on a single machine. We then describe the extension of the algorithm to the parallel and identical machines case. In Section 3, we consider the case of profit proportional to its processing time. We give a FPTAS with running time $O(\frac{n^2}{\epsilon})$, and describe the extension to the case of parallel and identical machines. Finally, we draw some conclusions in Section 4.

2 Arbitrary Profit

In this section we consider the arbitrary profit case. In Section 2.1, we give a pseudo-polynomial time algorithm for a single machine; the algorithm has a running time $O(nV)$, where $V = \sum_{i=1}^{n} pf_i$. In Section 2.2, we give a FPTAS for a single machine; the FPTAS has running time $O(\frac{n^3}{\epsilon})$. Finally, in Section 2.3, we extend the algorithm to parallel and identical machines.

2.1 Pseudo-polynomial Time Algorithm

In this section we present a dynamic programming algorithm for a single machine. The running time of the algorithm is $O(nV)$, where $V = \sum_{i=1}^{n} pf_i$ is the total profit of all the jobs.

For each $1 \leq t \leq z$, let J^t denote the set of jobs whose delivery time is d_t. We relabel all the jobs in J such that jobs in earlier time frame get smaller labels than jobs in later time frame, and for the jobs in the same time frame, jobs with longer processing time get smaller labels than jobs with shorter processing time. That is, for any pair of jobs $J_i \in J^t$ and $J_{i+1} \in J^{t'}$, either $t < t'$, or $t = t'$ and $p_i \geq p_{i+1}$.

Lemma 1. *For any feasible schedule with total profit P and finishing time t, there exists a feasible schedule $S = \{J_{i_1}, J_{i_2}, \cdots, J_{i_k}\}$ with the same profit P and the same finishing time t and $i_1 < i_2 < \cdots < i_k$.*

Proof. First, for any pair of jobs from any feasible schedule, if they are finished in different time windows, then the job that finished in the earlier time window must have smaller label than the job that finished in the later time window. Therefore, we only need to consider the order of jobs from the same time window. Suppose in the feasible schedule, $J_{i_1}, J_{i_2}, \cdots, J_{i_x}$ is the set of jobs that finished in the same time window, say $[w_t, d_t]$, and J_{i_1} is the first job that starts executing at time s. Let us rearrange these jobs after the time instant s in descending order of their processing times. It is easy to see that after rearranging: (1) all jobs are

finished at or before d_t, and (2) all jobs are finished at or after w_t. The reason is that the processing time of the first job is greater than or equal to the processing time of J_{i_1}, and $s + p_{i_1} \geq w_t$. So, after rearranging, the finishing time of the first job is greater than or equal to w_t. Therefore, the finishing time of all other jobs after rearranging is also greater than or equal to w_t. □

Let us define a table $T(i, j)$, where $0 \leq i \leq n$ and $0 \leq j \leq V$. $T(i, j)$ contains the minimum finishing time for scheduling the first i jobs such that a total profit of exactly j can be obtained. If there is no feasible schedule, we let $T(i, j)$ contain ∞. Here is the rule to compute $T(i, j)$.

(1) For $i = 0$, $T(0, 0) = 0$ and $T(0, j) = \infty$ for $j > 0$.
(2) For $i \geq 1$, let $J_i \in J^t$. Then,

$$T(i,j) = \min \begin{cases} T(i-1,j) & \\ T(i-1,j-pf_i)+p_i & \text{if } j \geq pf_i \text{ and } w_t \leq T(i-1,j-pf_i)+p_i \leq d_t \\ w_t & \text{if } j \geq pf_i \text{ and } T(i-1,j-pf_i)+p_i < w_t \\ \infty & \text{if } j \geq pf_i \text{ and } T(i-1,j-pf_i)+p_i > d_t \end{cases}$$

After filling in the whole table, we check the last row (row n) from right to left until we find the first entry $T(n, j)$ such that $T(n, j) < \infty$; j is the total profit of the optimal schedule. The running time of the algorithm is simply the size of the table which is $O(nV)$, where $V = \sum_{i=1}^{n} pf_i$. Next, we will show that the table is computed correctly.

Theorem 1. *The above algorithm correctly computes the optimal schedule.*

Proof. We will prove the theorem by induction on the number of rows in the table. The basis case, row 0, is filled correctly because we can only get zero profit from an empty job set.

Assume that rows $0, 1, \ldots, i-1$ are computed correctly, we now show that row i is also computed correctly. For row i, there are two cases to consider: (1) job J_i is not scheduled, and (2) job J_i is scheduled. In the former case, the minimum finishing time to obtain a profit exactly j is $T(i-1, j)$, by the induction hypothesis. In the latter case, we can always assume that J_i is the last job to be scheduled, by Lemma 1. In this case, we want to find a schedule with total profit exactly j and job J_i will be finished as early as possible in this schedule. There are several sub-cases to consider: (1) If $j \geq pf_i$ and $w_t \leq T(i-1, j-pf_i)+p_i \leq d_t$, then the minimum finishing time is $T(i-1, j-pf_i) + p_i$. This is because, by the induction hypothesis, $T(i-1, j-pf_i)$ is the minimum finishing time to obtain a profit exactly $j - pf_i$ from the first $i-1$ jobs, so by scheduling job J_i immediately after, job J_i will finish at time $T(i-1, j-pf_i) + p_i$. (2) If $j \geq pf_i$ and $T(i-1, j-pf_i) + p_i < w_t$, then the minimum finishing time is w_t. This is because job J_i does not finish at or beyond w_t. Therefore, we have to right-shift job J_i so that it finishes at exactly w_t. (3) If $j \geq pf_i$ and $T(i-1, j-pf_i)+p_i > d_t$, then job J_i finishes beyond its delivery time. Hence, $T(i, j) = \infty$. □

2.2 Fully Polynomial Time Approximation Scheme

The above dynamic programming algorithm is a pseudo-polynomial time algorithm. It is efficient only when the total profit is not too large. Using the method from [10], we can obtain a FPTAS for one machine.

The algorithm works as follows. Let K be a parameter to be determined later.

- Create a new set of job instance by replacing each job J_i with a job J_i' such that $pf_i' = \lfloor \frac{pf_i}{K} \rfloor$, and keep all other parameters unchanged.
- Run the dynamic programming algorithm to obtain an optimal solution for the new job instance.
- Translate the solution for the new job instance back to the solution for the original job instance.

It is clear that the running time of this algorithm is $O(\frac{nV}{K})$. Let PF_{opt} be the total profit of the optimal schedule of the original job instance and $PF_{opt'}$ be the total profit of the optimal schedule of the new job instance. Clearly, $PF_{opt'}$ can be obtained by the dynamic programming algorithm for the new job instance. For each job J_i, since $pf_i' = \lfloor \frac{pf_i}{K} \rfloor$, we have $pf_i - K \cdot pf_i' \leq K$. It follows that $PF_{opt} - K \cdot PF_{opt'} \leq Kn$. Let PF_{alg} be the total profit of our algorithm. It is clear that $PF_{alg} \geq K \cdot PF_{opt'}$. By setting $Kn = \frac{\epsilon \cdot v_{\max}}{(1+\epsilon)}$, where $v_{\max} = \max_{i=1}^{n}\{pf_i\}$, we have

$$\begin{aligned} PF_{opt} - PF_{alg} &\leq PF_{opt} - K \cdot PF_{opt'} \\ &\leq Kn \\ &\leq \frac{\epsilon \cdot v_{\max}}{(1+\epsilon)} \\ &\leq \frac{\epsilon \cdot PF_{opt}}{(1+\epsilon)}. \end{aligned}$$

It follows that $\frac{PF_{opt}}{PF_{alg}} \leq (1+\epsilon)$. The running time is

$$O\Big(\frac{nV}{K}\Big) = O\Big(\frac{n^2 \cdot v_{\max}}{K}\Big) = O\Big(\frac{n^2 \cdot v_{\max}}{\epsilon \cdot v_{\max}/n(1+\epsilon)}\Big) = O\Big(n^3(1+\frac{1}{\epsilon})\Big) = O\Big(\frac{n^3}{\epsilon}\Big).$$

Theorem 2. *There is a FPTAS for arbitrary profits on a single machine with running time* $O(\frac{n^3}{\epsilon})$.

2.3 Arbitrary Number of Machines

We use the same technique as in [1] to extend the algorithm for a single machine to arbitrary number of machines. Suppose we have an *Algorithm A* with approximation ratio β. We repeatedly use *Algorithm A* to schedule jobs, one machine after another, until all m machines are scheduled. The following lemma can be proved.

Lemma 2. *For any* β*-approximation algorithm on one machine, we can extend it to* m *machines with approximation ratio at most* $\frac{1}{1-e^{-1/\beta}}$.

Proof. Let J be the entire set of jobs, S^* be the optimal schedule for J on m machines and PF_{opt} be its total profit. Let us apply *Algorithm A* repeatedly on the m machines, one machine after another, until all m machines are scheduled. Let PF_A be the total profit obtained by this algorithm. Let S_i be the set of jobs scheduled on machine i and $A_i = \sum_{J_k \in S_i} pf_k$. Let U be the set of jobs scheduled on the first $k-1$ machines, i.e., $U = \cup_{i=1}^{k-1} S_i$. Let us consider the scheduling of the set of jobs $J \setminus U$ on m totally unoccupied machines. Let S_1^* be the optimal schedule on one machine for the set of jobs $J \setminus U$, and let $PF_{opt'}$ be the total profit of S_1^*. We have the following claim.

Claim: $PF_{opt'} \geq \frac{PF_{opt} - \sum_{i=1}^{k-1} A_i}{m}$.

Proof. For any job $J_i \in U$, if J_i is scheduled in S^*, we delete it from S^*. We then get a new schedule $\hat{S}$. Suppose that jobs $J_{i_1}, J_{i_2}, \cdots, J_{i_r}$ are deleted from S^*. Then the total profit of $\hat{S}$ will be $PF_{opt} - \sum_{j=1}^{r} pf_{i_j} \geq PF_{opt} - \sum_{i=1}^{k-1} A_i$. Clearly, $\hat{S}$ is a feasible schedule for the set of jobs $J \setminus U$. And in this feasible schedule, there must be one machine containing jobs whose total profit is at least $\frac{PF_{opt} - \sum_{i=1}^{k-1} A_i}{m}$. Therefore, $PF_{opt'} \geq \frac{PF_{opt} - \sum_{i=1}^{k-1} A_i}{m}$. □

Now, if we use *Algorithm A* to schedule the set of jobs $J \setminus U$ on one machine, we will get the same schedule as S_k. Since *Algorithm A* is a β-approximation algorithm, we have $A_k \geq \frac{PF_{opt'}}{\beta}$, and hence we have $A_k \geq \frac{1}{\beta m}(PF_{opt} - \sum_{i=1}^{k-1} A_i)$.

Adding $\sum_{i=1}^{k-1} A_i$ to both sides, we obtain

$$\sum_{i=1}^{k} A_i \geq \frac{1}{\beta m} PF_{opt} + \Big(1 - \frac{1}{\beta m}\Big) \sum_{i=1}^{k-1} A_i.$$

Letting $f(k) = \sum_{i=1}^{k} A_i$, we have

$$f(k) \geq \frac{1}{\beta m} PF_{opt} + \Big(1 - \frac{1}{\beta m}\Big) f(k-1), \quad \text{or}$$

$$f(k) - PF_{opt} \geq \Big(1 - \frac{1}{\beta m}\Big)(f(k-1) - PF_{opt}), \quad \text{or}$$

$$f(m) - PF_{opt} \geq \Big(1 - \frac{1}{\beta m}\Big)^m \cdot (f(0) - PF_{opt}), \quad \text{or}$$

$$f(m) \geq PF_{opt} - \Big(1 - \frac{1}{\beta m}\Big)^m \cdot PF_{opt} \geq (1 - e^{-1/\beta}) PF_{opt}.$$

Therefore, we have

$$\frac{PF_{opt}}{PF_A} = \frac{PF_{opt}}{f(m)} \leq \frac{1}{1 - e^{-1/\beta}}.$$ □

Theorem 3. *There is an $\frac{e}{e-1}$-approximation algorithm for m machines with running time $O(mnV)$. Moreover, for any $\epsilon > 0$, there is an approximation algorithm with approximation ratio at most $\frac{1}{1-e^{-1/\beta}} = \frac{1}{1-e^{-1/(1+\epsilon)}}$ and running time $O(\frac{mn^3}{\epsilon})$.*

Proof. The pseudo-polynomial time algorithm for a single machine is optimal; so, $\beta = 1$. By extending it to m machines, we obtain a $\frac{e}{e-1}$-approximation algorithm whose running time is $O(mnV)$.

By Lemma 2, the FPTAS can be extended to m machines to obtain an approximation algorithm with approximation ratio at most $\frac{1}{1-e^{-1/\beta}} = \frac{1}{1-e^{-1/(1+\epsilon)}}$ and running time $O(\frac{mn^3}{\epsilon})$. □

3 Profit Proportional to Processing Time

In this section, we study the case where the profit of a job J_i is proportional to its processing time; i.e., $pf_i = \alpha \cdot p_i$ for some constant α. Since α is a constant, we can scale it and consider only $\alpha = 1$. We will show that in this case, the running time of the FPTAS can be reduced to $O(\frac{n^2}{\epsilon})$. This compares favorably with the running time of the FPTAS for the arbitrary profit case, which is $O(\frac{n^3}{\epsilon})$.

Let us change the original problem to a slightly different problem: In each time frame, we set the length of the time window to be 0 and the length of the leading interval to be the length of the entire time frame; i.e., $w_i = d_i$ for each $1 \leq i \leq z$. In this new problem, a job must be finished at the end of the time frame, and each time frame can schedule at most one job. We will show that this problem can be solved optimally in $O(n^2)$ time.

Let $G(i)$ be the maximum total profit that can be obtained by scheduling the jobs whose delivery time is d_i or earlier. We will compute $G(1), G(2), \cdots, G(z)$, and the maximum total profit will be given by $G(z)$. The base case can be computed easily: $G(1)$ is the profit of the longest job that can be finished at time d_1. Assume that $G(1), G(2), \cdots, G(i-1)$ have been computed correctly, we now compute $G(i)$ as follows.

Algorithm B

$max = G(i-1)$
For each job $J_k \in J^i$

- Suppose $d_i - p_k$ is in the time frame $F_{i'}$.
- If $G(i'-1) + pf_k > max$, then $max = G(i'-1) + pf_k$.

$G(i) = max$

The correctness of the algorithm is straightforward: At most one job can be scheduled in the time frame F_i and the above procedure tries all possibilities. Moreover, for each job $J_k \in J^i$ and $d_i - p_k$ in the time frame $F_{i'}$, the maximum total profit that we can obtain is $G(i'-1) + pf_k$, since we cannot schedule any other jobs in the time frames $F_{i'}, \cdots, F_i$. The running time of the dynamic programming algorithm is $O(n^2)$, since there are at most $O(n)$ time frames and for each time frame we spend at most $O(n)$ time.

Let S be the schedule produced by Algorithm B for the modified problem. We now convert the schedule S into the schedule $\hat{S}$ for the original problem as follows. In the schedule S, we change back each time window W_i into its original

length W and we change back each leading interval L_i into its original length L. Clearly, in S, at most one job is completed in any time window. We scan the schedule S, window by window, to construct the schedule $\hat{S}$. When we scan the first window W_1, we have two cases to consider.

Case I: No job of J^1 is completed in this time window. We have two sub-cases to consider.

- **Case I(a):** No job is scheduled in this time window.
  In this case, we do nothing and scan the next time window W_2.
- **Case I(b):** There is a job, say job J_k, scheduled in this time window.
  In this case, it is easy to see that job J_k is not in J^1. Assume that job J_k is completed at time d_i. We scan the next time window W_{i+1}.

Case II: There is one job, say J_k, completed in the time window W_1. We again have two sub-cases to consider.

- **Case II(a):** $p_k \geq \frac{W}{2}$.
  If the processing time of J_k is greater than or equal to $\frac{W}{2}$, then keep the position of J_k unchanged; i.e., the completion time of J_k will be at d_1. Scan the next time window W_2.
- **Case II(b):** $p_k < \frac{W}{2}$.
  If the processing time of J_k is less than $\frac{W}{2}$, then move the job J_k as far left as possible. That is, schedule the job J_k as early as possible but keep it in the time frame F_1. Schedule the jobs of $J^1 \setminus \{J_k\}$ in the remaining space of the time window W_1 by the Largest-Processing-Time (LPT) rule until no jobs can be scheduled or the total processing time of the scheduled jobs is greater than or equal to $\frac{W}{2}$. Scan the next time window W_2.

We scan the schedule S, window by window, until the last window. Finally, we obtain the schedule $\hat{S}$.

Let PF_S be the total profit obtained by S and $PF_{\hat{S}}$ be the total profit obtained by $\hat{S}$. Clearly, we have $PF_S \leq PF_{\hat{S}}$. Let S^* be an optimal schedule and PF_{S^*} be the total profit obtained by S^*. By Lemma 1, we may assume that within each time frame, the jobs are scheduled in descending order of their processing times. We divide S^* into two schedules, say S_1 and S_2, such that S_1 contains the longest job from each time frame and S_2 contains all the remaining jobs. Figure 2 shows a schedule S^* and its subdivision into S_1 and S_2.

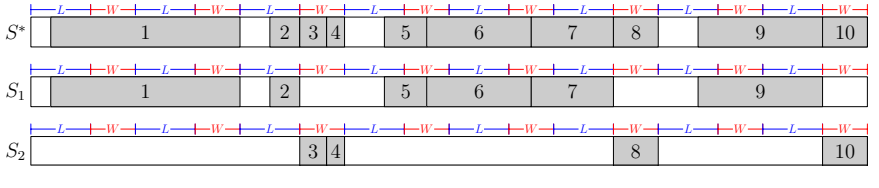


Fig. 2. Decomposition of S^* into S_1 and S_2

We consider the schedule S_1 first. Let us label all the jobs from S_1 from left to right as $J_1, J_2, \cdots, J_x$. We divide S_1 further into S_1' and S_1'' such that S_1' contains the jobs with odd labels and S_2'' contains the jobs with even labels. Figure 3 shows the schedule S_1 and its subdivision into S_1' and S_2''. In S_1', for each time frame, there is at most one job that completes in that time frame. Moreover, any pair of jobs in S_1' do not share a time window. So, S_1' is a feasible schedule for the new instance where each time window has length zero. Therefore, the total profit of S_1' is less than or equal to PF_S. Similarly, the total profit of S_1'' is less than or equal to PF_S as well. Therefore, the total profit of S_1 is less than or equal to $2 \cdot PF_S$. Since $PF_S \leq PF_{\hat{S}}$, the total profit of S_1 is less than or equal to $2 \cdot PF_{\hat{S}}$.

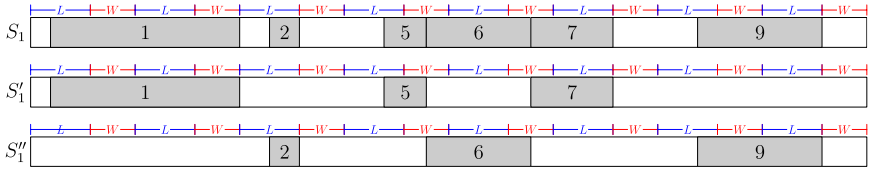


Fig. 3. Decomposition of S_1 into S_1' and S_1''

We now consider the schedule S_2 and compare it with $\hat{S}$. We have the following lemma.

Lemma 3. *Let $PF_{\hat{S}}$ be the total profit of $\hat{S}$ and PF_{S_2} be the total profit of S_2. We have $PF_{S_2} \leq 2 \cdot PF_{\hat{S}}$.*

Finally, we have the following result for the case of profit proportional to its processing time.

Theorem 4. *There is an $O(\frac{n^2}{\epsilon})$-time FPTAS for the case of profit proportional to its processing time on a single machine. The algorithm can be extended to $m \geq 2$ machines with running time $O(\frac{mn^2}{\epsilon})$ and performance bound $\frac{1}{1-e^{-1/(1+\epsilon)}}$.*

4 Conclusion

In this article we give a model of production and delivery scheduling where each job is supposed to be completed within a specified time window and the time windows are disjoint. We consider two of profits: (1) arbitrary profit, and (2) profit proportional to its processing time. In the first case, we give a pseudo-polynomial time algorithm to find an optimal schedule for a single machine. Based on the pseudo-polynomial time algorithm, we develop a FPTAS with running time $O(\frac{n^3}{\epsilon})$. In the second case, we give an improved FPTAS with running time $O(\frac{n^2}{\epsilon})$. All of our algorithms can be extended to parallel and identical machines with a certain degradation of performance bounds.

In our model, we have assumed that all time windows have the same length W and all leading intervals have the same length L. This assumption can in fact be relaxed to allow for variable window lengths and variable leading interval lengths. Our pseudo-polynomial time algorithm and polynomial time algorithm will work under this relaxation.

Our model can be extended in several directions. First, we have assumed that a job not scheduled can be discarded without any penalty. In some applications, the job that is not scheduled incurs a small penalty. We have not been able to extend our pseudo-polynomial time algorithm for this more general case. Second, we have assumed that there is no vehicle limitation at any delivery time. Suppose that at each delivery time, we can ship at most c jobs. For this problem, we have been able to extend the pseudo-polynomial time algorithm to handle this case for a single machine, but we are unable to solve the parallel machines case. Third, we have assumed that each job has a specified delivery time. A more general problem may be that each job has several specified delivery times, and the scheduler can decide which delivery time to deliver.

For future research, we believe that the above problems are worthwhile to pursue.

References

1. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating the throughput of Multiple Machines in Real-Time Scheduling. SIAM J. on Computing 31, 331–352 (2001)
2. Bilgen, B., Ozkarahan, I.: Strategic Tactical and Operational Production-Distribution Models: A Review. International J. of Technology Management 28, 151–171 (2004)
3. Chen, Z.-L.: Integrated Production and Distribution Operations: Taxonomy, Models, and Review. In: Simchi-Levi, D., Wu, S.D., Shen, Z.-J. (eds.) Handbook of Quantitative Supply Chain Analysis: Modeling in the E-Business Era. Kluwer Academic Publishers, Dordrecht (2004)
4. Chen, Z.-L.: Integrated Production and Outbound Distribution Scheduling in a Supply Chain: Review and Extensions. Working paper, Robert H. Smith School of Business, University of Maryland, College Park, MD 20742 (2006)
5. Coffman Jr., E.G., Leung, J.Y.-T., Ting, D.W.: Bin Packing: Maximizing the Number of Pieces Packed. Acta Informatica 9, 263–271 (1978)
6. Erenguc, S.S., Simpson, N.C., Vakharia, A.J.: Integrated Production/Distribution Planning in Supply Chains: An Invited Review. European J. of Operational Research 115, 219–236 (1999)
7. Garcia, J.M., Lozano, S.: Production and Delivery Scheduling Problem with Time Windows. Computers and Industrial Engineering 48, 733–742 (2004)
8. Garcia, J.M., Lozano, S., Canca, D.: Coordinated Scheduling of Production and Delivery from Multiple Plants. Robotics and Computer-Integrated Manufacturing 20, 191–198 (2004)
9. Garcia, J.M., Smith, K., Lozano, S., Guerrero, F.: A Comparison of GRASP and an Exact Method for Solving a Production and Delivery Scheduling Problem. In: Proceedings of Hybrid Information Systems: Advances in Soft Computing, pp. 431–447. Physica-Verlag, Heidelberg (2002)

10. Goetschalckx, M., Vidal, C.J., Dogan, K.: Modeling and Design of Global Logistics Systems: A Review of Integrated Strategic and Tactical Models and Design Algorithms. European J. of Operational Research 143, 1–18 (2002)
11. Ibarra, O.H., Kim, C.E.: Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. J. of ACM 22, 463–468 (1975)
12. Leung, J.Y.-T., Dror, M., Young, G.H.: A Note on an Open-End Bin Packing Problem. Journal of Scheduling 4, 201–207 (2001)
13. Sarmiento, A.M., Nagi, R.: A Review of Integrated Analysis of Production-Distribution Systems. IIE Transactions 31, 1061–1074 (1999)
14. Thomas, D.J., Griffin, P.M.: Coordinated Supply Chain Management. European J. of Operational Research 94, 1–15 (1996)
15. Yang, J., Leung, J.Y.-T.: The Ordered Open-End Bin Packing Problem. Operations Research 51, 759–770 (2003)

Fault-Tolerant Routing: k-Inconnected Many-to-One Routing in Wireless Networks

Deying Li*, Qinghua Zhu, and Huiqiang Yang

School of Information, Renmin University of China, Beijing, China
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China, MOE
deyingli@ruc.edu.cn

Abstract. This paper addresses the problem of fault-tolerant many-to-one routing in static wireless networks with asymmetric links, which is important in both theoretical and practical aspects. The problem is to find a minimum energy subgraph for a given subset and a destination node such that there are k node-disjoint paths from each node in the subset to the destination node in the subgraph. We prove that the problem is NP-hard, and propose two efficient heuristic approaches, namely, the minimum weight k node-disjoint paths based (MWkNDPB) approach and the minimum energy k node-disjoint paths based (MEkNDPB) approach. Extensive simulations have been conducted to show that proposed algorithms are efficient.

Keywords: wireless networks, fault-tolerant, k-inconnected many-to-one routing, heuristics.

1 Introduction

All-to-one or many-to-one routing is one of the important and primary communication methods in wireless networks for the data collection. In most cases, wireless networks are deployed under a harsh environment, thus wireless nodes and links are easy to experience frequent failures. However, because node or link failures have a significant impact on the performance and reliability of wireless networks, how to ensure fault tolerance all-to-one or many-to-one routing becomes a very important issue in wireless networks. On the other hand, energy efficiency is also an important issue in wireless networks since nodes are powered by batteries that may not be possible to be recharged or replaced during a mission. How to provide energy efficient fault tolerance all-to-one or many-to-one routing in wireless networks is very challenging.

There are four classes of the communication models in wireless networks: anycast; unicast; all-to-one or one-to-all communication; and many-to-one or one-to-many communication. Most of the previous studies for fault tolerant focused on anycast, unicast and all-to-one or one-to-all routing. For example, some approximation algorithms are proposed in [3,4,7] for constructing minimum power

* Corresponding author.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 483–493, 2009.

k node-disjoint paths between any two nodes. An optimal solution is constructed in [13] that gives k node-disjoint paths between the given source and destination. Two approaches are proposed in [15] for all-to-one and one-to-all communication model, namely minimum weight based approach and nearest neighbor augmentation approach. And it is proved that the minimum weight approach has k-approximation algorithms for all-to-one fault topology control (where k is the number of disjoint paths).

There are little works on the fault tolerant many-to-one routing. However it is very meaningful and practical in wireless networks. For example, the important and main tasks of wireless sensor networks are monitoring a geographical region and collecting the relevant data. We need to collect data from sensors in relevant region to sink. In this paper, we address the k-inconnected many-to-one routing problem: to construct a routing with minimum total energy in which there are k node disjoint paths from any node in a given subset of nodes to one specific node. We first prove that the k-inconnected many-to-one problem is NP-hard, and then propose two heuristics for the problem. The simulation results evaluate the efficiency of the algorithms.

2 Related Work

Previous work related to fault tolerant topology control in static wireless networks focus on the following cases: 1) k-fault tolerant anycast, 2) k-fault tolerant unicast, 3) k-fault tolerant all-to-one problem and k-fault tolerant one-to-all problem.

For minimum total power all-to-all k-fault tolerance problem, authors in [10,11,2] used the algorithm BICONNECTED-KR proposed in [5] to construct a minimum weight 2-connected subgraph as an approximation for minimum power all-to-all 2-connected problem, and analyzed its approximation ratio. [2] gave the best approximation ratio 4. [1] proposed a localized algorithm FTCBTC, which generalized the well-known Cone-Based Topology Control ([16,6]). [12,7] proposed greedy algorithms for minimum maximal power consumption for 2-node and k-node connectivity, respectively. A localized implementation of the centralized algorithm is proposed [7].

A novel polynomial time algorithm is proposed in [13] that optimally solves the minimum energy 2-link-disjoint paths problem for unicast, as well as a polynomial time algorithm called Source Transmit Power Selection (STPS) for minimum energy k node-disjoint paths problem for unicast, based on the minimum weight k node-disjoint S-D path algorithm [14]. In addition, some efficient heuristic algorithms for link/node-disjoint paths problems are presented in [13].

The minimum energy broadcast/multicast problem [8,9] is a special case of the one-to-all(one-to-many) k-fault-tolerant problem. The work in [8] proved that the minimum energy broadcast problem is NP-hard. And gave three algorithms, and one of them is $1 + 2ln(n-1)$-approximation algorithm. For minimum total power all-to-one and/or one-to-all k-fault tolerance, two approaches are proposed in [15]: minimum weight based approach and nearest neighbor augmentation approach. And they gave theoretical analysis for the proposed algorithms. The

minimum weighted based approach is a k-approximation algorithm for all-to-one k-fault tolerant topology control.

In this paper, we study the minimum energy many-to-one k-fault tolerant problem.

3 Problem Formulations

In this paper, we use the following network model. A wireless network consists of N nodes, each of which is equipped with an omni-directional antenna with a maximal transmission range r_{max}. The power required for a node to attain a transmission range r is at least Cr^{α}, where C is a constant, α is the power attenuation exponent and usually chosen between 2 and 4. For any two nodes u and v, there exists a directed edge from u to v if the Euclidean distance $d(u,v) \leq r_u$, where r_u is the transmission range of node u determined by its power level. In this paper, we consider asymmetric networks in which the existence of a directed edge from u to v does not guarantee the existence of a directed edge from v to u.

The network can be modeled by an edge-weight directed graph $G = (V, E, c)$. V is a set of N nodes. For any pair of nodes u and v, there is a directed edge from u to v if and only if $d(u,v) \leq r_{max}$. We assign a weight to a directed edge (u,v) by $c(u,v) = Cd(u,v)^{\alpha}$. In fact, since each node has the same maximal transmission range r_{max}, the graph G is a symmetric directed graph, i.e., if there is a directed edge from u to v, then there is a directed edge from v to u. But, because each node may not have the same transmission assignment, the graph induced by the assignment will be a directed graph.

Suppose H is a subgraph of G. Let $p(u)$ be the power assignment of node u, $c(u,v)$ be the cost of a directed edge (u,v), and $p(H)$ be the total energy of H, $c(H)$ be the total cost of H, then we have:

$$p(u) = \max_{(u,v)\in H} c(u,v) \tag{1}$$

$$c(H) = \sum_{(u,v)\in H} c(u,v) \tag{2}$$

$$p(H) = \sum_{u\in H} p(u) \tag{3}$$

Minimum Energy k-inconnected Many-to-One Routing Problem. Given a directed graph $G = (V, E, c)$, a root r and a subset S of nodes along with a positive integer k, find the power assignment of each node such that the sub-graph H induced by the power assignment has minimum energy, and for any $s \in S$, H contains k node-disjoint paths from s to r. We call this problem as MEkinMRP in short.

4 Algorithms

In the section, we first prove the minimum energy k-inconnected many-to-one routing problem is NP-hard. Then propose two algorithms to solve the problem.

Theorem 1. *For any $k \geq 2$, the minimum energy k-inconnected many-to-one routing problem (MEkinMRP) is NP-hard.*

Proof. We will prove it by two cases:

Case 1: $k = 2$, we transfer the vertex-cover problem to the MEkinMRP problem. Suppose $G = (V, E)$ is instance of vertex-cover problem. We construct a new directed graph $G_1 = (V_1, E_1)$ as followings:

(1) For each edge (u, v) of G, add a node s_{uv} at the middle. i.e., $\{u, v, s_{uv}\} \subseteq V_1$ and $(s_{uv}, u) \in E_1, (s_{uv}, v) \in E_1$. Assign distance 6 between u and v such that $d(s_{uv}, u) = 3$ and $d(s_{uv}, v) = 3$.
(2) Let r, a, b be three additional new nodes in G_1. Connect each node u of G to a and b with distance 1 and 2 respectively, i.e. for any $u \in V$, $(u, a) \in E_1, (u, b) \in E_1$, and $d(u, a) = 1$, $d(u, b) = 2$.
(3) Connect a and b to r with distance $d(a, r) = 1$ and $d(b, r) = 1$.

Therefore we get an edge-weight directed graph $G_1 = (V_1, E_1)$, where,

$$V_1 = V \cup \{s_{uv} | \forall e = (u, v) \in E\} \cup \{a, b, r\}$$

$$E_1 = \{(s_{uv}, u), (s_{uv}, v) | \forall (u, v) \in E\} \cup \{(u, a), (u, b) | \forall u \in V\} \cup \{(a, r), (b, r)\}$$

An example is shown in Figure 1.

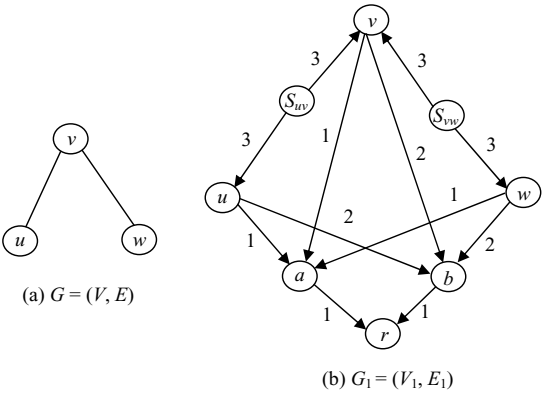


Fig. 1. An example of the transformation for $k = 2$

Set $S = \{s_{uv} | \forall (u, v) \in E(G)\}$. Then G has a vertex cover of size h if and only if there is k-inconnected routing from S to r in G_1 using power at most $|E(G)| \times 3^\alpha + (|V| - h) \times 1^\alpha + h \times 2^\alpha + 2 \times 1^\alpha$, where we suppose it need power $d^\alpha(u, v)$ from u to v.

Case 2: general k, we reduce the hitting set of hypergraph problem to the MEkinMRP problem.

The hitting set of hypergraph problem is formally represented as follows: A given hypergraph $G = (V, E)$, where V is a set of nodes, and for each $e \in E$, e

is a subset of V. Find minimum cardinality set of nodes S that covers all edges of G, i.e. every edge contains at least one node of S.

Suppose G is a hypergraph with uniform k-edge (i.e. each edge contains k nodes), we construct a new directed graph $G_1 = (V_1, E_1)$. For each edge e in hypergraph, we add a new node $v(e)$, let $v(e)$ connect each node of edge e, and assign distance 3 for these edges. Let $r, a_1, a_2, \cdots, a_{k-1}, b$ be $k+1$ additional nodes, connect each node u of G to $a_1, a_2, \cdots, a_{k-1}$ and b with distance $1, 1, \cdots, 1$ and 2, respectively. And connect $a_1, a_2, \cdots, a_{k-1}$ and b to r with distance 1.

We get an edge-weight directed graph $G_1 = (V_1, E_1)$, where

$$V_1 = V \cup \{v(e) | \forall e \in E\} \cup \{r, a_1, a_2, \cdots, a_{k-1}, b\}$$

$$E_1 = \{(v(e), u) | \forall e \in E, \forall u \in e\} \cup \{(u, a_1), (u, a_2), \cdots, (u, a_{k-1}), (u, b) | \forall u \in V\}$$
$$\cup \{(a_1, r), \cdots, (a_{k-1}, r), (b, r)\}$$

An example for this construction is shown in Figure 2.

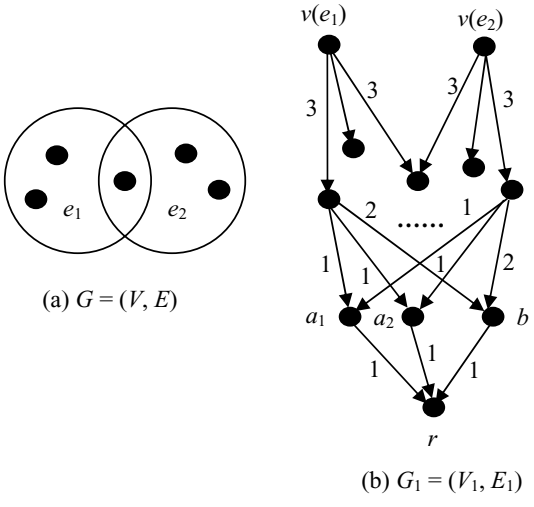


Fig. 2. An example of the transformation for general k

Set $S = \{v(e) | \forall e \in E\}$. Then G has a hitting set of size h if and only if there is k-inconnected routing from S to r in G_1 with using power at most $|E(G)| \times 3^\alpha + (|V| - h) \times 1^\alpha + h \times 2^\alpha + k \times 1^\alpha$.

Because the vertex-cover problem and the hitting set of uniform hypergraph problem are both NP-hard, the minimum energy k-inconnected many-to-one routing problem (MEkinMRP) is NP-hard, for any $k \geq 2$. □

Since the MEkinMRP is NP-hard, we need to find approximation algorithms or heuristic algorithms for this problem. In the following, we will propose two heuristic algorithms: minimum weight k node-disjoint paths based (MWkNDPB) algorithm and minimum energy k node-disjoint paths based (MEkNDPB) algorithm.

4.1 MWkNDPB Algorithm

Before introducing MWkNDPB algorithm, we first address the minimum weight k node-disjoint paths of node pair problem.

Minimum Weight k Node-disjoint Paths Problem(MWkNDP). Given an edge-weighted digraph $G = (V, E, c)$ and a pair nodes (s, r) in V, find a subgraph H of G with minimum $c(H) = \sum_{(u,v)\in H} c(u, v)$ such that H contains at least k node-disjoint paths from s to r.

In the following, we propose minimum weight k node-disjoint paths flow based (MWkNDPFB) algorithm for MWkNDP problem.

It is well known that all the classical flow algorithms are applied to solve edge-disjoint paths problem, but the MWkNDP problem is to find node-disjoint paths. Therefore we first construct a new graph $G_a = (V_a, E_a, c_a)$, and transform MWkNDP problem in G to the edge-disjoint paths problem in G_a.

Given a graph $G = (V, E, c)$, the graph $G_a = (V_a, E_a, c_a)$ is constructed as following:

(1) For each node $i \in V$, there are two nodes i_t and i_h in V_a coressponding to i. And $(i_t, i_h) \in E_a$ with $c_a(i_t, i_h) = 0$.
(2) For each edge (i, j) in G, there is an edge $(i_h, j_t) \in E_a$ with $c_a(i_h, j_t) = c(i, j)$.

Therefore, we got $G_a = (V_a, E_a, c_a)$, where

$V_a = \cup_{i\in V}\{i_t, i_h\}$;
$E_a = \cup_{i\in V}\{(i_t, i_h)\} \cup \{(i_h, j_t)|\forall(i, j) \in E\}$.

For a pair nodes (s, r) in V, there is a corresponding pair nodes (s_h, r_t) in G_a. The minimum weight k node-disjoint paths problem for (s, r) in G is transformed to the minimum weight k edge-disjoint paths problem for (s_h, r_t) in G_a, which can be modeled to the following minimal cost flow problem:

$$\begin{aligned} &\text{minimize} \sum c_a(i, j) \cdot x_{ij} \\ &\sum x_{ij} - \sum x_{ji} = \begin{cases} k & \text{if } i = s_h \\ 0 & \text{if } i \neq s_h, r_t \text{ (for any } i \in V_a) \\ -k & \text{if } i = r_t \end{cases} \\ &x_{ij} = 0, 1, \text{for any}(i, j) \in E_a \end{aligned} \tag{4}$$

Note that: we split each node $i \in V$ into two nodes i_t and i_h along with an edge (i_t, i_h) in V_a, which guarantees that the subgraph obtained from the solution of ILP (4) is node-disjoint after contracting all the (i_t, i_h) edges.

Because the relaxation of ILP has an integer solution since the ILP (4) is an integer flow problem, ILP can be solved in polynomial time. Therefore, there is an optimal algorithm for the MEkinMRP.

MWkNDPFB Algorithm for the MEkinMRP:
Input: $G(V, E, c)$, a pair of nodes (s, r) and constant k.
Output: A subgraph H, in which there are k node-disjoint paths from s to r with minimum $c(H)$.

1. Construct a new graph $G_a = (V_a, E_a, c_a)$ according G.
2. Solve the relaxation linear programming of (4) to get the optimal solution of (4).
3. Construct a subgraph of G_a by the solution of (4).
4. Contract all the (i_t, i_h) edges in the subgraph into a node, and we get the subgraph H of G.

Based on the MWkNDPFB algorithm, we propose MWkNDPB algorithm for the minimum energy k-inconnected many-to-one routing problem. The main idea of MWkNDPB algorithm is that: given the (S, r), for each $s \in S$, we invoke the MWkNDPFB to get a subgraph H_{sr}. Then we can get a subgraph $H = \bigcup H_{sr}$. H guarantees that there are k node-disjoint paths from any $s \in S$ to r.

MWkNDPB Algorithm:
Input: (G, k, r, S), where $G = (V, E, c)$ is a directed weight graph of network topology and r is the root, $S \subseteq V - r$, k is a constant integer.
Output: A subgraph H that is k-inconnected from S to r.

1. For each $s \in S$, invoke the MWkNDPFB routine to get a subgraph H_{sr}.
2. Union all the subgraphs H_{sr} got in step 1, that is $H = \bigcup_{s \in S} H_{sr}$.

4.2 MEkNDPB Algorithm

In this section, we propose another algorithm:MEkNDPB algorithm. Before introducing the algorithm, we first introduce the minimum energy k node-disjoint paths of node pair problem.

Minimum Energy k Node-disjoint Paths Problem: Given an edge-weighted digraph $G = (V, E, c)$ and a node pair (s, r) in V, find a subgraph H of G such that H contains at least k node-disjoint paths from s to r and $p(H) = \sum_{u \in H} p(u)$ is minimized where $p(u) = \max_{(u,v) \in E} c(u, v)$.

Suppose H is a subgraph which contains k node-disjoint paths from s to r. Each node in H except s and r has exactly one ingoing edge and one outgoing edge. s has at least k outgoing edges and no ingoing edge. r has at least k ingoing edges and no outgoing edge. Therefore, $p(s) = \max_{(s,v) \in H} c(s, v)$, $p(r) = 0$, and $p(i) = \max_{(i,v) \in H} c(i, v), i \in H$ and $i \neq s, r$. Then $p(H) = p(s) + \sum_{u \neq s, (u,v) \in H} c(u, v)$. The authors in [13] proposed the STPS algorithm to the minimum energy k node-disjoint paths of node pair problem by using that the value $p(s)$ must be s's some outgoing edge's weight $c(s, v)$.

STPS Algorithm:
Input: $G(V, E, c)$, a pair of nodes (s, r) and a constant integer k.
Output: A subgraph H, in which there are k node-disjoint paths from s to r with minimum $p(H)$.
Initialize: order s's M outgoing edges as $m_1, m_2, \ldots, m_M$, where $c(m_i) > c(m_j) \Leftrightarrow i > j$, where $c(m_i)$ is the weight of the edge m_i. Let $p_i(s)$ represent the current iteration transmission power of s, corresponding to i-th closest

nodes "reached" by s. Initialize $i = k$ and thus $p_i(s) = c(m_k)$. Note that starting with $i < k$ would be fruitless, as the existence of k node-disjoint paths requires at least k outgoing edges from s. Finally, let E_{min} represent the overall energy cost of the k minimum energy node-disjoint paths. Initialize E_{min} to ∞.

1. Construct a new graph G_i, where G_i is a modified version of G. Accordingly, let G_i be equal to G, except removing the outgoing edges $m_{i+1}, m_{i+2}, \ldots, m_M$ for s whose cost are bigger than $p_i(s)$, and setting the costs of all the other outgoing edges $m_1, m_2, \ldots, m_i$ of s to zero.
2. Run a minimum weight k node-disjoint paths algorithm on G_i. Let $c(H_i) = \sum_{(u,v)\in H_i} c(u,v)$ represent weight of subgraph H_i. If k-disjoint paths cannot be found by the minimum weight k node-disjoint paths algorithm, then set $c(H_i) = \infty$ and continue.
3. Evaluate the following condition: If $c(H_i) + p_i(s) < E_{min}$, then set $E_{min} = c(H_i) + p_i(s)$.
4. Increment $i = i + 1$ and correspondingly increase s's transmission power, i.e. $p_{i+1}(s) = c(m_{i+1})$. Repeat Steps 1-4 until $i > M$, at which point all relevant $p(s)$ will have been considered, and the overall minimum energy k node-disjoint subgraph H for node pair (s, r) is determined.

Based on the STPS algorithm, we propose MEkNDPB algorithm for the minimum energy k-inconnected many-to-one routing problem. The main idea of MEkNDPB algorithm is that: given the (S, r), for each $s \in S$, we invoke the STPS algorithm in [13] to get a subgraph H_{sr}, and STPS guarantees that subgragh H_{sr} is minimum energy cost for a pair nodes (s, r). Then we can get a subgraph $H = \bigcup_{s\in S} H_{sr}$. H guarantees that there are at least k node-disjoint paths from any $s \in S$ to r.

MEkNDPB Algorithm:
Input: (G, k, r, S), where $G = (V, E, c)$ is a directed weight graph of network topology and r is the root, $S \subseteq V - r$, k is a constant integer.
Output: A subgraph H that is k-inconnected from S to r.

1. For each $s \in S$, invoke the STPS routine to get a subgraph H_{sr}.
2. Union all the subgraphs H_{sr} got in step 1, that is $H = \bigcup_{s\in S} H_{sr}$.

5 Simulations

In the simulations, we focus on comparing our two heuristic algorithms MWkNDPB and MEkNDPB since the MEkinMRP problem has been little studied.

We study how the total energy cost is affected by varying three parameters over a wide range: the total number of nodes in the network(N), the number of source nodes group(M), the maximum transmission range of all nodes(r_{max}).

The simulation is conducted in a 100×100 2-D free-space by randomly allocating N nodes. The unit of r_{max} is respect to the length of one side in the

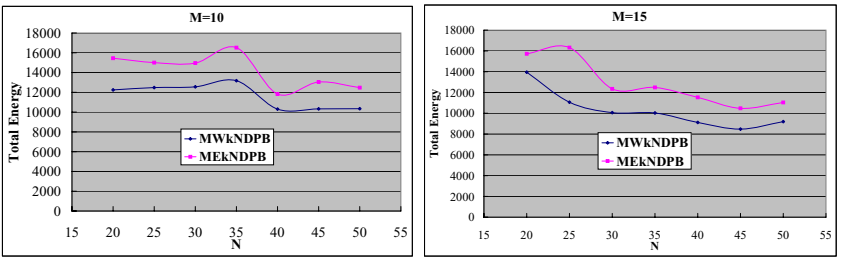


Fig. 3. $r_{max} = 1/2 \cdot (R)$

square region, i.e., when $r_{max} = \sqrt{2}R$, a node's transmission range using r_{max} covers the whole region. The power model is: $P = r^2$, where P is the transmission power and r the radius that the signal can reach. Each node has the same maximum transmission range r_{max} and the transmission range of each node can be any value between zero to r_{max}. We fix $k = 2$, i.e., each source node has at least 2 node-disjoint paths to the destination node.

We present averages of 100 separate runs for each result shown in the following figures. In each run of the simulations, for given N, M, r_{max}, we randomly place N nodes in the square, and randomly select M source nodes and a destination. Then we assign each node with the maximum transmission range r_{max}, and any topology that is not k node-connected is discarded. Then we run the two algorithms on this network.

In Fig.3, we fix $M = 10$ and $M = 15$, $r_{max} = 1/2 \cdot (R)$ while vary N. As we can see, the total energy cost of the subgraph decreases as the growth of N approximately. It is comprehensible that the number of hops between source and destination increases with N increasing, and as we know, more hops between source and destination implies smaller total energy cost approximately. It is also shown that the MWkNDPB algorithm is better than MEkNDPB algorithm all the time in this condition.

In Fig.4, we fix $N = 40$ and $N = 50$, $r_{max} = 1/2 \cdot (R)$ while vary M. As we can see, the total energy cost of the subgraph increases as the growth of M. It is intelligible that since there must be k node-disjoint paths between each source and destination, some nodes all in the subgraph should increase

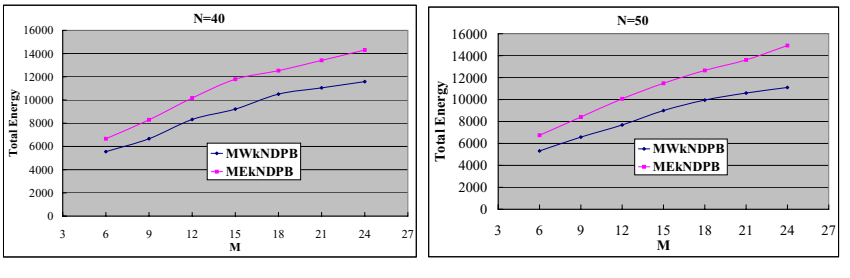


Fig. 4. $r_{max} = 1/2 \cdot (R)$

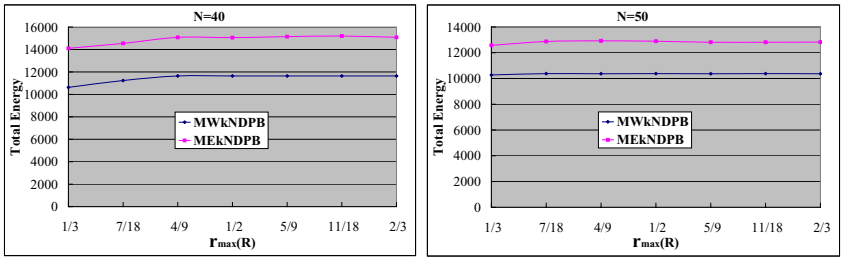


Fig. 5. $M = 15$

their transmission range or some nodes not in the subgraph should assign a transmission range to provide more paths. It is also shown that the MWkNDPB algorithm is better than MEkNDPB algorithm all the time in this condition.

In Fig.5, we fix $M = 15$, $N = 40$ and $N = 50$ while vary r_{max}. In this simulation, we keep the location of all the nodes invariably when vary r_{max}. As we can see, the total energy cost of the subgraph is invariable. It is apprehensible that since the graph with all nodes having smaller r_{max} is subgraph of the graph with bigger r_{max}, the subgraph obtained from MWkNDPB and MEkNDPB algorithms for smaller r_{max} must be a feasible subgraph for bigger r_{max}. Considering that the total energy cost is not the sum of weights of edges in the subgraph, the total energy cost of the subgraph changes a little. It is also shown that the MWkNDPB algorithm is better than MEkNDPB algorithm all the time in this condition.

6 Conclusion

We have studied the minimum energy k-inconnected many-to-one routing problem in wireless networks. Two heuristic algorithms MWkNDPB and MEkNDPB for this problem have been proposed. Extensible simulations have been conducted to compare the two heuristic algorithms. And simulation results have shown that the two heuristic algorithms have comprehensible characteristic, and MWkNDPB algorithm has better simulation evaluation than MEkNDPB algorithm.

Acknowledgement

This work was supported in part by the NSF of China under Grant No. 10671208 and 863 High-Tech Project under grant 2007AA01Z414.

References

1. Bahramgiri, M., Hajiaghayi, M., Mirrokni, V.: Fault-Tolerant and 3-Dimensional Distributed Topology Control Algorithms in Wireless Multi-Hop Networks. In: Proc. 11th IEEE Int'l Conf. Computer Comm. and Networks (2002)

2. Calinescu, G., Wan, P.: Range Assignment for High Connectivity in Wireless Ad Hoc Networks. In: Proc. Second Int'l Conf. Ad-Hoc Networks and Wireless (2003)
3. Hajiaghayi, M., Immorlica, N., Mirrokni, V.: Power Optimization in Fault-Tolerant Topology Control Algorithms for Wireless Multi-Hop Networks. In: Proc. ACM MobiCom (2003)
4. Jia, X., Kim, D., Wan, P., Yi, C.: Power Assignment for k-connectivity in Wireless Ad Hoc Networks. In: Proc. IEEE INFOCOM (2005)
5. Khuller, S., Raghavachari, B.: Improved Approximation Algorithms for Uniform Connectivity Problems. In: Proc. 27th Ann. ACM Symp. Theory of Computing (1995)
6. Li, L., Halpern, J., Bahl, P., Wang, Y., Wattenhofer, R.: Analysis of a Cone-Based Distributed Topology Control Algorithm for Wireless Multi-Hop Networks. In: Proc. 20th Ann. ACM Symp. Principles of Distributed Computing (2001)
7. Li, N., Hou, J.C.: FLSS: A Fault-Tolerant Topology Control Algorithm for Wireless Networks. In: Proc. ACM MobiCom (2004)
8. Li, D., Jia, X., Liu, H.: Energy Efficient Broadcast Routing in Static Ad Hoc Wireless Networks. IEEE Transcations on Mobile Computing 3(2), 144–151 (2004)
9. Li, D., Liu, Q., Hu, X., Jia, X.: Energy efficient multicast routing in ad hoc wireless networks. Computer Communications 30(18), 3746–3756 (2007)
10. Lloyd, E.L., Liu, R., Marathe, M.V., Ramanathan, R., Ravi, S.: Algorithmic Aspects of Topology Control Problems for Ad Hoc Networks. In: Proc. ACM MobiHoc (2002)
11. Lloyd, E.L., Liu, R., Marathe, M.V., Ramanathan, R., Ravi, S.: Algorithmic Aspects of Topology Control Problems for Ad Hoc Networks. Mobile Network Application 1-2(10), 19–34 (2005)
12. Ramanathan, R., Hain, R.: Topology Control of Multihop Wireless Networks Using Transmit Power Adjustment. In: Proc. IEEE INFOCOM (2000)
13. Srinivas, A., Modiano, E.: Minimum Energy Disjoint Path Routing in Wireless Ad-Hoc Networks. In: Proc. ACM MobiCom (2003)
14. Suurballe, J.W.: Disjoint Paths in a Network. Networks 4(2), 125–145 (1974)
15. Wang, F., Hai, T., Li, Y., Cheng, X., Du, D.: Fault-Tolerance Topology Control for All-to-One and One-to-All Communication in Wireless Networks. IEEE Transaction on Mobile Computing 3(7), 322–331 (2008)
16. Wattenhofer, R., Li, L., Bahl, P., Wang, Y.: Distributed Topology Control for Wireless Multihop Ad-Hoc Networks. In: Proc. IEEE INFOCOM (2001)

A Branch-and-Cut Algorithm for the Minimum Energy Symmetric Connectivity Problem in Wireless Networks

Xiangyong Li and Y.P. Aneja

Odette School of Business, University of Windsor
Windsor, Ontario, Canada N9B 3P4
{xiangli,aneja}@uwindsor.ca

Abstract. In this paper, we study the minimum energy symmetric connectivity problem in wireless sensor networks. This problem is to assign transmission power to each sensor node in a wireless sensor network such that all the sensor nodes are connected by bidirectional links and the overall power consumption is minimized. The determined transmission powers must support the communication session between each pair of nodes along a single-hop path or a multi-hop path. We view this feature as strong symmetric connectivity defined on a digraph. We present two mixed integer programming formulations involving an exponential number of constraints. By introducing some logical constraints, we give the first formulation. Based on this formulation, we further produce some strong cuts which result in the second stronger formulation. Using these formulations, we then devise a branch-and-cut algorithm. Computational results demonstrate that our algorithm is an efficient algorithm for the instances with up to 100 nodes.

1 Introduction

Recently wireless networks have attracted a great deal of attention for their potential for many applications in scenarios such as military target tracking and surveillance, health monitoring, natural disaster relief, and hazardous environment exploration and seismic sensing [2,12]. Generally a wireless network has no infrastructure and is composed of a number of sensor nodes. A communication session is achieved either through single-hop transmission if the recipient is within the transmission range of the source node, or by relaying through intermediate nodes [3]. As the sensor nodes operate on limited, generally irreplaceable power sources, the energy usage is an issue of common concern in wireless networks. Therefore there has been extensive research in wireless networks literature focusing on minimizing the energy consumption.

In this paper, we consider the minimum power symmetric connectivity problem (MPSC). This problem can be formally defined as follow: given a set of sensors in the plane, assign transmission power to each sensor such that the sensors are connected by using only bidirectional links, and the sum of all the

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 494–506, 2009.

transmission powers is minimized. Throughout this paper, unless otherwise specified, a sensor is equivalent to a node. As in [3], we adopt the most commonly used power-attenuation model [10] to deal with the MPSC. Based on a typical assumption that all nodes have the same transmission efficiency and detection sensitivity coefficients, the energy requirement for supporting a link between nodes i and j is defined as

$$f_{i,j} = f_{j,i} = d^{\kappa} \tag{1}$$

where κ is the attenuation coefficient, which is typically between 2 and 4 (It is set to 4 in our experiments), and d is the distance between nodes i and j. Using the model described above the power function is symmetric. However results in this paper are valid in models where $f_{ij} \neq f_{ji}$. Let w_i be a transmission range function associated with node i. A unidirectional link from node i to node j is established only if $w_i \geq f_{i,j}$; and a bidirectional link between nodes i and j is established only when $w_i \geq f_{i,j}$ and $w_j \geq f_{j,i}$.

Let $G = (N, E)$ be a complete graph where N is the set of n sensor nodes and E is the set of all undirected edges. Throughout this paper, we use (i, j) and $\{i, j\}$ to represent the directed arc from i to j, and the undirected edge between nodes i and j respectively. From the perspective of graph theory, the MPSC can be defined as follows: given a complete undirected graph G and the energy function defined in equation (1), determine the transmission ranges $\{w_1^*, w_2^*, \cdots, w_n^*\}$ such that the induced undirected graph $G_0 = (N, E_0)$ with $E_0 = \{\{i, j\} : w_i^* \geq f_{i,j} \text{ and } w_j^* \geq f_{j,i},\ i, j \in N\}$ is connected and the total energy consumption of $\sum_{i=1}^{n} w_i^*$ is minimized. Since any spanning tree of G results in a feasible solution, the MPSC can also be equivalently defined as finding a spanning tree of G with minimum power consumption.

Clementi et al. first prove that the MPSC is NP-hard [7]. The approximation algorithms and heuristics are studied for this problem. Kirousis et al. design a minimum spanning tree (MST) based 2-approximation algorithm [8]. Using the similarities between the MPSC and the steiner tree problem, Călinescu et al. decrease the approximation ratio to below 2 [5]. Blough et al. give the asymptotic bounds on the solution for the randomly generated Euclidean instances [4].

Althaus et al. formulate the MPSC in ad hoc wireless networks as a minimum power-cost spanning tree problem and presented an integer programming (IP) formulation [3]. Using this formulation, the authors develop a branch-and-cut algorithm, which is experimentally shown to be practical for instances with up to 35-40 nodes within about one hour.

Montemanni and Gambardella study exact algorithms for the MPSC [9]. Two mixed integer programming (MIP) formulations are presented. One is a single-commodity flow model where one node is chosen as the source of the flow and one unit of flow is sent from this source to every other node. Another formulation uses cut-based inequalities to guarantee the connectivity in the spanning tree. The authors give two exact algorithms where MIP formulation is directly solved by CPLEX 6.0, and possible cuts are derived from the integer solution, added to the formulation, and augmented MIP formulation is resolved by CPLEX. The experimental results in instances with up to 40 nodes show that (1) the first

algorithm requires average 2.7 hours to optimally solve the instances; (2) the second algorithm can decreased the average CPU time to about 80 seconds.

In this paper, we study the branch-and-cut algorithm for the MPSC. We cast the MPSC on the complete digraph, requiring that the induced topology of the solution must be not only strongly connected, but also connected using only bidirectional links. The remainder of this paper is organized as follows. In section 2, we present two formulations for the MPSC. A branch-and-cut algorithm is proposed in Section 3. We carry out a series of experiments to evaluate the proposed approach in Section 4. Finally, we summarize this paper in Section 5.

2 Formulations

We deal with the MPSC from the perspectives of a digraph and an undirected graph. The previous section has indicated that any spanning tree corresponds to a feasible solution to the MPSC where the connectivity requirement is defined on an undirected graph. In addition, the power associated with such a spanning tree must support the communication session between each pair of nodes. We therefore deal with a strong connectivity defined on a digraph. For this purpose, we define a digraph $G' = (N, A)$ derived from $G = (N, E)$ by defining correspondingly two arcs in A for each edge in E. Accordingly we define two types of binary variables, i.e., range variables and tree variables. The range variables $x_{i,j}$ defined in digraph G', equals 1 if the transmission power of node i can support the direct communication session from nodes i to j, and 0, otherwise. The tree variable z_e is set to 1 if edge e belongs to the induced spanning tree, which is defined in the undirected graph G. These two kinds of "connectivity" constraints are used to develop the following formulations. For any given set S of nodes with $S \neq N$, and $S \neq \emptyset$, let $(S, \overline{S})$ be the set of arcs going from S to $N \backslash S$, and $\delta(S)$ be the set of edges $e \in E$ that have one endpoint in S.

We first introduce some logical constraints that are derived from the incremental mechanism in wireless networks. As explained below, we re-label the outgoing arcs at each node. For each node i, we sort the outgoing arcs in the ascending order of transmission energy $f_{i,j}$ and label the arcs as $(i, i_1), (i, i_2), \cdots, (i, i_{n-1})$ such that $f_{i,i_1} \leq f_{i,i_2}, \cdots, \leq f_{i,i_{n-1}}$. A transmission operated by one node can also be received by all nodes within its transmission range. Accordingly we have $x_{i,i_1} = x_{i,i_2} = \cdots = x_{i,i_{j-1}} = 1$ for a solution x with $x_{i,i_j} = 1$. For each node i, we therefore can define the following logical constraints

$$x_{i,i_j} - x_{i,i_{j+1}} \geq 0 \ \ \forall j = 1, 2, \cdots, n-2. \tag{2}$$

For each node i, we can also consider another kind of constraints

$$x_{i_j,i} \geq x_{i,i_j} - x_{i,i_{j+1}} \ \ \forall j = 1, 2, \cdots, n-2. \tag{3}$$

We argue this constraint as follows. Suppose $x_{i,i_j} = 1$ and $x_{i,i_{j+1}} = 0$. It means the transmission power of node i can only support the communication from i to i_k with $k = 1, 2, \cdots, j$. The only reason is that there exists a bidirectional link

$\{i, i_j\}$ in G_0. We thus have $x_{i_j,i} = 1$. If $x_{i,i_j} = 1$ and $x_{i,i_{j+1}} = 1$, or $x_{i,i_j} = 0$ and $x_{i,i_{j+1}} = 0$, constraint (3) always holds and has no contribution. These constraints are also used in [9]. Further, for the outgoing arc (i, i_{n-1}) of node i, another logical constraint can be added as

$$x_{i_{n-1},i} \geq x_{i,i_{n-1}} \quad \forall i \in N. \tag{4}$$

This constraint has the contribution only when $x_{i,i_{n-1}} = 1$. We can argue as follows. $x_{i,i_{n-1}} = 1$ means that the power of node i support the communication from i to i_{n-1}. We also know that (i, i_{n-1}) is the longest outgoing arc of node i. Therefore the only reason for supporting such a communication is that there exists a bidirectional link $\{i, i_{n-1}\}$ in G_0 since (i, i_{n-1}) is the longest outgoing arc of node i. These constraints are used in [9] as well.

Using the above incremental mechanism and constraints, we could describe the first mixed integer programming formulation (MIP) as

$$\textbf{(MIP)} \quad \min \quad \sum_{i=1}^{n} w_i \tag{5}$$

$$\texttt{s.t.} \quad w_i \geq f_{i,j} z_e \quad \forall e = \{i, j\} \in A, \forall i \in N \tag{6}$$

$$(2) - (4) \tag{7}$$

$$x_{i,i_{n-1}} \geq 0 \quad \forall i \in N \tag{8}$$

$$\sum_{(i,j)\in(S,\overline{S})} x_{i,j} \geq 1 \quad \forall S \subset N, S \neq N, S \neq \emptyset \tag{9}$$

$$z_e \leq x_{i,j} \quad \forall e = \{i, j\} \in E \tag{10}$$

$$z_e \leq x_{j,i} \quad \forall e = \{i, j\} \in E \tag{11}$$

$$\sum_{e\in\delta(R)} z_e \geq 1 \quad \forall 1 \in R, R \subset N, R \neq N, R \neq \emptyset \tag{12}$$

$$x_{i,j} \in \{0, 1\} \quad \forall (i, j) \in A \tag{13}$$

$$z_e \in \{0, 1\} \quad \forall e \in E \tag{14}$$

$$w_i \geq 0 \quad \forall i \in N. \tag{15}$$

where constraint (6) defines the transmission power associated with each node and guarantees that $w_i^* = \max_j\{f_{i,j} : e = \{i, j\} \in E\}$. Constraint (9) ensures

that the induced graph G' is strongly connected for the purpose of communication sessions. Constraints (10) and (11) connect x-variables to z-variables. Constraint (12) indicates that the induced graph in terms of z-variables (that is, those edges with $z_e = 1$) must be connected.

In the following we give the second formulation, through which we develop some strong cuts. Using the previous incremental mechanism, we first redefine the transmission power associated with each arc. For each node i, we change the power of the outing arcs as follows: $c_{i,i_1} = f_{i,i_1}$, and $c_{i,i_j} = f_{i,i_j} - f_{i,i_{j-1}}$ for all $j = 2, \cdots, n-1$. We therefore have $f_{i,i_j} = \sum_{k=1}^{j} c_{i,i_k}$. Accordingly in the second formulation, we can replace the objective function in (5) with $\sum_{(i,j)\in A} c_{i,j}x_{i,j}$ and provide the following equivalent formulation:

$$\textbf{(BIP)} \quad \min \quad \sum_{(i,j)\in A} c_{i,j}x_{i,j} \tag{16}$$

$$\texttt{s.t.} \quad (7) - (14). \tag{17}$$

Using formulation (BIP), we present an equivalent formulation with strong cuts. The x-variables and the formulation are first rewritten in the matrix form. We first define some notations. Let T be an upper triangular non-singular matrix of order $n-1$ with $T = (t_{p,q})$ defined as:

$$t_{p,q} = \begin{cases} 1, \text{ if } p = q \\ -1, \text{ if } q = p+1 \\ 0, \text{ otherwise.} \end{cases} \tag{18}$$

Define $X_i = (x_{i,i_1}, x_{i,i_2}, \cdots, x_{i,i_{n-1}})^T$, and $X = (X = (X_1^T, X_2^T, \cdots, X_n^T)^T$. For each node, the logical constraints defined in (2) as well as constraint (8) can be written in the matrix form as $TX_i \geq 0$. Define $C_i = (c_{i,i_1}, c_{i,i_2}, \cdots, c_{i,i_{n-1}})$ for each node $i \in N$; and let E be the cut-arc incidence matrix with arcs listed in the same order as variables in X. We can rewrite formulation (BIP) as follows:

$$\textbf{(IP1)} \quad \min \quad \sum_{i=1}^{n} C_i X_i \tag{19}$$

$$\texttt{s.t.} \quad x_{i,i_1} = 1 \ \ \forall i \in N \tag{20}$$

$$TX_i \geq 0 \ \ \forall i \in N \tag{21}$$

$$EX \geq \mathbf{1} \tag{22}$$

$$(3) - (4), \text{ and } (10) - (14) \tag{23}$$

$$X = 0/1\text{-vector.} \tag{24}$$

where **1** is a column vector with n 1's. If let $Y_i = TX_i$, $i \in N$, then $X_i = T^{-1}Y_i$. Furthermore if vector X_i satisfies the logical constraints with $x_{i,i_1} = x_{i,i_2}, \cdots, x_{i,i_j} = 1$, and $x_{i,i_{j+1}}, \cdots, x_{i,i_{n-1}} = 0$, then Y_i is a unit vector with $y_{i,i_j} = 1$. Let Γ be an $|A| \times |A|$ square matrix with T along the diagonal. Γ and the inverse matrix Γ^{-1} are defined as

$$\Gamma = \begin{bmatrix} T & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & T \end{bmatrix} \text{ and } \Gamma^{-1} = \begin{bmatrix} T^{-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & T^{-1} \end{bmatrix}.$$

We thus have $Y = (Y_1^T, Y_2^T, \cdots, Y_n^T)^T = \Gamma X$, and $X = \Gamma^{-1}Y$. Further let $D_i = (d_{i,i_1}, d_{i,i_2}, \cdots, d_{i,i_{n-1}}) = C_i T^{-1}$. Accordingly we can present the following equivalent of formulation (IP1)

$$\textbf{(IP2)} \quad \min \quad \sum_{i=1}^{n} D_i Y_i \tag{25}$$

$$\texttt{s.t.} \quad e_i^T Y_i = 1 \ \ \forall i \in N \tag{26}$$

$$E\Gamma^{-1}Y \geq 1 \tag{27}$$

$$(3) - (4), \text{ and } (10) - (14) \tag{28}$$

$$Y = 0/1\text{-vector.} \tag{29}$$

where e_i^T is a row vector with $n-1$ 1's.

In the following section, we deduce the second formulation with strong cuts. Let $B = E\Gamma^{-1}$. Then constraint (27) can be rewritten as $BY \geq 1$. In the previous section, we have stated that E is the cut-arc incidence matrix in terms of x-variables. Now we consider p-th row (E_p) of E as follows:

$$E_p = (a_{p,(1,1_1)}, \cdots, a_{p,(1,1_{n-1})} \,|\, a_{p,(2,2_1)}, \cdots, a_{p,(2,2_{n-1})} | \cdots | a_{p,(n,n_1)}, \cdots, a_{p,(n,n_{n-1})}). \tag{30}$$

And we have the corresponding p-th row of B as

$$B_p = (b_{p,(1,1_1)}, \cdots, b_{p,(1,1_{n-1})} \,|\, b_{p,(2,2_1)}, \cdots, b_{p,(2,2_{n-1})} | \cdots | b_{p,(n,n_1)}, \cdots, b_{p,(n,n_{n-1})}). \tag{31}$$

We therefore have

$$b_{p,(i,i_j)} = \sum_{k=1}^{j} a_{p,(i,i_k)}, \ j = 1, \cdots n-1; \ i = 1, \cdots n. \tag{32}$$

The stronger formulation can be produced by replacing B with a new matrix $\widehat{B}$. This matrix $\widehat{B}$ is defined as follows: for each entry in B, the corresponding

entry $\widehat{b}_{p,(i,j)}$ in $\widehat{B}$ is set to 1 if $b_{p,(i,j)} \geq 1$, and 0 otherwise. We present the following stronger formulation (IP3):

$$\textbf{(IP3)} \quad \min \quad \sum_{i=1}^{n} D_i Y_i \tag{33}$$

$$\texttt{s.t.} \quad e_i^T Y_i = 1 \ \ \forall i \in N \tag{34}$$

$$\widehat{B}Y \geq 1 \tag{35}$$

$$(3) - (4), \text{ and } (10) - (14) \tag{36}$$

$$Y = 0/1\text{-vector.} \tag{37}$$

Let $\widehat{E} = \widehat{B}\Gamma$. Since $X = \Gamma^{-1}Y$, we have $\widehat{B}Y = \widehat{B}\Gamma X = \widehat{E}X$, and thus have $\widehat{E} = \widehat{B}\Gamma$. We first rewrite E_p in (30) as $(E_{p,1}|E_{p,2}|\cdots|E_{p,n})$ with $E_{p,i} = (a_{p,(i,i_1)}, a_{p,(i,i_2)}, \cdots, a_{p,(i,i_{n-1})})$. Similarly, the $p-$th rows of $\widehat{E}$ and $\widehat{B}$ are rewritten as $(\widehat{E}_{p,1}|\widehat{E}_{p,2}|\cdots|\widehat{E}_{p,n})$ and $(\widehat{B}_{p,1}|\widehat{B}_{p,2}|\cdots|\widehat{B}_{p,n})$ respectively. Now $\widehat{E}_{p,i} = \widehat{B}_{p,i}T$. The previous transformation indicates that $\widehat{B}_{p,i}$ is a row vector of size $n-1$ with a sequence of zeros (possibly empty) followed by a sequence of 1's (again possibly empty). $\widehat{E}_{p,i}$ therefore is a unit row with 1 at the position where the first 1 appears in $E_{p,i}$, if at least one coefficient 1 exists in $\widehat{B}_{p,i}$. We use the following example to show the above transformations. Given $E_{p,i} = (0,1,0,1,1,0,1,1)$, we then have the following transformations:

$$E_{pi} = (0,1,0,1,1,0,1,1) \Rightarrow B_{pi} = (0,1,1,2,3,3,4,5) \Rightarrow$$
$$\widehat{B}_{pi} = (0,1,1,1,1,1,1,1) \Rightarrow \widehat{E}_{pi} = (0,1,0,0,0,0,0,0).$$

Using $\widehat{E}$ and $X = \Gamma^{-1}Y$, we transfer formulation (IP3) to formulation (IP4) as

$$\textbf{(IP4)} \quad \min \quad \sum_{i=1}^{n} C_i X_i \tag{38}$$

$$\texttt{s.t.} \quad x_{i,i_1} = 1 \ \ \forall i \in N \tag{39}$$

$$TX_i \geq 0 \ \ \forall i \in N \tag{40}$$

$$\widehat{E}X \geq 1 \tag{41}$$

$$(3) - (4), \text{ and } (10) - (14) \tag{42}$$

$$X = 0/1\text{-vector.} \tag{43}$$

Based the fractional solution $\overline{x}$ of the linear relaxation of (IP4), we can efficiently identify any violated constraint not satisfying $EX \geq 1$ (See the separation in the next section). We thus can get a possibly strong cut using the above transformation between E and $\widehat{E}$.

3 A Branch-and-Cut Algorithm

The branch-and-cut algorithm is a branch-and-bound algorithm where cutting planes are generated throughout the search tree. Algorithm 1 outlines our branch-and-cut (B&C) algorithm for the MPSC. We first list some notations.

$\mathscr{X}^*$: the optimal solution to the MPSC;
$\mathscr{Z}^*$: the objective function value of optimal solution $\mathscr{X}$;
P_i: the LP relaxation problem at node i;
$\mathscr{X}^k$: the optimal solution to subproblem P_i;
$\mathscr{Z}^k$: the objective function value of $\mathscr{X}^k$;
P: the set of the subproblems P_i.

(1) Starting incumbent
In our B&C, we present a starting incumbent to accelerate the solution process by eliminating part of the solution space. Prim's algorithm is first implemented to find an MST in G [1]. We then set z_e to 1 if edge e appears in the MST; and 0 otherwise. x-variables in the starting solution are determined as follows. For each edge $e = \{i, j\}$ in the MST, we set $x_{i,j} = 1$ and $x_{j,i} = 1$; For each node i, $x_{i,i_1} = 1$. Furthermore, for each node and $x_{i,i_j} = 1$, we set $x_{i,i_k} = 1(k = 1, 2, \cdots, j)$, and $x_{i,i_k} = 1(k = j + 1, \cdots, n - 1)$.

(2) Separation
The separation problem is solved to identify a violated inequality, if one exists. As stated previously, our proposed formulations contain an exponential number of constraints for strong connectivity in terms of x-variables, and symmetric connectivity in terms of z-variables. Our B&C starts to solve each formulation only with a few cuts of the connectivity: for each node i, it adds $\sum_{e\in\delta(\{i\})} z_e \geq 1$ and $\sum_{(i,j)\in A} x_{i,j} \geq 1$. Based on the intermediate solutions to LP subproblems, B&C gradually identifies valid inequalities and adds them to the subproblem. Two classes of cuts are identified, i.e., constraints in (9) or (12). Furthermore, each kind of cut can be identified based on whether the current LP solution is fractional or integer but infeasible. Let $\overline{x}$ and $\overline{z}$ be the x- and z-parts in the current LP solution respectively.

If the current LP solution is fractional, we identify the the first kind of cuts by solving an overall min-cut problem in digraph $\hat{G} = (N, \hat{A})$ with $\hat{A} = A\backslash\{(i, j) : x_{i,j}$ is set to zero by branching$\}$. The capacity $u_{i,j}$ on each arc (i, j) is set as follows: $u_{i,j} = \infty$ if $x_{i,j}$ has been set to 1 in the enumeration tree; otherwise $u_{i,j} = \overline{x}_{i,j}$. We implement the approach in [6] to solve the all-pair min-cut problem defined for more general symmetric cut functions. Let π_{st} be the max-flow value from source s to sink t, and define $v(s, t) = \min\{\pi_{st}, \pi_{ts}\}$. By the

Algorithm 1. Branch-and-cut algorithm

Step 1. Initialization
input instance data and a starting incumbent $\overline{\mathscr{X}}$.
set $P = \{P_0\}$, $\mathscr{X}^* = \overline{\mathscr{X}}$.
Step 2. if $|P| = \emptyset$, **then** go to Step 3; **else** do
Step 2.1. select one subproblem P_k, and set $P \leftarrow P \backslash \{P_k\}$.
Step 2.2. Solving LP relaxation
solve subproblem P_k.
if $\mathscr{X}^k$ is infeasible, **then** prune this node and goto Step 2.
if $\mathscr{X}^k$ is integer, **then** goto Step 2.4.
if $\mathscr{X}^k$ is not integer, and $\mathscr{Z}^k > \mathscr{Z}^*$, **then** prune this node and goto Step 2.1.
Step 2.3. Separation
identify possibly violated inequalities.
if no cut is found, **then** goto Step 2.6.
else add this cut as local cut and goto Step 2.2.
end if
Step 2.4. Connectivity check for integer solution
if $\mathscr{X}^k$ is an infeasible integer solution, **then**
a valid cut is found, and added to P_k as a local cut; goto Step 2.2.
end if
Step 2.5. Pruning
if $\mathscr{Z}^k < \mathscr{Z}^*$, **then** set $\mathscr{X}^* \leftarrow \mathscr{X}^k$ and $\mathscr{Z}^* \leftarrow \mathscr{Z}^k$.
prune node k and goto Step 2.
Step 2.6. Branching
select a fractional variable $x_{i,j}$ or z_e for the branching.
check (strong) connectivity after the branching.
if the graph is (strongly) connected, **then**
create two nodes with $x_{i,j} = 0$ and $x_{i,j} = 1$, or $z_e = 0$ and $z_e = 1$ respectively.
else create one node with $x_{i,j} = 1$ or $z_e = 1$.
end if
add created subproblems to P, and goto Step 2.1.
Step 3. Output the best-found solution $\mathscr{X}^*$ and $\mathscr{Z}^*$

approach in [6], we can get $n-1$ values of $v(s,t)$ by solving $2(n-1)$ max-flow problems. Let $v_0 = \min\{v(s,t) : s \neq t, s,t \in N\}$ and $(S_0, \overline{S}_0)$ be the corresponding min-cut. If $v_0 < 1$, then we have a violated cut

$$\sum_{(i,j)\in(S_0,\overline{S}_0)} x_{i,j} \geq 1.$$

For formulation (IP4), the above cut can be further strengthened. The violated cut is added to the current subproblem and the resulting LP problem is resolved.

If the current LP solution is an integer solution with $\overline{x}$ violating constraint (9), the induced graph in terms of x-variables is not strongly connected, which results in another violated cut. We implement forward and reverse search to determine such kinds of cuts [1]. This method is to identify the maximum components that are reachable by directed paths from any specific node s, or identify the maximum components from which we can reach any given node t along directed

paths. In our implementation, both s and t are set to 1. The forward and reverse searches are implemented on an auxiliary digraph $\overline{G} = (N, \overline{A})$ with $\overline{A} = \{(i,j) \in A : \overline{x}_{i,j} = 1\}$. If the set R_0 of maximum connected components is not equal to N, we can identify a violated cut $\sum_{(i,j)\in(R_0,\overline{R}_0)\bigcap \hat{A}} x_{i,j} \geq 1$.

The second class of cuts can be identified based on $\overline{z}$, if they exist. If the LP solution is fractional, we try to get a violated cut by solving an overall min-cut problem in the undirected graph. The approach in [11] is implemented to find the overall min-cut of an undirected edge-weighted graph $\widetilde{G} = (N', \widetilde{E})$. We first describe how to create this undirected graph. Since some z-variables may have been set to 0 in the search tree, the corresponding edges cannot appear in graph $\widetilde{G}$. Further, those edges with $z_e = 1$ are not allowed to appear in the minimum cut. We first initialize $\widetilde{E}$ as $E\backslash\{e \in E : z_e$ has been set to $0\}$, and N' as N. The arc weights in $\widetilde{E}$ is set as $w_e = \overline{z}_e$. If there is at least one acnode in N', a violated cut with the value of 0 has been found, and this procedure terminates. Otherwise we continue the following merging procedure. For each edge $e = \{i,j\} \in \widetilde{E}$ where z_e is set to 1 in the search tree, these two vertices i and j are *merged*, that is, these two vertices are replaced by a new vertex, and any edges from these two vertices to a remaining vertex are replaced by an edge weighted by the sum of the weights of the previous two edges. We repeat the above merging procedure and create a new undirected edge-weighted graph $\widetilde{G} = (N', \widetilde{E})$ where N' is the new set of nodes. Based on such a graph, we implement the algorithm in [11] to get the minimum cut, of which the overall running time is $O(|N'||\widetilde{E}|+|N'|^2 \log |N'|)$. If the weight of the minimum cut is less than 1, we have identified a violated cut defined in $\widetilde{G} = (N', \widetilde{E})$. We then replace these new vertices in the cut by original vertices, which results in a violated cut defined in z-variables. If the current LP solution is an infeasible integer solution with the infeasibility in z part, we can identify the cut by implementing the above forward search for undirected edge-weighted graph $\widetilde{G} = (N, \widetilde{E})$ with $\widetilde{E} = \{e \in E : \overline{z}_e = 1\}$.

(3) Branching rule

The strategy of choosing the branching variables is fundamental to the performance of B&C. Clever choice of branching variables can significantly reduce the number of nodes explicitly examined and the computational time. In our B&C, we will compare two well-known branching rules as well as the CPLEX default branching strategy: (1) Minimum integer infeasibility; (2) Maximum integer infeasibility and (3) The default branching rule in CPLEX.

4 Experiments and Computational Results

In this section, we report our computational experience with the B&C. The algorithm is coded in C language and complied on VC++ 6.0. CPLEX 11.1 is used as the solver of the linear relaxation problem at each enumeration tree node. All experiments are carried out on a PC with Intel CPU 2.2 G under Windows XP. In the following experiments, the computation time is reported in seconds.

Table 1. Comparing branching rules

Problem	n	Max Integer Infeasibility		Min Integer Infeasibility		CPLEX Strategy	
		CPUs	Node	CPUs	Node	CPUs	Node
1	45	9.94	139	11.14	174	4.19	21
2	45	9.62	157	9.06	144	6.66	36
3	45	1404.22	33852	1297.71	33418	10.48	60
4	45	28.87	785	167.74	5228	5.32	28
5	45	15.4	377	64.32	1889	6.16	40
6	45	9.17	215	6.17	114	4.36	24
7	45	204.36	6891	-	-	6.36	33
8	50	23.19	468	42.57	991	6.75	17
9	50	28.17	511	32.53	677	7.16	24
10	50	141.46	3106	-	-	13.69	79
11	50	408.91	6306	775.7	16621	7.44	31
12	50	152.08	2302	56.85	1069	11.34	83
13	50	98.14	2299	218.78	5760	12.61	74
14	50	1.97	14	2.19	26	2.39	16
15	50	58.48	1243	463.1	11314	12.65	41

4.1 Testing Problems

We first need to generate some testing instances. The underlying networks in all instances are complete graphs. 17 sets of instances are generated and each set contains 10 instances. The number n of nodes ranges from 20 to 100 with step of 5. As in [9], we randomly choose n nodes in $[0, 10, 000]^2$ according to a uniform distribution and define the transmission power $f_{i,j}$ as $(d_{i,j})^4$, where $d_{i,j}$ is the distance between nodes i and j.

4.2 Computational Results

We conduct the first experiment to compare different branching rules. B&C using the second formulation is implemented to solve a set of 15 instances. The results are reported in Table 1. For each branching rule, we give the number of nodes explored in the search tree and the CPU time required to verify an optimal solution. "-" means that no optimal solution is obtained within 1800 seconds. The results show that the best branching rule is the default strategy of CPLEX. With this branching, B&C is able to solve the instances within 13 seconds. With other two strategies, more nodes are explored before B&C returns the optimal solutions. In the following experiments, we use this default branching rule.

The second experiment is conducted to demonstrate the significance of strong cuts in (IP4). B&C is implemented to solve formulations (MIP) and (IP) respectively. Since only small instances can be solved within acceptable CPU time using formulation (MIP), we only test B&C in 20 instances with 25 and 30 nodes. The results are reported in Table 2. For each pair of instance and formulation, we present two statistics: the number of the search tree nodes explored and the corresponding CPU time before reaching optimality. The results show that the strong cuts in (IP4) do significantly improve the performance of our B&C. Using formulation (IP4), B&C can solve all these instances within about 1 seconds. However it requires about 1132.48 seconds to deal with these instances by our approach with formulation (MIP).

Table 2. Effects of strong cuts

Problem	n	MIP		IP4	
		CPUs	Node	CPUs	Node
1	25	90.14	1950	0.13	10
2	25	405.6	4184	0.54	43
3	25	41.14	871	0.16	11
4	25	170.18	5522	0.42	13
5	25	210.37	5240	0.48	17
6	25	229.74	2521	0.3	15
7	25	38.9	791	1.11	7
8	25	149.35	2193	0.19	3
9	25	550.72	6175	0.39	12
10	25	548.33	2853	0.17	20
11	30	890.96	3989	1.31	16
12	30	2185.51	10895	1.13	21
13	30	2359.07	15274	1.11	24
14	30	5045.88	25780	1.01	15
15	30	739.72	5011	1.21	10
16	30	2856.76	12167	1.04	21
17	30	1353.07	8755	0.87	8
18	30	1401.42	6396	0.33	5
19	30	1918.39	10798	1.3	22
20	30	1464.32	9395	2.15	20
Average		1132.48	7038	0.77	16

Table 3. Computational results on larger instances

Set	n	Node	Computational times		
			Min	Max	Avg.
1	55	53	6.81	19.68	13.28
2	60	88	8.56	63.41	22.40
3	65	129	5.52	101.25	44.24
4	70	185	19.86	148.25	85.27
5	75	191	29.59	209.92	101.70
6	80	458	40.53	541.84	266.87
7	85	227	66.83	227.88	142.54
8	90	386	65.27	669.17	278.02
9	95	372	98.58	727.22	361.79
10	100	325	178.85	1058.83	413.60

Using formulation (IP4), we further demonstrate the performance of B&C in larger instances in other 10 sets. The result are reported in Table 3. For ten instances in each set, we report the average number of nodes explored, the minimum (Min), maximum (Max), and the average (Avg.) of computational times required to find and verify an optimal solution. It takes, on the average, 414 seconds to optimally solve the instances with 100 nodes.

5 Conclusion

This paper studies the minimum energy topology problem in wireless networks. This problem is to assign transmission powers to the sensor nodes so that the energy consumption is minimized while ensuring the network connectivity by bidirectional links. We develop two formulations for this problem. The first formulation contains some logical constraints derived from the general incremental

mechanism. Based on the first formulation, we then generate some strong cuts and thus present a stronger formulation. We further develop a branch-and-cut algorithm, of which the performance is verified by a series of experiments.

References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Inc., Upper Saddle (1993)
2. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless Sensor Networks: a Survey. Comput. Networks 38, 393–422 (2002)
3. Althaus, E., Călinescu, G., Măndoiu, I., Prasad, S., Tchervenski, N., Zelikovsky, A.: Power Efficient Range Assignment in Ad-Hoc Wireless Networks. In: IEEE Wireless Communications and Networking Conference (WCNC), pp. 1889–1894. IEEE Press, New York (2003)
4. Blough, D., Leoncini, M., Resta, G., Sant, P.: On the Symmetric Range Assignment Problem in Wireless Ad Hoc Networks. In: 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002), pp. 71–82. Kluwer Academic Publishers, Dordrecht (2002)
5. Călinescu, G., Măndoiu, I., Zelikovsky, A.: Symmetric Connectivity with Minimum Power Consumption in Radio Networks. In: 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002), pp. 119–130. Kluwer Academic Publishers, Dordrecht (2002)
6. Cheng, C., Hu, T.: Ancestor Tree for Arbitrary Multi-Terminal Cut Functions. Ann. of Oper. Res. 33, 199–213 (1991)
7. Clementi, A., Penna, P., Silvestri, R.: On the Power Assignment Problem in Radio Networks. Technical Report TR00-054, Electronic Colloquium on Computational Complexity (ECCC) (2000)
8. Kirousis, L., Kranakis, E., Krizanc, D., Pelc, A.: Power Consumption in Packet Radio Networks. Theoret. Comput. Sci. 243, 289–305 (2000)
9. Montemanni, R., Gambardella, L.: Exact Algorithms for the Minimum Power Symmetric Connectivity Problem in Wireless Networks. Comput. & Oper. Res. 32, 2891–2904 (2005)
10. Rappaport, T.: Wireless Communications: Principles and Practices. Prentice Hall, Inc., Upper Saddle (1996)
11. Stoer, M., Wangner, F.: A Simple Min-Cut Algorithm. J. of ACM 44, 585–591 (1997)
12. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Networks 52, 2292–2330 (2008)

Minimum Energy Broadcast Routing in Ad Hoc and Sensor Networks with Directional Antennas

Zheng Li and Deying Li*

Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China, MOE
School of Information, Renmin University of China, Beijing 100872, China
deyingli@ruc.edu.cn

Abstract. In this paper we discuss minimum energy broadcast routing with directional antennas in ad hoc and sensor networks. We assume that the network consists of sensor nodes whose antennas are *Switched beam* directional antennas. The problem of our concern is: given a set V with n nodes and each node v_i has $l(i)$ transmission *directions* (i.e. the antenna sectors) and a broadcast request sourced at s, how to find a broadcast tree rooted at s and spanning all nodes in V such that the total energy is minimized. This problem involves with the choice of transmitting nodes and their transmission *directions*, which is NP-hard. We propose a $|V|$-approximation algorithm and one heuristic for the problem. Extensive simulations have demonstrated the efficiency of our algorithms.

Keywords: ad hoc and sensor networks, energy efficient, broadcast, directional antenna, approximation algorithm.

1 Introduction

With the emergency of the smart antennas (i.e. directional antennas) technology, using the smart directional antennas to improve network performance becomes more and more popular in recent years. Previous researches have shown that the application of directional antennas can further reduce the energy consumption and the radio interference, improve the throughout [3,9,11]. As a result, the use of directional antennas has a great potential in wireless ad hoc and sensor networks.

Broadcasting is an important function in applications of wireless ad hoc and sensor networks. Most of the existing works on energy efficient broadcast/multicast routing focused on omni-directional antenna. In this paper, we consider the energy efficient broadcast routing problem in wireless ad hoc and sensor networks with directional antennas model. Note that the directional characteristic discussed in this paper is from the viewpoint of transmitting, but not from receiving (i.e. the reception beam is omni-directional).

There are two techniques used in smart directional antenna systems: *switched beam* and *steerable beam* [9,11]. The *Switched beam* directional antennas model

* Corresponding author.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 507–518, 2009.

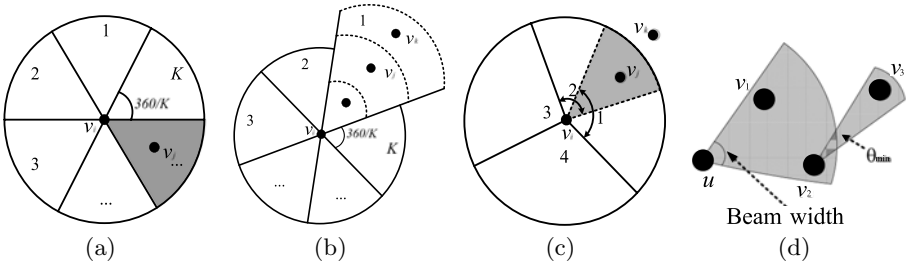


Fig. 1. Some illustrations of directional antennas model

divides the transmission range of each node into different sectors. Each node can switch on one or several sectors for transmission, as shown in Fig.1(a). The *steerable beam* directional antennas model, as shown in Fig.1(d), allows that the antenna orientation of each node can be steered to any desired direction and the beam-width can be adjusted from θ_{min} to 360^0, where θ_{min} is a threshold for communication. This paper uses the *Switched beam* directional antennas model.

In this paper, we address the *minimum energy broadcast routing with directional antennas problem*(D-MEB), in which each node equips a number of antennas sectors. The problem needs to determine a set of transmitting nodes and which *directions* of each transmissing node should be switched on. We use a general antennas model and remove some assumptions about the directional antenna model in [12,10], and assume each node has a switched beam directional antenna and different sectors can be irregular, overlapping and have different shapes. We consider asymmetric networks for removing the assumption [4] that links are bidirectional. The only constraint is that each antenna sector must have a fixed size and shape (See the Fig.1c). This is more practical in real systems because the radio performance may degenerate dramatically, i.e. although the antenna sectors are identical in idealizing model, their transmission range may become very different in practice.

The energy efficient broadcasting/multicasting with switched beam directional has also been studied in [12,10,4,8]. However, most of their algorithms are heuristics without any performance guarantee. And the author in [12] proposed an $O(\log^2 n)$-approximation algorithm but its time complexity is quasi-polynomial time $(O((Kn)^{3\log n})$, where n and K are the numbers of nodes in network and their directions. In this paper, we propose an approximation algorithm with a provably performance ratio n in polynomial time. And also propose another heuristic with lower time complexity and, from simulation results, we can show that the performances of our algorithms are better than the best heuristic in [12] and the BIP-based algorithm [10]. To our best knowledge, this is the first paper proposing an approximation algorithm with a provably performance ratio in polynomial time for the D-MEB problem.

The rest of the paper is organized as follows: In Section 2, we briefly summarize some related work. Section 3 describes the directional antennas model, the

network model and problem specification. Two algorithms for the D-MEB problem are proposed in Section 4. Simulation results about proposed algorithms are exhibited in Section 5. Finally, we give a conclusion in Section 6.

2 Related Work

Most of the previous studies on energy efficient broadcast/multicast have focused on omni-directional antennas. Different from the scenarios of omni-directional antennas, using directional antennas can further reduce the energy consumption while increasing the complexity of the problems [9]. For the *steerable beam* directional antennas model, the most famous and initial work was produced in [14], where two heuristic algorithms for broadcast routing problem: RB-BIP and D-BIP, were proposed as extensions of BIP algorithm [13]. Guo *et. al.* in [5] constructed the MILP (Mixed Integer Linear Programming) for the multicasting with directional antennas model. Li *et. al.* in [7] proposed a greedy heuristic for the broadcasting problem with directional antennas model. Cartigny *et. al.* [1] also developed the localized algorithms D-RBOP and A-DLBOP for the broadcasting with directional antennas model.

For the *switched beam* directional antennas model, Tang *et. al.* [12] extended the work in [6] to directional antennas scenario, and presented an $O(\log^2 n)$-approximation algorithm with $O((Kn)^{3\log n})$ time complexity. The authors also proposed several heuristics for D-MEB problem. Roy *et. al.* [10] investigated the MEB problem under a wide spectrum of directional antenna models and showed that the MEB problem under each of the antenna models is NP-complete. In addition, a heuristic based on BIP algorithm was proposed for the MEB problem under each directional antenna model. The work in [4] assumed that the shape and size of each sector must be fixed and wireless links in the network must be bidirectional links (i.e. the network is symmetric). The authors presented a distributed broadcasting protocol DSP, which is a nontrivial generalization of their previous work based on omni-directional antennas model. Kang and Poovendran [8] extended their GPBE algorithm to solve the minimum energy broadcast problem with directional antennas. It was assumed that each antenna sector's radius can be adjusted from 0 to R_{max} (R_{max} is the longest Euclidean distance between two nodes in networks) and the beam-with of each sector is fixed, as shown in Fig.1b.

3 Network Models and Problem Specification

In this section, we first introduce the *switched beam* directional antennas and network model. Then, we formally formulate the *minimal energy broadcast routing with directional antennas problem* (D-MEB) and prove that this problem is NP-Complete.

For the coherence of terminology, we use *directions* to denote antenna sectors in following discussion. Using such directional antennas model, only nodes located within one node's *directions* can receive the signal sent by the node.

We adopt the following notations throughout the paper.

- V: the set of network nodes.
- n: the number of network nodes, where $n = |V|$ and we will use $|V|$ and n interchangeably to denote the number of nodes in the following.
- $l(i)$: the number of *directions* of node v_i in V.
- d_{ij}: the j^{th} *direction* of the i^{th} node, $1 \leq i \leq n$, $1 \leq j \leq l(i)$. In addition, the shape and size of each *direction* are fixed. *Directions* can also be overlapping and irregular. Switching on a *direction* will lead to corresponding transmission power consumption of a node.
- w_{ij}: the transmission power consumption of node v_i switched on its j^{th} *direction*, $1 \leq i \leq n$, $1 \leq j \leq l(i)$. The *switched beam* directional antenna model allows that each node can transmit message by using multiple *directions*.
- $N^+(d_{ij})$: the set of nodes within transmission range of the j^{th} *direction* of node v_i. Since *directions* of a node maybe overlapping, a node may locate in several *directions*.

For a broadcast request $(s, V - \{s\})$, let T be a broadcast tree rooted at s. There are two kinds of nodes: the non-leaf nodes in T that need to transmit/relay messages and the leaf nodes in T that only receive message. We assume that a node costs energy only when it does transmissions. Let $RT(T)$ denote the set of transmitting nodes. Let $ON(i)$ denote IDs of v_i's all directions switched on in T. The power consumption of node v_i in a communication session is formulated as:

$$p(v_i) = \sum_{j \in ON(i)} w_{ij} \tag{1}$$

The total power of broadcast routing tree can be represented as:

$$C(T) = \sum_{v_i \in RT(T)} p(v_i) \tag{2}$$

Where the power consumption $p(v_i)$ is defined by formula (1).

The *Minimum energy broadcast routing with directional antennas problem*(D-MEB) can be formally represented as following: Given a broadcast request $(s, V - \{s\})$, and each node v_i equips a switched beam directional antenna, to determine the set of transmitting nodes, $RT(T)$, and $ON(i)$ of node i in $RT(T)$ such that the broadcast tree induced by $RT(T)$ and $ON(i)$ is rooted at s and spans all nodes in V and the total energy defined in (2) is minimized.

Theorem 1. *The D-MEB problem is NP-hard.*

Proof. It is known that when each node has only one direction and sector is a disk, the D-MEB problem becomes the MEB problem [6]. Since the MEB problem is special case of D-MEB problem, and the MEB problem is NP-Hard [6], therefore the D-MEB problem is NP-hard. □

4 Algorithms for D-MEB Problem

In this section, we will propose a $|V|$-approximation algorithm and one heuristic for D-MEB problem. Firstly, we construct a directed, edge-weighted auxiliary graph $G_A(N, E_A)$ as follows. For each node v_i $(1 \leq i \leq n)$ in V, we also use vertex v_i in G_A represents original network node v_i and then, vertices d_{ij}, $j = 1, 2, \ldots, l(i)$ represent v_i's directions. As shown in Fig. 2, arc(v_i, d_{ij}) means that network node v_i switches on its *direction* j to transmit message, and weight assigned to arc(v_i, d_{ij}) is the power consumption w_{ij}. If v_k is located in the transmission sector of v_i's *direction* j, there is a directed edge(d_{ij}, v_k) from d_{ij} to v_k. The weight of arc (d_{ij}, v_k) is assigned to 0. For the sake of convenience, vertex v_i is called as the network vertex, vertex d_{ij} is called the direction vertex, which is derived from v_i.

We get an edge-weighted directed graph $G_A = (N, E_A, w)$, where $N = V \cup \{d_{ij} | 1 \leq i \leq n, 1 \leq j \leq l(i)\}$, $E_A = \{(v_i, d_{ij}) | 1 \leq i \leq n, 1 \leq j \leq l(i)\} \cup \{(d_{ij}, v_k) | v_k \text{ is located in } v_i\text{'s } j\text{th direction}\}$, $w((v_i, d_{ij})) = w_{ij}$ and $w((d_{ij}, v_k)) = 0$ for any i, j, k. For each direction vertex, it has only one in-arc which is from its network node. Having the edge-weighted, directed auxiliary graph $G_A(N, E_A)$, any broadcast request $(s, V - \{s\})$ in original network is transformed to multicast request in G_A, we still denote $(s, V - \{s\})$ as multicast request in G_A.

The minimum energy broadcast routing with directional antennas problem is equivalently transformed to the minimum edge-weight directed Steiner tree in $G_A(N, E_A)$ rooted at s and spanning all nodes in $V - \{s\}$. When a Steiner tree T in $G_A(N, E_A)$ is got, using information provided by T, we can determine the transmitting nodes set and which *directions* should be switched on for each transmitting node to get a broadcast tree for D-MEB problem. This can be done as follows: for each transmitting node v_i in T, if there is a directed edge (v_i, d_{ij}) in T, then the *direction* j of v_i should be switched on.

For the minimum edge-weight directed Steiner tree problem, it was well known that there is $O(|D|^{\varepsilon})$-approximation algorithm for any $0 \leq \varepsilon \leq 1$ with time complexity $O(n^{3/\varepsilon}|D|^{1/\varepsilon})$, where n and D are the number of graph vertices and Steiner terminals respectively [2]. Therefore, we can use the algorithm in [2] to get an approximation algorithm for D-MEB problem in the above auxiliary

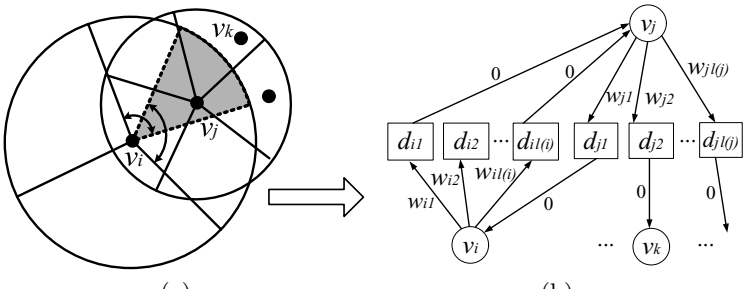


Fig. 2. The transformation from real network to the auxiliary graph

graph G_A (i.e. $O(n^{\varepsilon})$-approximation algorithm in time $O(n^{3/\varepsilon}k^{1/\varepsilon})$, where n is the number of nodes in network and $k = \max\{l(i), i = 1, 2, \ldots, n\}$. But its time complexity is very high; we will propose other algorithms with lower time complexity. In addition, when ε is set to 1, this approximation algorithm [2] degenerates to the trivial SPT algorithm (i.e. $O(n)$-approximation algorithm in time $O(n^3k)$) and, by simulation results, we can demonstrate our algorithms are better than the SPT algorithm.

4.1 A Shortest Path-Based Approximation Algorithm

In this subsection, we propose a Shortest Path-based Greedy algorithm for D-MEB problem by using the specialty of graph $G_A(N, E_A)$ and then, prove the correctness and efficiency of this algorithm. For the conveniency of description, in the following, we use (s, B) to denote a broadcast request in original network, where s is the source node and $B = V - \{s\}$.

According the property of auxiliary graph $G_A(N, E_A)$, we know: each *direction* vertex d_{ij} has only one in-arc which is from its network node and $N^+(d_{ij})$ is a subset of B for a request (s, B), where $N^+(d_{ij}) = \{v_k|(d_{ij}, v_k) \in E_A\}$. In addition, the weight of all arcs (d_{ij}, v_k) in E_A is zero. This property means that each direction node d_{ij} derived from a network node v_i can cover $|N^+(d_{ij})|$ nodes in B simultaneously without additional power cost (node d_{ij} covers node v_k, if and only if there exists an arc (d_{ij}, v_k) in E_A). We use $N(P)$ to denote the set of network nodes covered by a path P, $N(P) = \cup_{d_{ij} \in P} N^+(d_{ij})$ and $N(P) \subseteq B$, where $B = V - \{s\}$. We give a definition:

Definition 1 (Power Density). *Let $G_s(N_s, E_s)$ is a subgraph of G_A and $C(G_s)$ is total energy cost of G_s. $N(G_s) \subseteq B$ is a set of network nodes covered by G_s, i.e. for each node v_k in $N(G_s)$, there exists a direction node d_{ij} in N_s and an arc (d_{ij}, v_k) in E_s. The* Power Density *of G_s, denoted by $D(G_s)$, is $C(G_s)/|N(G_s)|$.*

Obviously, the power density of the optimal directed Steiner tree is minimal. Motivated by this fact, we can construct a directed Steiner tree T on $G_A(N, E_A)$ in this way: Initially, T only contains source node s. At each iterative step, finding a shortest path P from T to a direction vertex not in T to construct a subtree $T_{sub}(P)$, which induced by P and the network nodes in $B-T$ covered by P, with minimal power density. Then merging subtree $T_{sub}(P)$ into T. Since the power cost of arc from a direction vertex to a network node is zero, $C(T_{sub}(P)) = C(P)$. This Shortest Path-based Minimum Power Density algorithm can be formally described as follows:

Theorem 2. *Algorithm 1 is an approximation algorithm with ratio $|V|$ and its time complexity is $O(k^2n^3)$, where $k = \max\{l(i), i = 1, 2, \ldots, n\}$.*

Proof. It is easy to know that Algorithm 1 can correctly get a broadcast tree sourced at s.

Let T_{opt} be optimal directed Steiner tree rooted at s and spanning all nodes in $B(B = V - \{s\})$ in auxiliary graph G_A. Let $T_1, T_2, \ldots, T_r$ denote the subtrees

Algorithm 1. Shortest Path-based Minimum Power Density Greedy Algorithm

$C(P)$: total cost of a path P.
$N(P)$: set of network nodes covered by path P.
U: a set of destinations uncovered. Initially, $U = B$.
$SP(T, d_{ij})$: a shortest path such that $C(SP(T, d_{ij}) = \min\{C(SP(t, d_{ij})) \mid \forall\ t \in T\}$, where d_{ij} is a direction vertex not in T.
$T_{sub}(P)$: a subtree induced by P and the network nodes in U covered by P.
For improving efficiency of our algorithm, the *power density* of subtree $T(P)$ is modified to: $D(T(P)) = \frac{C(P)}{|N(P) \cap U|}$.
Input: n nodes in V, each of them equipped with a *switched beam* directional antenna, a broadcast request (s, B), where $B = V - \{s\}$.
Output: a broadcast routing tree rooted at s and spanning all nodes in B.

1: Initially, construct an edge-weighted directed auxiliary graph $G_A(N, E_A)$, and the broadcast request (s, B) in original network is transformed to a multicast request in G_A;
2: Let $T = \{s\}$ and $U = B$
3: **while** $U \neq \emptyset$ **do**
4: Find a shortest path $SP(T, d_{ij})$ with minimum power density $D(T_{sub}(SP(T, d_{ij})))$ from T to $\forall d_{ij} \notin T$;
5: Merge T and $T_{sub}(SP(T, d_{ij}))$ into a new tree T;
6: $U = U \setminus N(SP(s, d_{ij}))$.
7: **end while**
8: Switch on the *directions* for each non-leaf network node (i.e. relay node) to transmit broadcast message by using the information provide by T.

obtained at r iterative steps of algorithm 1 respectively and $T_a = T_1 \cup T_2 \cup \ldots \cup T_r, r \leq |B|$. From the property of graph G_A discussed above, we know that $C(T_1), C(T_2), \ldots, C(T_r)$ are equal to corresponding shortest paths' energy cost $C(SP_1), C(SP_2), \ldots, C(SP_r)$ respectively. Clearly, $C(T_a) = C(T_1) + C(T_2) + \ldots + C(T_r) \leq r \max\{C(SP_i) | 1 \leq i \leq r\} \leq |B| \times \max\{C(SP_i) | 1 \leq i \leq r\}$. Additionally, because $SP_i, \forall i$ is the shortest path from some s to other node d in B, its cost is not longer than the cost of path from s to d in T_{opt}. Therefore, $C(T_a) \leq |B| \times \max\{C(SP_i) | 1 \leq i \leq r\} \leq |B| C(T_{opt})$ and then Algorithm 1 is $|V|$-approximation.

The while-loop has at most n iterations. For each loop, computing the shortest path $SP(T, d_{ij})$ from T to all direction nodes takes time $O((nk + n)^2)(k = \max\{l(i) | i = 1, 2, \ldots, n\})$, in which, we can set weight of arcs in T to zero, and finding the shortest paths from s to all other nodes to get. Finding subtree $T(SP(s, d_{ij}))$ with minimal power density takes time $O(nk)$. Thus, the while-loop takes time $O(k^2 n^2)$. Therefore, the whole algorithm ends in time $O(k^2 n^3)$ in the worst case. □

4.2 A Heuristic Algorithm

In this subsection, we will propose a simpler heuristic algorithm whose time complexity is $O(kn^2)$, where $k = \max\{l(i), i = 1, 2, \ldots, n\}$. According the property

of graph $G_A(N, E_A)$, we have known: each *direction* vertex d_{ij} has only one in-arc which is from corresponding network node, and $N^+(d_{ij})$ must be a subset of B ($B = V - \{s\}$). We also can observe that subgraph $G_{ij}(N_{ij}, E_{ij})$, where $N_{ij} = \{v_i, d_{ij}\} \cup N^+(d_{ij})$ and $E_{ij} = (v_i, d_{ij}) \cup \{(d_{ij}, v_k) | v_k \in N^+(d_{ij})\}$, is the basic block of any Steiner tree on G_A. In other words, an optimal directed Steiner tree on G_A must be composed by such subgraphs. From definition 1, we know that an optimal directed Steiner tree has minimum *power density*. This fact motivates us to construct a directed Steiner tree by minimizing local power density. Clearly, subgraph $G_{ij}(N_{ij}, E_{ij})$ is a tree rooted at network node v_i and all its leaf nodes are the network nodes in B. We use subtree T_{ij} and $L(T_{ij})$ to denote the subgraph $G(N_{ij}, E_{ij})$ and its leaf nodes set respectively. Let $Son(v_i)$ be the set of all subtrees T_{ij} rooted at v_i. We can know $C(T_{ij}) = w_{ij}$, where w_{ij} is the weight of arc (v_i, d_{ij}).

Suppose U is uncovered set. We let U be B initially. For improving efficiency of our algorithm, T_{ij}'s power density $D(T_{ij})$ can be modified as $\frac{C(T_{ij})}{|L(T_{ij}) \cap U|}$.

A greedy algorithm can be implemented as follows. Initially, $T = \{s\}$. At each iterative step, T is grown by a subtree T_{ij} with minimal power density in $Son(v_i)$ corresponding to each network node v_i in T. This operation is repeated until U becomes empty, which means all nodes in B have been covered by T. We formally describe this algorithm as follows:

Algorithm 2. Minimum Local Power Density Algorithm

$L(T)$: the leaf node set of tree T.
$Son(v_i)$: a set of all subtrees T_{ij} rooted at network node v_i.
U: a set of destinations uncovered. Initially, $U = B$.
$D(T_{ij}) = \frac{C(T_{ij})}{|L(T_{ij}) \cap U|}$ is called as the *modified power density* of T_{ij}.
Input: n nodes in V, each node equips a *switched beam* directional antenna, a broadcast request (s, B), where $B = V - \{s\}$.
Output: a broadcast tree rooted at s and spanning all nodes in B.

1: Initially, construct an edge-weighted directed graph $G_A(N, E_A)$, and the broadcast (s, B) in original network is transformed to a multicast request in G_A;
2: Let $T = \{s\}$ and $U = B$
3: **while** $U \neq \emptyset$ **do**
4: Find a T_{ij} with minimum *modified power density* in $\cup_{v_i \in T} Sons(v_i)$;
5: Merge T and T_{ij} into T and pruning redundant arcs;
6: $U = U \setminus L(T_{ij})$.
7: **end while**
8: Switch on the *directions* for each non-leaf network node (i.e. relay node) to transmit broadcast message by using the information provided by T.

Clearly, the while-loop has at most n iterations. At each loop, finding the subtree with minimum power density at most take $O(kn)$ time in the worst case. Thus the time complexity of algorithm 2 is $O(kn^2)$, where $k = \max\{l(i) | i = 1, 2, \ldots, n\}$.

5 Simulations

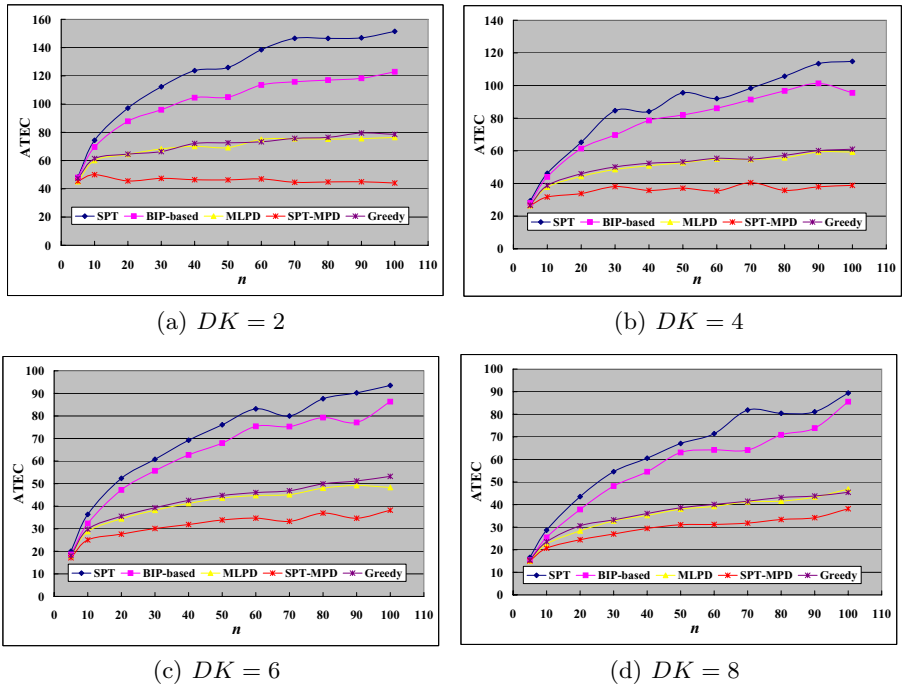
In this section, we evaluate the performance of our algorithms via simulations. We compare the performance of our SPT-MPD (Shortest Path-based Minimum Power Density algorithm) and MLPD (Minimum Local Power Density algorithm) algorithms with the SPT, BIP-based algorithm [10] and the Greedy algorithm, which is the best heuristic in [12]. We use the Average Total Energy Cost (ATEC) of computed trees as the performance metric. The simulations were carried out on our own simulation environment created by ANSI C.

We study how the total energy cost is affected by varying three system parameters over a wide range: the total number of nodes (n) in the network, the average number of directions of each node (DK) and the average length of transmission radii of all nodes' *directions* (R). The simulation is conducted in a 1000×1000 2-D free-space by randomly allocating n nodes. The power model of each *direction* is: $P = C\theta/360 \times r^{\alpha}$, where θ is the beam-width of each *direction*, r is the length of transmission radius and the constant C and α are set to 0.0001 and 2 respectively in all simulations. The number of *directions* of each node is generated form a Gauss distribution with mean and variance $(DK, 2)$ and must be more than 1. For each node, the radius r and beam-width θ of each its *direction* are generated from a Gauss distribution with mean and variance (E, V) equal to (R, R) and $(360/DK, 180/DK)$ respectively.

First, we present averages of 100 separate runs for each result shown in the following figures. In each run of the simulations, for given n, we randomly place n nodes in the square, and randomly select a source node. Network topology got by switching on all *directions* of each node, which is not connected from source node to all other nodes, is discarded. Then we run the five algorithms on this network.

In Fig.3, we fixed R, DK while vary n. All four subfigures in Fig.3 are obtained by setting $R = 600$. As we can see, our algorithm SPT-MPD is the best in all algorithms. With the growth of n, both average total energy costs of Greedy, MLPD and SPT-MPD are smaller than SPT and BIP-based algorithms dramatically. The performance of the Greedy and our algorithm MLPD are similar and more effective than SPT and BIP-based algorithms, both the energy cost and the time complexity. In addition, the average total energy cost of broadcast routing increases as the growth of n in most subfigures. However, the SPT-MPD is an exception in Fig.3(a). We find out that, when DK is very small (< 3), the layer of the broadcast tree constructed by SPT-MPD algorithm in the auxiliary graph become less with the increment of network density. It will decrease the total energy cost of the broadcast tree.

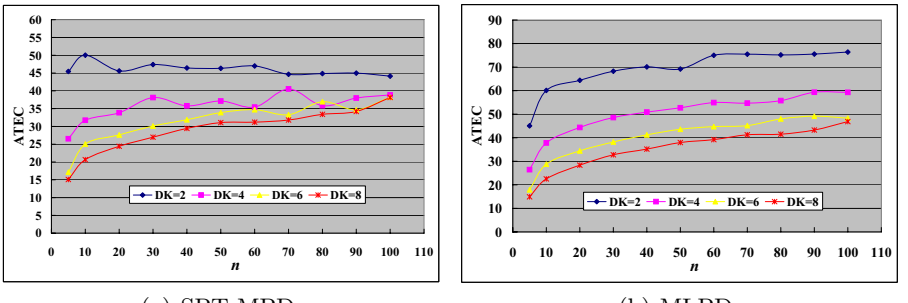
In Fig.4, we fixed R while vary n, we compare $DK = 2$, 4, 6 and 8 for SPT-MPD and MLPD. As we can see, the total energy cost decreases with the growth of *directions* number DK for the two algorithms. This is because when DK becomes more, each node can switched off more *directions* which not cover any other nodes, and the total energy cost is less. Some anomalistic points of these curves in Fig.4(a) are acceptable since the network is different among every running when the n is varying. From the figures, we also get fact that the energy saved is more when the number of sectors increases.



(a) $DK = 2$ (b) $DK = 4$

(c) $DK = 6$ (d) $DK = 8$

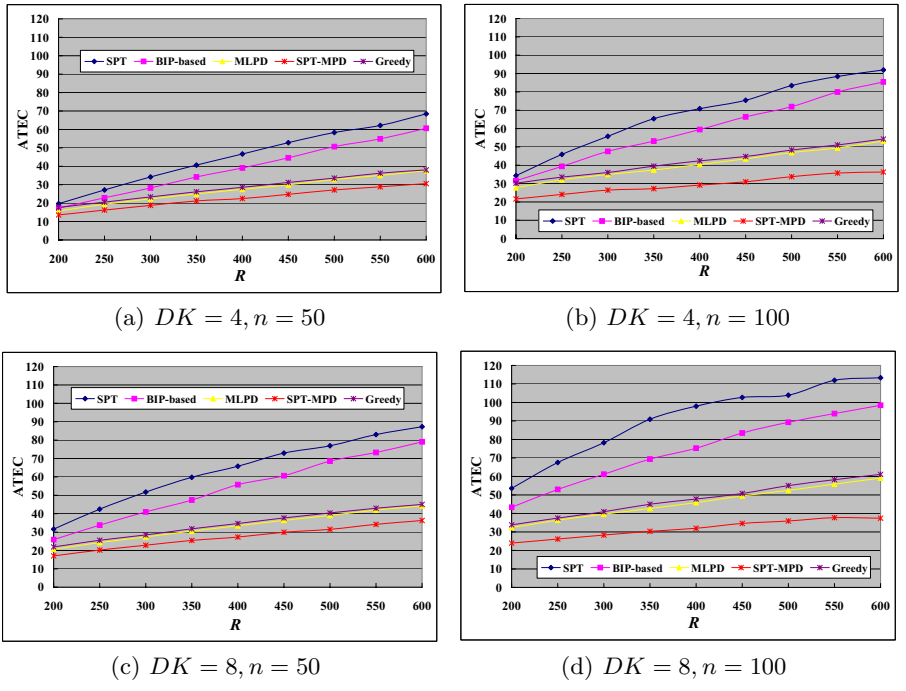
Fig. 3. Average total energy cost vs. the number of nodes in network



(a) SPT-MPD (b) MLPD

Fig. 4. Average total energy cost vs. the number of nodes in network

In Fig.5, we fixed DK and n while vary R. We make the following observations from the simulation results. The SPT-MPD algorithm performs the best at all different cases. The MLPD and the Greedy algorithm also have similar performance. However, the Greedy algorithm is the best heuristic in [12], we can know our SPT-MPD and MLPD algorithms outperform the other heuristics in [12]. The average total energy cost of broadcast routing increases as the growth of R in all subfigures. With the growth of the average length of transmission radii of



(a) $DK = 4, n = 50$ (b) $DK = 4, n = 100$

(c) $DK = 8, n = 50$ (d) $DK = 8, n = 100$

Fig. 5. Average total energy cost vs. the average length of transmission radius(R)

all nodes' *directions* (R), there are more arcs in auxiliary graph but the weights of these arcs also become heavier. Therefore, the networks, in which the nodes have relatively longer transmission radii, not always have good performance of energy efficiency.

6 Conclusion

We have studied the minimum energy cost broadcast routing problem with directional antennas in ad hoc wireless networks. We proposed a $|V|$-approximation algorithm SPT-MPD and one heuristic MLPD for this problem. Extensive simulations have been conducted to compare our SPT-MPD and MLPD algorithms with the Greedy algorithm [12], BIP-based algorithms [10] and the approximation algorithm in [2]. And simulation results have shown that our SPT-MPD algorithm outperform these three algorithms dramatically.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China under Grant No. 10671208 and 863 High-Tech Project under grant 2008AA01Z120.

References

1. Cartigny, J., Simplot-Ryl, D., Stojmenovic, I.: Localized Energy Efficient Broadcast for Wireless Networks with Directional Antennas. In: IFIPMED-HOC-NET Workshop (2002)
2. Charikar, M., Chekuri, C., Cheung, T.-Y., Dai, Z., Goel, A., Guha, S., Li, M.: Approximation Algorithms for Directed Steiner Problems. Journal of Algorithms 33(1), 73–91 (1999)
3. Chlamtac, I., Conti, M., Liu, J.N.: Mobile ad hoc networking: imperatives and challenges. Ad Hoc Networks 1, 13–64 (2003)
4. Dai, F., Wu, J.: Efficient Broadcasting in Ad Hoc Wireless Networks Using Directional Antennas. IEEE Transactions on Parallel and Distributed Systems 17(4), 1–13 (2006)
5. Guo, S., Yang, O.: Minimum-energy Multicast in Wireless Ad Hoc Networks with Adaptive Antennas: MILP Formulations and Heuristic Algorithms. IEEE Transactions on Mobile Computing 4(5), 333–346 (2006)
6. Li, D., Jia, X., Liu, H.: Energy Efficient Broadcast Routing in Static Ad Hoc Wireless Networks. IEEE Transactions on Mobile Computing 3(2), 144–151 (2004)
7. Li, D., Li, Z., Liu, L.: Energy Efficient Broadcast Routing in Ad Hoc Sensor Networks with Directional Antennas. In: Li, Y., Huynh, D.T., Das, S.K., Du, D.-Z. (eds.) WASA 2008. LNCS, vol. 5258. Springer, Heidelberg (2008)
8. Kang, I., Poovendran, R.: S-GPBE: A Power-Efficient Broadcast Routing Algorithm Using Sectored Antenna. In: Proc. of the Wireless and Optical Communications (2003)
9. Ramanathan, R.: On the Performance of Ad Hoc Networks with Beamforming Antennas. In: Proc. of the ACM MobiHoc, pp. 95–105 (2001)
10. Roy, S., Hu, Y.C., Peroulis, D., Li, X.-Y.: Minimum-Energy Broadcast Using Practical Directional Antennas in All-Wireless Network. In: Proc. of the IEEE INFOCOM, pp. 1–12 (2006)
11. Sarkar, T.K., Wicks, M.C., Salazar-Palma, M.: Smart Antennas. John Wiley and Sons, Hoboken (1996)
12. Tang, J., Xue, G., Zhang, W.: Power Efficient Broadcasting and Multicasting in Wireless Networks with Directional Antennas. In: Proc. of the IEEE ICC (2005)
13. Wieselthier, J.E., Nguyen, G.D.: On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks. In: Proc. of the IEEE INFOCOM, pp. 585–594 (2000)
14. Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: Energy-Limited Wireless Networking with Directional Antennas: the Case of Session-Based Multicasting. In: Proc. of the IEEE INFOCOM, vol. 1, pp. 190–199 (2002)

Approximating the Multicast Traffic Grooming Problem in Unidirectional SONET/WDM Rings*

Jiguo Yu[1,**], Suxia Cui[1], and Guanghui Wang[2,3]

[1] School of Computer Science, Qufu Normal University
Ri-zhao, Shandong 276826, China
jiguoyu@sina.com
[2] School of Mathematics and System Science, Shandong University
Jinan Shandong 250100, China
[3] Laboratoire de Mathématiques Appliquées aux Systèmes, Ecole Centrale Paris
Grande voie des vignes, 92295 Châtenay-Malabry Cedex, France

Abstract. As an important technique, traffic grooming aggregates the traffic of multiple connections into a single wavelength channel, and significantly increases the utilization of bandwidth of wavelength channels. In this paper, for the problem of grooming multicast traffics in unidirectional SONET/WDM rings, an efficient approximation algorithms with approximation factor of $2lng + lnM + o(ln(M \cdot g))$ for any given number of multicast requests M and grooming factor g is proposed.

1 Introduction

Wavelength-division multiplexing (WDM) is emerging as a dominant technology used in backbone networks. WDM technology can significantly increase the capacity of a fiber by allowing simultaneous transmission of multiple wavelengths (channels). A WDM network consists of switching nodes interconnected by optical fiber links. In the WDM network architecture, a WDM wavelength channel can either carry a high-speed traffic stream (e.g. OC-48), or a number of low-speed traffic streams (e.g. OC-3). In a communication system, most of the connections have a small bandwidth requirement, such as voice data or text data. Without grooming, each connection would occupy a dedicated wavelength. Since wavelength is a limited resource in a WDM network, there are usually not enough wavelengths to support each connection with a dedicated wavelength. Therefore, it is necessary to allow multiple low-bandwidth connections to share the same wavelength. Traffic grooming is a technique that has one function, which aggregates the traffic of multiple connections into a single wavelength channel. Traffic

* The work is supported by NNSF (60373012, 10871119) of China, the French-Chinese foundation for sciences and their applications and the China scholarship Council, Promotional Foundation (2005BS01016) for Middle-aged or Young Scientists of Shandong Province, SRI of SPED(J07WH05), DRF and UF(XJ0609)of QFNU.

** The corresponding author.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 519–529, 2009.

grooming can significantly increase the utilization of bandwidth of wavelength channels.

The problem of effectively packing lower rate traffic streams onto the available wavelengths in order to achieve some desired goal is called traffic grooming. If the traffic demands are known in advance, then the problem is called static, otherwise the problem is called dynamic. In static traffic grooming, usually the aim is to minimize the overall network cost. Here the network cost includes the cost of electronics (this is the dominant cost) as well as the cost of optics (wavelengths per fiber). In dynamic traffic grooming, the aim is to groom the incoming traffic demands such that the blocking probability is minimum. Grooming multicast traffic is different from the case of unicast traffic, the former has efficient grooming in multicast traffic(one source and several destinations nodes) while the latter has efficient grooming in unicast traffic(one source and one destination). Grooming multicast traffic is an area of active research and although a lot of literature is available, not many results are known.

In WDM networks, an important issue is that the burden on electronic switching equipments become enormous, and their cost is the dominant cost in the networks. Fortunately, it is not necessary to electronically process all the incoming traffic at each node since most of the incoming traffic is neither sourced at that node nor destined to it. So to reduce the cost of electronic components at each node, we can selectively drop the wavelengths carrying traffic that requires electronic processing at that node and allow the remaining wavelengths optically bypass the node. So if a node (say n) does not act as a source or a destination for any traffic on some wavelength (say λ), then there is no need for an ADM corresponding to wavelength λ at node n. Since the cost of the ADMs (electronics) form the bulk of the network cost [1], we can see that intelligent grooming of low-rate traffic onto wavelengths can result in ADM savings, which results in a lower network cost.

When the various parameters comprising the switching mechanism in these networks became clearer, the focus of studies shifted, and today a large portion of research concentrates with the total hardware cost. This is modeled by considering the basic switching unit of Add-Drop-Multiplexer (ADM), and focusing on the total number of these ADMs. The key point here is that each lightpath uses two ADMs, one at each endpoint. If two adjacent lightpaths are assigned the same wavelength, then they can use the same ADM. An ADM may be shared by at most two lightpaths. Moreover, in studying the hardware cost, the issue of grooming became central. This problem stems from the fact that the network usually supports traffic that is at rates which are lower than the full wavelength capacity, and therefore the network operator has to be able to put together (groom) low-capacity demands into the high capacity fibers. In graph-theoretic terms, this can viewed as assigning colors to the lightpaths so that at most g lightpaths can share one edge. In terms of ADMs, each lightpath uses two ADMs, one at each endpoint, and in case g lightpaths of the same wavelength enter through the same edge to one node, they can all use the same ADM (thus saving $g - 1$ ADMs). Moreover, all the same colored paths ending at the

node through two given incident edges can share the same ADM. The goal is to minimize the total number of ADMs. Note that the above coloring problem is simply the case of $g = 1$.

2 Related Work

Grooming static unicast subwavelength traffic to minimize either the number of ADMs or the number of wavelengths required per fiber in WDM ring networks is a well studied problem [5,21]. Different traffic scenarios such as uniform all-to-all traffic [22], distance dependent traffic [5] and non-uniform traffic [21,1] have been studied. Work has also been done on other cost functions such as the overall network cost [15], which includes the cost of transceivers, wavelengths and the number of required hops. Recently there has been a lot of work on grooming both static [23] as well as dynamic [10] traffic in mesh networks. Although multicast traffic grooming in mesh WDM networks is a general case of the same problem in WDM rings, the ideas that are applied for mesh networks in [6,16,7] are not very attractive for unidirectional rings, since for unidirectional rings the routing is already fixed and the only way to effectively groom traffic is by using intelligent wavelength assignment. There has been some work in the case of WDM rings also. More specifically, in [18] the authors look at the problem of grooming given multicast traffic demands in a bidirectional WDM ring. They present a heuristic algorithm inspired by the algorithm to groom unicast traffic demands on WDM rings given in [21]. In [19], Rawat presented a node architecture with tap-and-continue capability. They also showed a lower bound and an upper bound for an algorithm, which has an approximation ratio of $\frac{3(N+z_{\min})}{2z_{\min}}$, where z_{min} is the minimum possible size of multicast sessions.

The problem of minimizing the number of ADMs for the case $g = 1$ was introduced in [14] for ring networks. For such a topology it was shown to be NP-complete in [8] and an approximation algorithm with approximation ratio$\frac{3}{2}$ was presented in [13] and improved in [20,9] to $\frac{10}{7} + \varepsilon$ and $\frac{10}{7}$ respectively. For general topologies [13] described an algorithm with approximation ratio $\frac{8}{5}$ and an improved $\frac{3}{2} + \varepsilon$-approximation one was presented in [2]. Finally, an algorithm with approximation ratio of $2lng + o(lng)$ for any fixed g in a ring topology has been given in [12]. In a different scenario, the problem of minimizing hardware components in optical networks using grooming in order to exploit large bandwidth has been studied in [3] for ring networks and in [4] for stars networks. An approximating algorithm with approximation ratio of $2lng + ln\Delta + o(ln(\Delta \cdot g))$ in tree networks is given in [11], where Δ is the maximum degree of nodes in tree networks.

In this paper we extend the results to multicast requests in unidirectional ring and the general topology networks. Namely, we provide a polynomial time algorithm with approximation factor of $2lng + lnM + o(ln(M \cdot g))$ for any given number of multicast sessions M and grooming factor g.

The rest of the paper is organized as follows: In the next section, we give our contribution, firstly we give some problem definitions and assumptions in

subsection 3.1, subsequently we give an approximated solutions for multicast traffic grooming in unidirectional ring in subsection 3.2, and in subsection 3.3 we also get an approximated solutions for the traffic grooming problem in general topology networks. Finally, we make a conclusion in section 4.

3 Our Contribution

3.1 Definition and Assumption

An instance of the traffic grooming problem is a triple (G, P, g) where $G = (V, E)$ is a graph, P is a set of simple paths in G and g is a positive integer, namely the grooming factor.

Definintion 3.1.1. *Given a subset $Q \subseteq P$ and an edge $e \in$ E, Q_e is the set of paths from Q using edge e. $l_Q(e)$ is the number of these paths, or in network terminology, the load induced on the edge e by the paths in Q. L_Q is the maximum load induced by the paths in Q on any edge of G.*

Definintion 3.1.2. *A coloring (or wavelength assignment) of (G, P) is a function $P \longmapsto \mathbb{N}^+ = \{1, 2, ...\}$. We extend the definition of ω on any subset Q of P as $\omega(Q) = \cup_{p \in Q} \omega(p)$. For a coloring ω, a color λ and any $Q \subseteq P$, Q^ω_λ is the subset of paths from Q colored λ by ω.*

Definintion 3.1.3. *A proper coloring (or wavelength assignment) ω of (G, P, g) is a coloring of P in which for any edge e at most g paths using e are colored with the same color. Formally, $\forall \lambda \in \mathbb{N}^+, L_{P^\omega_\lambda} \leq g$.*

Definintion 3.1.4. *A coloring ω is a 1-coloring of $Q \subseteq P$, if it colors the paths of Q using one color. A set Q is 1-colorable if there exists a proper 1-coloring for it. Note that a set $Q \subseteq P$ is 1-colorable iff $L_Q \leq g$.*

Definintion 3.1.5. *For a coloring ω of P and a node $v \in V$, Q_v is the subset of paths from Q having an endpoints in v, v may be a source node or destination node. The degree of one node $v (v \in Q_v)$ is the number of paths which v as the source node or destination node, formally, $d(v) = |Q_v|$. The degree of Q is the summation of the degree of all nodes in Q. ADM^ω_λ is the minimum number of ADMs operating at wavelength λ in all the network and ADM^ω is the minimum total number of ADMs in the network, that is for all the used wavelengths. Formally,*

$$d(Q) = \sum\nolimits_v d(v) = \sum\nolimits_v |Q_v|, ADM^\omega_\lambda(Q) = d(Q^\omega_\lambda),$$

and

$$ADM^\omega = \sum_\lambda ADM^\omega_\lambda(P) = \sum_\lambda d(P^\omega_\lambda).$$

The traffic grooming problem is the optimization problem of finding a proper coloring ω of (G, P, g) minimizing ADM^ω. Now we have a review of some assumption of multicast traffic grooming problem in unidirectional SONET/WDM rings.

Assumption 1(Physical network) [19]. *The physical network is assumed to be a clockwise unidirectional SONET/WDM ring with N nodes numbered $0, 1, ..., N-1$ distributed on the ring in the clockwise direction as shown in Figure 1. We assume that there is a single fiber between adjacent nodes, which can support W wavelengths given by $\lambda_0, \lambda_1, ..., \lambda_{W-1}$ and the capacity of each wavelength is assumed to be C units.*

Assumption 2 (Traffic [19]). *We assume that there are M given multicast traffic requests denoted by $R_0, R_1, ..., R_{M-1}$. Every multicast request specifies a source node and a set of destination nodes. We assume that each multicast request is for r units of traffic. Also, the wavelength capacity C is assumed to be an integral multiple of the required traffic rate r, i.e. $C = g \times r$. We refer to g, the number of subwavelength multicast demands that can be groomed on a single wavelength channel, as the grooming ratio.*

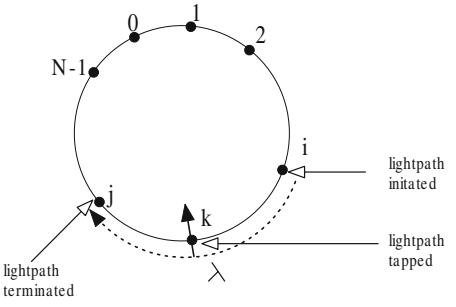


Fig. 1. Node model in Unidirectional SONET ring

As already remarked in the introduction, in order to reduce the cost of electronic components at each node, we can selectively drop the wavelengths carrying the traffic that requires electronic processing at that node and allow the remaining wavelengths to optically bypass the node. Figure 2 shows all the multicast traffic requests that pass through network node i. The solid arcs correspond to the requests that contain node i as source or one of the destinations and the dotted arcs represent the traffic requests that pass through node i but do not contain it as source or one of the destinations.

3.2 Approximated Solutions for Multicast Traffic Grooming in Unidirectional SONET/WDM Rings

In this section we present and analyze a modified approximation algorithm for unidirectional rings, GROOMBYSC(k) where has a parameter k, depending only on g, that will be properly determined in the analysis.

Algorithm GROOMBYSC(k). Note that a multicast session can be seen as a combination several unicast requests, or several sets of paths.

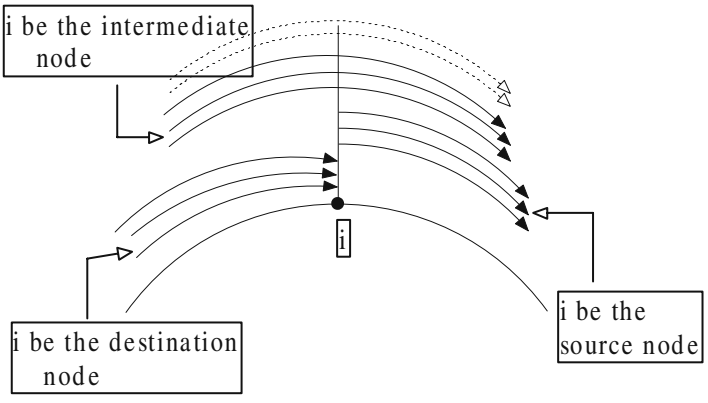


Fig. 2. Multicast traffic requests that pass node i

Let z represent the maximum possible size of multicast sessions, here by the size of a session, we mean the total number of source and destination nodes in that session. In the following discussion, provided there are M multicast sessions.

Generally, the algorithm GROOMBYSC(k) [12] described below has three phases. During Phase 1 it computes 1-colorable sets and their corresponding weights. It considers subsets of the paths P of size at most $k \cdot g$. Whenever a 1-colorable set Q is found, it is added to the list of relevant sets, together with a corresponding weight, that is the minimum number of ADMs requied by Q (when its path receive the same color). In Phase 2 it finds a set cover of P using subsets calculated in Phase 1. It uses the GREEDYSC approximation algorithm for the minimum weight set cover problem presented in [17]. In Phase 3 it transforms the set cover into a partition by eliminating intersections and then colors the paths belonging to each set with the same color, different from the one of the other sets of the partition.

Now we give a small modification in the algorithm GROOMBYSC(k) presented in [12]. we add a outer loop to apply for grooming several multicast sessions in phase 1, and we also simplified the weight as the degree of one path.

---

a) Phase 1- Prepare the input for GREEDY:

$S \longleftarrow \emptyset$

For each $R_i \subseteq R = \{R_1, R_2, ..., R_M\}$,
such that $|R_i| \leq z$ {
For each $U \subseteq V$, such that $|U| \leq k$ {
For each $Q \subseteq P_U$, such that $|Q| \leq k \cdot g$ {
If Q is 1-colorable then {
$S \longleftarrow S \cup \{Q\}$
weight[Q]=d(Q)
}
}
}
}

b) Phase 2- Run GREEDYSC:
// Assume w.l.o.g $SC = \{S_1, S_2, ..., S_W\}$
$SC \longleftarrow GREEDYSC(S, weight)$.
c) Phase 3- Transform the Set Cover SC into a Partition PART:
$PART \longleftarrow \emptyset$
For $i = 1$ to W $\{PART_i \longleftarrow S_i\}$
As long as there are two intersecting sets $PART_i$, $PART_j$ {
$PART_i \longleftarrow PART_i \setminus PART_j$
}
For $\lambda = 1$ to W $(\omega(Q) = W)$ {
$PART \longleftarrow PART \cup \{PART_\lambda\}$
For each $p \in PART_\lambda$ $\{\omega(p) = \lambda\}$
}

Correctness and Running Time. The correctness and time complexity of the algorithm follow the arguments of [12]. In particular, it can be shown that its running time is polynomial in $n = |P|$ and $m = |E|$, for any fixed k and g and for all instances (G, P, g), by taking into account the given M multicast sessions.

Lemma 1. *Given an integer z, there exists a solution $\overline{SC}$ for the instance of the set cover problem determined on Phase 2 of GROOMBYSC(k), with $k = M \cdot z$, such that weight$(\overline{SC}) \leq ADM^*(1 + 2g/z)$.*

Proof. Let $\omega^*(\mathrm{P}) = \{1, 2, ..., \mathrm{W}^*\}$ and $1 \leq \lambda \leq \mathrm{W}^*$. Consider the set V^*_λ of node v having at least one ADM operating at wavelength λ at node v, we divide V^*_λ into sets of at least z by starting from an arbitrary node(see Fig. 3) , going clockwise along the unidirectional ring, and at most k nodes in the following way.

Let $V_{\lambda,j} (j = 1, ..., P_\lambda)$ be the subsets of nodes obtained in this way, and q_λ be the number of them having at least z nodes in V^*_λ. Let $r_\lambda \leq z$ be the number of nodes in the residual set, if it exists. Formally, let $\mathrm{ADM}^*_\lambda = |\mathrm{V}^*_\lambda| = zq_\lambda + \mathrm{r}_\lambda$, where $\mathrm{r}_\lambda = |\mathrm{V}^*_\lambda| \bmod z$, and $0 \leq \mathrm{r}_\lambda \leq z$.

Notice that if no set with less than z nodes is in the partition, $p_\lambda = q_\lambda$, and $\mathrm{r}_\lambda = 0$, otherwise, $p_\lambda = q_\lambda + 1$.

Clearly, $\forall 1 \leq \mathrm{j} \leq q_\lambda$, $|V_{\lambda,j}| = z$, and in case $r_\lambda > 0$, we have $|\mathrm{V}_{\lambda,q_\lambda+1}| < z$. In both cases $|V_{\lambda,j}| \leq z$. Therefore, each $V_{\lambda,j}$ is considered in the outer loop of phase 1 of the algorithm, and hence, is added to S.

For $V_{\lambda,j}$, we define $S_{\lambda,j}$ to be the set of paths colored by ω^* with both source and destination clockwise endpoints in $V_{\lambda,j}$, or only one of the two endpoints in $V_{\lambda,j}$. As $V_{\lambda,j}$ has at most z nodes, and every node may be the clockwise endpoint of at most g paths from a 1-colorable set, so by the feasibility of the coloring at most g paths are charged to each node in $V_{\lambda,j}$, we have $|\mathrm{S}_{\lambda,\mathrm{j}}| \leq g \cdot k$.

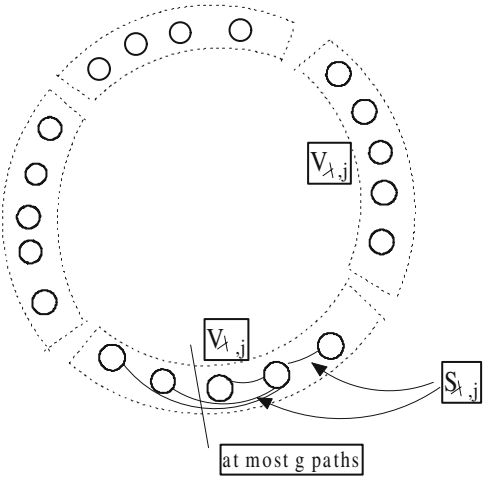


Fig. 3. The sets of $V_{\lambda,j}$

Therefore, $\mathrm{S}_{\lambda,\mathrm{j}}$ is considered by the algorithm in the inner loop of phase 1, thus $\mathrm{S}_{\lambda,\mathrm{j}} \in \mathrm{S}$.

Notice that every p with $\omega^*(p) = \lambda$ is contained in exactly one set $\mathrm{S}_{\lambda,\mathrm{j}}$, therefore $\overline{\mathrm{SC}_\lambda} = \cup_\mathrm{j}\{\mathrm{S}_{\lambda,\mathrm{j}}\}$ is a cover of all the paths colored λ by ω^*. Considering all colors $1 \leq \lambda \leq \mathrm{W}^*$, we conclude that $\overline{\mathrm{SC}} = \cup_{\lambda=1}^{\mathrm{W}^*}\overline{\mathrm{SC}_\lambda}$ is a cover of P with sets from S. It remains to show that its weight has the claimed property.

First observe that $\mathrm{ADM}^*_\lambda = zq_\lambda + \mathrm{r}_\lambda$, summing up over all possible values of λ, we obtain: $\mathrm{ADM}^* = z\sum_\lambda q_\lambda + \sum_\lambda \mathrm{r}_\lambda$, which implies

$$\sum\nolimits_\lambda q_\lambda \leq \frac{\mathrm{ADM}^*}{z} \tag{1}$$

We consider two cases as follows:

Case 1: We claim that $\forall \mathrm{j} \leq q_\lambda$, weight$(S_{\lambda,j})$=$d(S_{\lambda,j}) \leq z + g$. This is because:

-The endpoints of the paths with both source and destination node in $S_{\lambda,j}$ are in $V_{\lambda,j}$ and $|V_{\lambda,j}| = z$.

-The number of paths having only one of source or destination endpoints in $V_{\lambda,j}$ is at most g.

Case 2: For the set $\mathrm{j} = q_\lambda + 1$, we have weight$(\mathrm{S}_{\lambda,\mathrm{j}}) \leq g \cdot q_\lambda + \mathrm{r}_\lambda$[19]. The reasons are:

-The same discussion as above, the number of paths having only one endpoint in $\mathrm{V}_{\lambda,q_\lambda+1}$ are at most g.

-The endpoints of the paths with both source and destination nodes in $\mathrm{S}_{\lambda,q_\lambda+1}$ are in $\mathrm{V}_{\lambda,q_\lambda+1}$, and $|\mathrm{V}_{\lambda,q_\lambda+1}| = r_\lambda$.

Then, summing up for all $1 \leq \mathrm{j} \leq q_\lambda + 1$, we have

$$\mathrm{weight}(\overline{\mathrm{SC}}_\lambda) \leq \sum_{\mathrm{j}=1}^{\mathrm{q}_\lambda} (z+g) + g \cdot \mathrm{q}_\lambda + \mathrm{r}_\lambda.$$

As $\mathrm{ADM}^* = z\sum_\lambda q_\lambda + \sum_\lambda \mathrm{r}_\lambda$, and $\sum_\lambda \mathrm{q}_\lambda \leq \frac{\mathrm{ADM}^*}{z}$, This is clear that

$$\mathrm{weight}(\overline{\mathrm{SC}}) = \sum\nolimits_\lambda \mathrm{weight}(\overline{\mathrm{SC}}_\lambda) \leq \mathrm{ADM}^* + 2g\sum\nolimits_\lambda q_\lambda$$

$$\leq \mathrm{ADM}^* + 2g \cdot \frac{\mathrm{ADM}^*}{z} = \mathrm{ADM}^*\Big(1 + \frac{2g}{z}\Big) \tag{2}$$

The proof is then finished. □

Theorem 1. *There is a $2lng + lnM + o(ln(M \cdot g))$-approximation algorithm for the multicast traffic grooming problem in unidirectional SONET/WDM rings.*

Proof. The greedy algorithm for the Minimum Weight Set Cover Problem is a H_f -approximation algorithm, where f is the maximum cardinality of the sets in the input and $H_f = 1 + \frac{1}{2} + ... + \frac{1}{f}$ be the f-th harmonic number.

As $\mathrm{ADM}^\omega = H_f \cdot \mathrm{weight}(\overline{\mathrm{SC}})$, which obtained in [19], we have $\mathrm{ADM}^\omega = H_f \cdot \mathrm{ADM}^* \cdot (1 + \frac{2g}{z})$, which equivalent to

$$\rho \leq H_f \cdot \Big(1 + \frac{2g}{z}\Big) \tag{3}$$

Let $z = glng$ and $f = M \cdot g \cdot z$. Then,

$$\rho \leq H_{M \cdot g \cdot z} \cdot \Big(1 + \frac{2g}{z}\Big) \leq \big[1 + ln(M \cdot g^2 lng)\big]\Big(1 + \frac{2}{lng}\Big)$$

$$= (1 + 2lng + lnM + lnlng)\Big(1 + \frac{2}{lng}\Big)$$

$$= 2lng + lnM + o\big(ln(M \cdot g)\big). \tag{4}$$

The proof is then finished. □

From the above, we know that the approximation ratio of the algorithm depends on the number of multicast requests in unidirectional SONET/WDM rings.

4 Conclusion

In this paper, we addressed the traffic grooming problem with respect to the minimization of ADMs. An approximation algorithm with approximation ratios logarithmic in g is studied. we modified the algorithm to solve the multicast traffic grooming problem in unidirectional SONET/WDM rings, and gave the approximation ratio of $2lng + lnM + o(ln(M \cdot g))$.

References

1. Billah, A.R.B., Wang, B., Awwal, A.A.S.: Effective Traffic Grooming in WDM Rings. In: Proceedings of IEEE Globecom, vol. 3, pp. 2726–2730 (2002)
2. Călinescu, G., Frieder, O., Wan, P.: Minimizing Electronic Line Terminals for Automatic Ring Protection in General WDM Optical Networks. IEEE Journal of Selected Area on Communications 20(1), 183–189 (2002)
3. Chen, B., Rouskas, G.N., Dutta, R.: Traffic Grooming in WDM Ring Networks with the Min-Max Objective. In: Mitrou, N.M., Kontovasilis, K., Rouskas, G.N., Iliadis, I., Merakos, L. (eds.) NETWORKING 2004. LNCS, vol. 3042, pp. 174–185. Springer, Heidelberg (2004)
4. Chen, B., Rouskas, G.N., Dutta, R.: Traffic Grooming in Star Networks. In: Broadnets 2004: Workshop on Traffic Grooming in WDM Networks (2004)
5. Chiu, A.L., Modiano, E.H.: Traffic Grooming Algorithms for Reducing Electronic Multiplexing Costs in WDM Ring Networks. IEEE/OSA Journal of Lightwave Technology 18(1), 2–12 (2000)
6. Chowdhary, G.V., Murthy, C.S.R.: Grooming of Multicast Sessions in WDM Mesh Networks. In: The 1st Workshop on Traffic Grooming in WDM Networks, San Jose, USA (2004)
7. Chowdhary, G.V., Murthy, C.S.R.: Dynamic Multicast Traffic Engineering in WDM Groomed Mesh Networks. In: The 1st Workshop on Traffic Grooming in WDM Networks, San Jose, USA (2004)
8. Eilam, T., Moran, S., Zaks, S.: Lightpath Arrangement in Survivable Rings to Minimize the Switching Cost. IEEE Journal of Selected Area on Communications 20(1), 172–182 (2002)
9. Epstein, L., Levin, A.: Better Bounds for Minimizing SONET ADMs. In: The 2nd Workshop on Approximation and Online Algorithms, Bergen, Norway (2004)
10. Farahmandz, F., Huang, X., Jue, J.P.: Efficient Online Traffic Grooming Algorithms in WDM Mesh Networks with Drop-and-Continue Node Architecture. In: Proc. of IEEE Broadnets, pp. 180–189 (2004)
11. Flammini, M., Monaco, G., Moscardelli, L., Shalom, M., Zaks, S.: Approximating the Traffic Grooming Problem in Tree and Star Networks. Journal of Parallel and Distributed Computing 68(7), 939–948 (2008)
12. Flammini, M., Moscardelli, L., Shalom, M., Zaks, S.: Approximating the Traffic Grooming Problem. Journal of Discrete Algorithms 6(3), 472–479 (2008)
13. Gargano, L., Vaccaro, U.: Routing in All-Optical Networks: Algorithmic and GraphCTheoretic Problemsin. In: Cai, N., Dueck, G., Althöfer, I. (eds.) Numbers, Information and Complexity. Kluwer Academic Publishers, Dordrecht (2000)
14. Gerstel, O., Lin, P., Sasaki, G.: Wavelength Assignment in a WDM Ring to Minimize Cost of Embedded SONET Rings. In: Proc. of the 17th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1998), pp. 94–101 (1998)
15. Gerstel, O., Ramaswami, R.: Cost-Effective Traffic Grooming in WDM Rings. IEEE/ACM Transactions of Networking 8(5), 618–630 (2000)
16. Huang, X., Farahmandz, F., Jue, J.P.: Multicast Traffic Grooming in Wavelength-Routed WDM Mesh Networks using Dynamically Changing Light-Trees. IEEE/OSA Journal of Lightwave Technology 23(10), 3178–3187 (2005)
17. Johnson, D.S.: Approximation Algorithms for Combinatorial Problems. J. Comput. System Sci. 9, 256–278 (1974)

18. Madhyastha, H.V., Srinivas, N., Chowdhary, G.V., Murthy, C.S.R.: Grooming of Multicast Sessions in WDM Ring Networks. In: Torra, V., Narukawa, Y. (eds.) MDAI 2008. LNCS, vol. 5285, pp. 1–12. Springer, Heidelberg (2008)
19. Rawat, A., La, R., Marcus, S., Shayman, M.: Grooming Multicast Traffic in Unidirectional SONET/WDM Rings. IEEE Journal on Selected Areas in Communications - Optical Communications and Networking Series 25(6), 70–83 (2007)
20. Shalom, M., Zaks, S.: A $\frac{10}{7} + \varepsilon$ Approximation Scheme for Minimizing the Number of ADMs in Sonet Rings. In: The 1st Annual International Conference on Broadband Networks. San-Jos'e, California, USA (2004)
21. Zhang, X., Qiao, C.: An Effective and Comprehensive Approach for Traffic Grooming and Wavelength Assignment in SONET/WDM Rings. IEEE/ACM Transactions On Networking 8(5), 608–617 (2000)
22. Zhang, X., Qiao, C.: Scheduling in Unidirectional WDM Rings and its Extensions. In: Vicedo, J.L., Martínez-Barco, P., Muńoz, R., Saiz Noeda, M. (eds.) EsTAL 2004. LNCS, vol. 3230, pp. 208–219. Springer, Heidelberg (2004)
23. Zhu, K., Mukherjee, B.: Traffic Grooming in an Optical WDM Mesh Network. IEEE Journal on Selected Areas in Communications 20(1), 122–133 (2002)

An Algorithm with Better Approximation Ratio for Multicast Traffic in Unidirectional SONET/WDM Rings*

Jiguo Yu[1,**], Suxia Cui[1], and Guanghui Wang[2,3]

[1] School of Computer Science, Qufu Normal University at Ri-zhao
Shandong 276826, China
jiguoyu@sina.com
[2] School of Mathematics and System Science, Shandong University
Shandong 250100, China
[3] Laboratoire de Mathématiques Appliquées aux Systèmes, Ecole Centrale Paris
Grande voie des vignes, 92295 Châtenay-Malabry Cedex, France

Abstract. The goal for the problem of efficient grooming of given non-uniform multicast traffic demands on a unidirectional SONET/WDM ring is to try to minimize the network cost as given by (i) the number of wavelengths required per fiber and (ii) the number of electronic Add-Drop Multiplexers (ADMs) required in the ring. The problem with cost function (i) can be reduced to a corresponding traffic grooming problem for unicast traffic which can then be modeled as a standard circular-arc graph coloring problem.The function (ii) is a main research topic in recent studies.The problem is NP hard for both the cost functions. For the problem with fuction (ii), we present a algorithm with better approximation ratio, $\frac{3}{2}(\frac{N}{N-1}+1)$. Additionally, we give a lower bound and upper bound for the number of ADMs.

1 Introduction

Wavelength Division Multiplexing (WDM) significantly increases the available network bandwidth capacity by delivering data over multiple wavelengths (channels) simultaneously. With each channel operating at a high rate and multiple channels deployed per fiber, very high transmission capacity can be achieved. An important issue with such a high capacity network is that it places enormous burden on the electronic switches. Hence, it is hardly surprising that the dominant cost in WDM networks is the cost of the electronic switching equipment required. Fortunately, it is not necessary to electronically process all the incoming traffic at each node since most of the incoming traffic is neither sourced at

* The work is partially supported by NNSF (60373012, 10871119) of China, the French-Chinese foundation for sciences and their applications and the China scholarship Council, Promotional Foundation (2005BS01016) for Middle-aged or Young Scientists of Shandong Province, SRI of SPED (J07WH05), DRF and UF (XJ0609) of QFNU.

** The corresponding author.

D.-Z. Du, X. Hu, and P.M. Pardalos (Eds.): COCOA 2009, LNCS 5573, pp. 530–540, 2009.

that node nor destined to it. So to reduce the cost of electronic components at each node, we can selectively drop the wavelengths carrying traffic that requires electronic processing at that node and allow the remaining wavelengths optically bypass the node.

In a SONET/WDM ring each wavelength operates at a line rate of OC-N and can carry several low rate OC-M ($M \leq N$) traffic channels using Time Division Multiplexing (TDM). The timeslots on a wavelength are referred to as the subwavelength channels. Electronic Add-Drop Multiplexers (ADMs) are required to add (drop) the subwavelength traffic at the source (destination) node. On receiving a wavelength channel, the ADM, corresponding to that particular wavelength, can add/drop timeslots on the wavelength channel without disrupting the onward transmission of other timeslots on the wavelength. So if a node (say n) does not act as a source or a destination for any traffic on some wavelength (say λ), i.e., if no add/drop of any timeslot on λ is required at n, then there is no need for an ADM corresponding to wavelength λ at node n. Since the cost of the ADMs (electronics) form the bulk of the network cost [1], we can see that intelligent grooming of low-rate traffic onto wavelengths can result in ADM savings, which results in a lower network cost.

Typically in WDM based optical networks, the bandwidth available per wavelength is much larger than the bandwidth required per session, and with the advancement of optical technology, it seems likely that this mismatch will continue to grow in the near future. Hence for efficient bandwidth usage, it is prudent to combine several low rate traffic sessions onto a single wavelength. The problem of effectively packing lower rate traffic streams onto the available wavelengths in order to achieve some desired goal is called traffic grooming. If the traffic demands are known in advance, then the problem is called static, otherwise the problem is called dynamic. In static traffic grooming, usually the aim is to minimize the overall network cost. Here the network cost includes the cost of electronics (this is the dominant cost) as well as the cost of optics (wavelengths per fiber). In dynamic traffic grooming, the aim is to groom the incoming traffic demands such that the blocking probability is minimum.

Grooming multicast traffic is different from unicast traffic, the former has efficient grooming in multicast traffic(one source and several destinations nodes) while the latter has efficient grooming in unicast traffic(one source and one destination node). While most of the earlier studies of the traffic grooming problem have dealt exclusively with the unicast traffic, it is expected that in the future a sizable portion of the traffic will be multicast in nature. This is mainly because of the increasing demands of multicast services such as multimedia conferencing, video distribution, collaborative processing, etc.. Grooming multicast traffic is still an area of active research and although a lot of literature is available, not many results are known.

In this paper we concentrate on three points as follows, (i) the static traffic grooming; (ii) the multicast traffic grooming and (iii) the minimum number of ADMs required in networks.

2 Related Work

Grooming static unicast subwavelength traffic to minimize either the number of ADMs or the number of wavelengths required per fiber in WDM ring networks is a well studied problem [2]. Different traffic scenarios such as uniform all-to-all traffic [12], distance dependent traffic [2] and non-uniform traffic [13,1] have been studied. Work has also been done on other cost functions such as the overall network cost [6], which includes the cost of transceivers, wavelengths and the number of required hops. Recently there has been a lot of work on grooming both static [14] as well as dynamic [4] traffic in mesh networks. Although multicast traffic grooming in mesh WDM networks is a general case of the same problem in WDM rings, the ideas that are applied for mesh networks in [1,3,7] are not very attractive for unidirectional rings, since for unidirectional rings the routing is already fixed and the only way to effectively groom traffic is by using intelligent wavelength assignment. More specifically, in [8] the authors look at the problem of grooming given multicast traffic demands in a bidirectional WDM ring. They present a heuristic algorithm inspired by the algorithm to groom unicast traffic demands on WDM rings given in [13]. In [9], Anuj Rawat presented a node architecture with tap-and-continue capability. They also showed a lower bound and a upper bound for an algorithm, which has an approximation ratio of $\frac{3(N+z_{\min})}{2z_{\min}}$, where z_{min} is the minimum size of all the multicast sessions, and N is the number of nodes in the ring.

In this paper we consider the problem of static grooming of non-uniform multicast traffic on a unidirectional SONET/WDM ring. We previously consider the problem of the number of wavelengths required per fiber can be modeled as a circular-arc graph coloring problem. Thus, we can apply the standard coloring techniques to the problem. We then suggest a graph based heuristic for cost function (ii). We extend the traffic grooming heuristic for non-uniform unicast traffic given in [6] to the multicast case. We show a lower bound on the number of ADMs which is presented in [9] depends on the number of nodes in the networks. Additionally, we present an algorithm with a better approximation ratio and give a better upper bound on the number of ADMs.

The rest of the paper is organized as follows: some problem statements such as assumption, node architecture and modeling are presented in section 3. In section 4, we develop an algorithm and give the analysis. Conclusion is made in section 5.

3 Problem Statement

3.1 Assumption and Node Architecture

Assumption 1(Physical network) [9]. *The physical network is assumed to be a clockwise unidirectional SONET/WDM ring with N nodes numbered $0, 1, ..., N-1$ distributed on the ring in the clockwise direction as shown in Fig.1, which with a tap-and-continue nodes presented in [9]. We assume that there is*

a single fiber between adjacent nodes, which can support W wavelengths given by $\lambda_0, \lambda_1, ..., \lambda_{W-1}$ *and the capacity of each wavelength is assumed to be C units.*

Assumption 2 (Traffic) [9]. *We assume that there are M given multicast traffic requests denoted by* $R_0, R_1, ..., R_{M-1}$*. Every multicast request specifies a source node and a set of destination nodes. We assume that each multicast request is for r units of traffic. Also, the wavelength capacity C is assumed to be an integral multiple of the required traffic rate r, i.e.* $C = g \times r$*. We refer to g, the number of subwavelength multicast demands that can be groomed on a single wavelength channel, as the grooming ratio.*

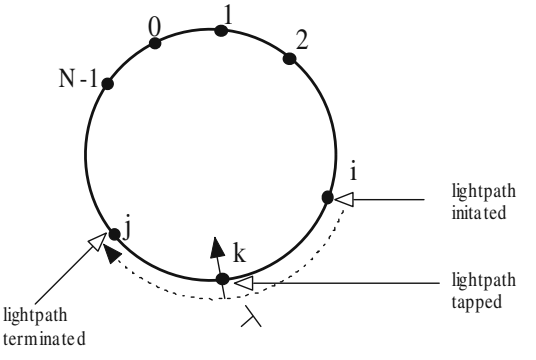


Fig. 1. Node model in Unidirectional SONET ring

As shown in Fig.1 [9], if a lightpath is set-up between nodes i and j on wavelength λ, and traffic from i to an intermediate node k is also groomed on λ or i has to send the same traffic to k (this is the case that i is the source and j and k are the destinations of a multicast traffic request), then instead of terminating the lightpath at k. We can drop a small amount of light of wavelength λ at k to extract the required data packets and let the rest of the light flow through, i.e. we can tap the lightpath at any intermediate node. It should be clear that if we want to add (groom) some subwavelength traffic on wavelength λ at node k, then we have to tear down the lightpath at k, carry out the grooming and then set-up a new lightpath on wavelength λ at node k. Note that in case we are sending different traffic to nodes j and k from node i on a single lightpath (by tapping the lightpath at intermediate node k), then using the above scheme, after passing node k there would be some unnecessary traffic on the lightpath (traffic sent from i to k). Clearly there is no such bandwidth wastage when we are sending the same traffic to both the nodes j and k.

Note that for SONET rings, without loss of generality, we can assume that r is equal to the capacity of individual subwavelength channel (timeslot). Hence, the grooming ratio g is equal to the number of subwavelength channels available on each wavelength.

With the above traffic model we can consider multicast requests of different bandwidth requirements also. The important requirement is that each request should be splittable into several individual multicast requests of granularity r.

3.2 Modeling

Since we assume the network to be a clockwise unidirectinal ring, each traffic request can be treated as an arc on the ring starting from the source and going clockwise through the intermediate destinations up to the final destination. Each arc should be assigned one subwavelength channel. So if any two multicast requests share some fiber, i.e. the corresponding arcs overlap, then they cannot be groomed on the same subwavelength channel. Consider a graph $G = (V, E)$ where $V = \{v_0, v_1, ..., v_{M-1}\}$ is the set of vertices with each vertex v_i representing a multicast request R_i and there is an edge $v_iv_j \in E$ if and only if the multicast requests R_i and R_j share some fiber, i.e. the arcs corresponding to requests R_i and R_j overlap. We call the graph G Request Graph. Thus, we can model the problem of minimizing the number of wavelengths per fiber as a proper graph coloring problem, which is equivalent to coloring the vertices of the Request Graph with the minimum number of colors such that no two adjacent vertices have the same color.

We observe that the Request Graph G belongs to the family of circular arc graphs [11]. Since minimum coloring of circular arc graphs is NP complete [5] and any instance of minimum arc graph coloring can be reduced to the traffic grooming problem under study, grooming multicast (or unicast) traffic on a unidirectional ring to minimize the number of wavelengths required per fiber is NP complete.

For the problem of minimizing the number of ADMs required in the network, we also use the Request Graph. Consider the vertex set $C_i \subseteq V$ representing all the multicast traffic requests groomed on wavelength λ_i. Note that the request graph corresponding to the traffic requests represented by C_i is exactly equal to $G[C_i]$, the subgraph induced by vertex set C_i on the Request Graph G. Now the minimum number of subwavelength channels required to groom the traffic requests represented by C_i is given by $\chi(G[C_i])$, the chromatic number of the request graph corresponding to the particular set of traffic requests. So the traffic requests represented by C_i can be groomed on a single wavelength only if $\chi(G[C_i]) \leq g$, i.e., the subgraph $G[C_i]$ induced on the request graph G by the vertex set C_i is g-colorable. Hence, given a set of multicast traffic requests modeled by the Request Graph $G = (V, E)$, any valid traffic grooming can be modeled as a cover $\mathcal{C} = \{V_0, V_1, ...\}$ of the vertex set V into non-overlapping clusters $V_0, V_1, ...$ such that $\cup_i V_i = V$, $V_i \cap V_j = \emptyset$ for all $i \neq j$ and $G[V_i]$ is g-colorable for all i. Therefore, the number of ADMs required in the network is given by $\sum_i |\cup_{v \in V_i} S_v|$, where S_v denotes a set of source and the destination node for request R_i.

Note that the problem of grooming multicast traffic on unidirectional rings to minimize the number of ADMs is NP hard.

4 GreedyGroom(M) Algorithm and Analysis

4.1 GreedyGroom(M) Algorithm

We randomly group M traffic requests into clusters of g requests each. As each g requests can model as a circular-arc graph, we try to using the minimum number of colors to color the graph. Then grouping g colors(subwavelength channels) together to form one wavelength and to groom these requests greedily.

Algorithm GreedyGroom(M).
Minimizing the number of ADMs for grooming the given M multicast requests.

Phase-1:
Randomly group M traffic requests into clusters of g requests each.
Phase-2:
Do {
For a graph $G_i = (V, E)$, try to use the minimum number of colors to coloring the graph.
// G_i is a circular-arc graph, and it can use proper coloring.
} while ($i \leq \lceil M/g \rceil$)
Phase-3:
After coloring each graph G_i, group g colors (subwavelength channels) together to form one wavelength.

From the algorithm above, we observe that each g requests can be modeled as a circular-arc graph which is g-colorable. Let the chromatic number of graph G_i be χ_i, and it can be colored by Karapetian's algorithm [10] using at most $\lfloor 3\chi_i/2 \rfloor$ colors.

4.2 Performance Analysis

Lower Bound. When each g requests can be assigned the same wavelength, in other words, they can use the same number of colors to complete the proper coloring. Then we can use the minimum number of wavelengths, it is given by $\lceil \chi_i/g \rceil$. We know that if network node i does not act as source or destination node for any multicast request, then since no traffic is being added or dropped at i, there is no need to equip i with ADM on any wavelength. Hence the total number of ADMs required at each node of the network is at least as large as L given by

$$L = \sum_{i=0}^{N-1} \left\lceil \frac{\chi_i}{g} \right\rceil \tag{1}$$

where N is the number of nodes in the ring, and χ_i is the chromatic number of the graph G_i, g is the grooming ratio.

In [9] showed that the lower bound does not depend on the number of nodes in the ring by considering each node of the ring in isolation. Now we will prove that

it will depend on the number of nodes by looking at a pair of nodes. To explain this, we try to calculate the expected value of the lower bound for grooming M multicast traffic requests on a SONET ring having N nodes and grooming ratio g. The process is similar to [9].

Let z_i represent the size of the i-th multicast session R_i. Here by the size of a session, we mean the total number of source and destination nodes in that session. We assume that the multicast session sizes $z_0, z_1, ..., z_{M-1}$ are independent and identically distributed according to some cumulative distribution function $\mathcal{F}$ having mean $\mu_{\mathcal{F}}$. We can randomly select a pair of nodes from N nodes, then the probability of one of the pair of nodes be a source or a destination is

$$\frac{2}{C_N^2} = \frac{4}{N(N-1)}.$$

Thus, for each of multicast R_i having size z_i, the probability of the node j which is source or destination selected is $\frac{4z_i}{N(N-1)}$.

We can estimate the expected value of the following which approximates the lower bound [9].

$$\hat{L} = \sum_{i-0}^{N-1} \left\lceil \frac{k_i}{g} \right\rceil \tag{2}$$

where k_i is the number of multicast sessions that having node i as either source or one of the destinations.

Then the expected value of the approximate lower bound $\hat{L}$ is given [9] by

$$E(\hat{L}) = E\Big(\sum_{i-0}^{N-1} \left\lceil \frac{k_i}{g} \right\rceil \Big) = \sum_{i=0}^{N-1} E\Big(\left\lceil \frac{k_i}{g} \right\rceil \Big) = N \cdot E\Big(\left\lceil \frac{k}{g} \right\rceil \Big). \tag{3}$$

Note that

$$E\Big(\frac{k}{g}\Big) \leq E\Big(\left\lceil \frac{k}{g} \right\rceil \Big) \leq E\Big(\frac{k}{g} + 1\Big). \tag{4}$$

Also, the number of multicast sessions selecting a particular network node as source or one of the destinations can be written as

$$k = x_0 + x_1 + ... + x_{M-1}$$

where random variable x_i is distributed according to a Bernoulli trial, and it takes value 1 if the i-th multicast session R_i selects the node under consideration as source or one of the destination and 0 otherwise. Then,

$$\begin{aligned} E(k) = \sum_{i=0}^{M-1} E(x_i) &= \sum_{i=0}^{M-1} E\Big(E(x_i|z_i)\Big) = \sum_{i=0}^{M-1} E\Big(\tfrac{4z_i}{N(N-1)}\Big) \\ &= \tfrac{4}{N(N-1)} \sum_{i=0}^{M-1} E(z_i) = \tfrac{4}{N(N-1)} M\mu_{\mathcal{F}}. \end{aligned} \tag{5}$$

Clearly, we have

$$E\Big(\frac{k}{g}\Big) = \frac{4M\mu_{\mathcal{F}}}{N(N-1)g} \tag{6}$$

and

$$E\Big(\frac{k}{g}+1\Big)=\frac{4M\mu_{\mathcal{F}}}{N(N-1)g}+1. \tag{7}$$

Combine (3),(4),(6),(7), we can get

$$\frac{4M\mu_{\mathcal{F}}}{(N-1)g}\le E(\hat{L})\le\frac{4M\mu_{\mathcal{F}}}{(N-1)g}+N \tag{8}$$

Note that

$$E(\hat{L})=\frac{4M\mu_{\mathcal{F}}}{(N-1)g},\ \text{if}\ \frac{4M\mu_{\mathcal{F}}}{(N-1)g}\gg N.$$

From the above, we know that the lower bound we obtained depends on the number of nodes in the network by considering a pair of nodes.

Upper bound. Now we investigate some upper bounds on the number of ADMs required in the network. We study the upper bounds for the GreedyGroom(M) algorithms.

In the algorithm, we randomly group the traffic requests into clusters of g requests each. Then for each g requests which can model as a g-colorable circular-arc graph, we use the minimum number of colors to coloring the graph and grouping g colors to form one wavelength. The circular-arc graph can be colored by Karapetian's algorithm [10] using at most $\lfloor 3\chi_i/2\rfloor$ colors, then the number of ADMs required in the network as follows:

$$U_m=N^{'}\cdot\sum\nolimits_{i=1}^{\lceil M/g\rceil}\left\lceil\frac{\left\lfloor\frac{3}{2}\chi_i\right\rfloor}{g}\right\rceil \tag{9}$$

where $N^{'}$ denote the number of nodes that act as source or destination node for at least one multicast request. Clearly, $N\ge N^{'}$.

Then we recall some results presented in [9],

$$L_2=z_{\min}\left\lceil\frac{\chi_i}{g}\right\rceil, \tag{10}$$

where L_2 is a lower bound other than one previously presented, and z_{min} is the minimum size of all the multicast sessions. Let z_{avg} be the average size of multicast sessions, i.e. let

$$\sum_{i=0}^{N-1}k_i=z_{avg}M. \tag{11}$$

Additionally,

$$k_i\le 2\chi_i$$

which is proved in [9].

Using the above results, we get

$$\begin{aligned} U_m &= N^{'} \cdot \sum_{i=1}^{\lceil M/g \rceil} \left\lceil \frac{\lfloor \frac{3}{2}\chi_i \rfloor}{g} \right\rceil \leq N^{'} \cdot \sum_{i=1}^{\lceil M/g \rceil} \left\lceil \frac{3}{2}\frac{\chi_i}{g} \right\rceil \\ &\leq N^{'} \cdot \sum_{i=1}^{\left\lceil \frac{\sum_{i=0}^{N-1} k_i}{g z_{avg}} \right\rceil} \left\lceil \frac{3}{2}\frac{\chi_i}{g} \right\rceil \leq N^{'} \cdot \sum_{i=1}^{\left\lceil \frac{\sum_{i=0}^{N-1} 2\chi_i}{g z_{avg}} \right\rceil} \left\lceil \frac{3}{2}\frac{\chi_i}{g} \right\rceil \\ &\leq N^{'} \cdot \sum_{i=1}^{\left\lceil \frac{\sum_{i=0}^{N-1} \chi_i}{g} \right\rceil} \left\lceil \frac{3}{2}\frac{\chi_i}{g} \right\rceil \leq N^{'} \cdot \sum_{i=1}^{L} \left\lceil \frac{3}{2}\frac{\chi_i}{g} \right\rceil. \end{aligned} \tag{12}$$

Here the fourth inequality is due to the fact that $z_{avg} \geq 2$. Let

$$\frac{\chi_i}{g} = 2n + \delta + \varepsilon \tag{13}$$

where n is a non-negative integer, $\delta \in \{0, 1\}$ and $0 \leq \varepsilon \leq 1$.

From (10) and (13), we have

$$\begin{aligned} L_2 &= z_{\min} \left\lceil \frac{\chi_i}{g} \right\rceil = z_{\min} \lceil 2n + \delta + \varepsilon \rceil \\ &= z_{\min}(2n + \delta + \lceil \varepsilon \rceil). \end{aligned} \tag{14}$$

Thus,

$$\begin{aligned} U_m &\leq N^{'} \sum_{i=1}^{L} \left\lceil \frac{3}{2}\frac{\chi_i}{g} \right\rceil \\ &= N^{'} \sum_{i=1}^{L} \left\lceil \frac{3}{2}(2n + \delta + \varepsilon) \right\rceil \qquad (15) \\ &= N^{'} \sum_{i=1}^{L} \left\lceil 3n + 3\lceil \frac{\delta + \varepsilon}{2} \rceil \right\rceil. \qquad (16) \end{aligned}$$

Now we can see two cases possibly.

1) $\delta + \varepsilon = 0 \Leftrightarrow \delta = 0, \varepsilon = 0$

$$L_2 = z_{\min}(2n + \delta + \lceil \varepsilon \rceil) = 2n z_{\min}.$$

Clearly,

$$\begin{aligned} U_m &\leq N^{'} \cdot \sum_{i=1}^{L} (3n + 3 \left\lceil \frac{\delta + \varepsilon}{2} \right\rceil) \\ &= N^{'} \cdot \sum_{i=1}^{L} (3n) = 3nN^{'} \cdot L = \frac{3L_2}{2z_{\min}} N^{'} \cdot L, \end{aligned}$$

and

$$L_2 = z_{\min} \left\lceil \frac{\chi_i}{g} \right\rceil = \frac{z_{\min}}{N-1} \sum_{i=0}^{N-1} \left\lceil \frac{\chi_i}{g} \right\rceil = \frac{z_{\min}}{N-1} \cdot L,$$

then

$$U_m \leq \frac{3L_2}{2z_{\min}} N^{'} \cdot L \leq 3N^{'} \cdot \frac{z_{\min} \cdot L}{2z_{\min}(N-1)} \cdot L$$
$$\leq \frac{3N^{'}}{2(N-1)} L \cdot L \leq \frac{3N}{2(N-1)} \cdot L_3 \tag{17}$$

where L_3 is another lower bound on the number of required ADMs given by

$$L_3 = \max\left\{L^2, N^{'}\right\}.$$

2) $\delta + \varepsilon > 0 \Leftrightarrow 0 < \delta + \varepsilon < 2$
In this case, we get

$$L_2 = z_{\min}(2n + \delta + \lceil \varepsilon \rceil) \geq z_{\min}(2n+1), \tag{18}$$

and

$$\begin{aligned} U_m &\leq N^{'} \cdot \sum\nolimits_{i=1}^{L} \left(3n + 3\left\lceil \frac{\delta+\varepsilon}{2} \right\rceil\right) \\ &= N^{'} \cdot \sum\nolimits_{i=1}^{L} \left(3n+3\right) = N^{'} \cdot L\Big(\frac{3}{2}\big(2n+1\big) + \frac{3}{2}\Big) \\ &\leq N^{'} \cdot L\Big(\frac{3}{2}\frac{L_2}{z_{\min}} + \frac{3}{2}\Big) = \frac{3}{2}\frac{L_2}{z_{\min}} N^{'} \cdot L + \frac{3}{2} N^{'} \cdot L \\ &= \frac{3}{2}\frac{L \cdot z_{\min}}{(N-1)z_{\min}} N^{'} \cdot L + \frac{3}{2} N^{'} \cdot L \\ &= \frac{3}{2}\frac{L}{N-1} N^{'} \cdot L + \frac{3}{2} N^{'} \cdot L \\ &\leq \frac{3}{2}\frac{N^{'}}{N-1} \cdot L^2 + \frac{3}{2} N^{'} \cdot L \\ &\leq \frac{3}{2}\left(\frac{N^{'}}{N-1} + 1\right) \cdot L_3 \\ &\leq \frac{3}{2}\left(\frac{N}{N-1} + 1\right) \cdot L_3. \end{aligned} \tag{19}$$

From the above results, we observe that the algorithm has an approximation ratio $3(N/(N-1)+1)/2$, which is better than the previously computed approximation ratio of $\frac{3(N+z_{\min})}{2z_{\min}}$ whenever $z_{min} \leq N-1$.

5 Conclusion

In this paper, we studied the problem of grooming non-uniform multicast traffic on a unidirectional SONET/WDM ring. We consider two different costs, (i) the number of wavelengths, and (ii) the number of ADMs. We observe that minimizing the number of wavelengths can be modeled as a standard graph coloring problem. Then we presented an algorithm to groom the multicast requests greedily, and show that a better approximation ratio of $\frac{3}{2}(\frac{N}{N-1} + 1)$.

References

1. Billah, A.R.B., Wang, B., Awwal, A.A.S.: Effective Traffic Grooming in WDM Rings. In: Proc. of IEEE Globecom, vol. 3, pp. 2726–2730 (2002)
2. Chiu, A.L., Modiano, E.H.: Traffic Grooming Algorithms for Reducing Electronic Multiplexing Costs in WDM Ring Networks. IEEE/OSA Journal of Lightwave Technology 18(1), 2–12 (2000)
3. Chowdhary, G.V., Murthy, C.S.R.: Grooming of Multicast Sessions in WDM Mesh Networks. In: Proc. of the 1st Workshop on Traffic Grooming in WDM Networks, San Jose, USA (2004)
4. Farahmandz, F., Huang, X., Jue, J.P.: Efficient Online Traffic Grooming Algorithms in WDM Mesh Networks with Drop-and-Continue Node Architecture. In: Proc. of IEEE Broadnets, pp. 180–189 (2004)
5. Garey, M., Johnson, D., Miller, G., Papadimitriou, C.: The Complexity of Coloring Circular Arcs and Chords. SIAM Journal on Algebraic and Discrete Methods 1(2), 216–227 (1980)
6. Gerstel, O., Ramaswami, R.: Cost-Effective Traffic Grooming in WDM Rings. IEEE/ACM Transactions of Networking 8(5), 618–630 (2000)
7. Huang, X., Farahmandz, F., Jue, J.P.: Multicast Traffic Grooming in Wavelength-Routed WDM Mesh Networks using Dynamically Changing Light-Trees. IEEE/OSA Journal of Lightwave Technology 23(10), 3178–3187 (2005)
8. Madhyastha, H.V., Srinivas, N., Chowdhary, G.V., Murthy, C.S.R.: Grooming of Multicast Sessions in WDM Ring Networks. In: Torra, V., Narukawa, Y. (eds.) MDAI 2008. LNCS, vol. 5285, pp. 1–12. Springer, Heidelberg (2008)
9. Rawat, A., La, R., Marcus, S., Shayman, M.: Grooming Multicast Traffic in Unidirectional SONET/WDM Rings. IEEE Journal on Selected Areas in Communications - Optical Communications and Networking Series 25(6), 70–83 (2007)
10. Karapetian, I.A.: On the Coloring Circular Arc Graphs. Docladi (Reports) of the Academy of Science of the Armenian Soviet Socialist Republic (1980)
11. Tucker, A.: Coloring a Family of Circular Arcs. SIAM Journal of Applied Mathematics 29(3), 493–502 (1975)
12. Zhang, X., Qiao, C.: Scheduling in Unidirectional WDM Rings and its Extensions. In: Vicedo, J.L., Martínez-Barco, P., Muńoz, R., Saiz Noeda, M. (eds.) EsTAL 2004. LNCS, vol. 3230, pp. 208–219. Springer, Heidelberg (2004)
13. Zhang, X., Qiao, C.: An Effective and Comprehensive Approach for Traffic Grooming and Wavelength Assignment in SONET/WDM Rings. IEEE/ACM Transactions On Networking 8(5), 608–617 (2000)
14. Zhu, K., Mukherjee, B.: Traffic Grooming in an Optical WDM Mesh Network. IEEE Journal on Selected Areas in Communications 20(1), 122–133 (2002)

Author Index