Franco P. Preparata
Xiaodong Wu
Jianping Yin (Eds.)

LNCS 5059

Frontiers in Algorithmics

Second International Workshop, FAW 2008
Changsha, China, June 2008
Proceedings

Lecture Notes in Computer Science 5059

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

Franco P. Preparata Xiaodong Wu
Jianping Yin (Eds.)

Frontiers in Algorithmics

Second International Workshop, FAW 2008
Changsha, China, June 19-21, 2008
Proceedings

Springer

Volume Editors

Franco P. Preparata
Brown University, Department of Computer Science
115 Waterman St., Providence, RI 02912-1910, USA
E-mail: franco@cs.brown.edu

Xiaodong Wu
University of Iowa, Department of Electrical and Computer Engineering
4016 Seamans Center for the Engineering Arts and Sciences
Iowa City, IA 52242-1595, USA
E-mail: xiaodong-wu@uiowa.edu

Jianping Yin
National University of Defense Technology, School of Computer Science
Changsha, Hunan 410073, China
E-mail: jpyin@nudt.edu.cn

Library of Congress Control Number: 2008928564

CR Subject Classification (1998): F.2.2, G.2.1-2, G.1.2, G.1.6, C.2.2, I.3.5, E.1

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

ISSN 0302-9743
ISBN-10 3-540-69310-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-69310-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12277209 06/3180 5 4 3 2 1 0

Preface

The Annual International Frontiers in Algorithmics Workshop is a focused forum on current trends in research on algorithms, discrete structures, and their applications. It intends to bring together international experts at the research frontiers in those areas to exchange ideas and to present significant new results. The mission of the workshop is to stimulate the various fields for which algorithmics can become a crucial enabler, and to strengthen the ties between the Eastern and Western algorithmics research communities. The Second International Frontiers in Algorithmics Workshop (FAW 2008) took place in Changsha, China, June 19–21, 2008.

In response to the Call for Papers, 80 papers were submitted from 15 countries and regions: Canada, China, France, Germany, Greece, Hong Kong, India, Iran, Japan, Mexico, Norway, Singapore, South Korea, Taiwan, and the USA. After a six-week period of careful reviewing and discussion, the Program Committee accepted 32 submissions for presentation at the conference. These papers were selected for nine special focus tracks in the areas of biomedical informatics, discrete structures, geometric information processing and communication, games and incentive analysis, graph algorithms, internet algorithms and protocols, parameterized algorithms, design and analysis of heuristics, approximate and online algorithms, and machine learning. The program of FAW 2008 also included three keynote talks by Xiaotie Deng, John E. Hopcroft, and Milan Sonka.

We thank all Program Committee members and the external referees for their excellent work, especially given the demanding time constraints. It has been a wonderful experience to work with them. We would like to express our gratitude to the three invited speakers and all the people who submitted papers for consideration: They all contributed to the high quality of the conference. We also thank the members of the Steering Committee, especially Danny Z. Chen and Xiaotie Deng, for their leadership, advice and assistance on crucial matters concerning the conference.

Finally, we would like to thank the Organizing Committee, led by Peng Zou and Hao Li, who worked tirelessly to put in place the logistical arrangements of FAW 2008 and to make this conference a success. We are also very grateful to our generous sponsors, the School of Computer Science, National University of Defense Technology, for kindly offering the financial and clerical support that made the conference possible and enjoyable.

June 2008

Franco P. Preparata
Xiaodong Wu
Jianping Yin

Organization

FAW 2008 was organized and sponsored by the School of Computer Science, National University of Defense Technology, China.

Executive Committee

Conference Chair	Huowang Chen (National University of Defense Technology, China)
Program Co-chairs	Franco P. Preparata (Brown University, USA) Xiaodong Wu (University of Iowa, USA) Jianping Yin (National University of Defense Technology, China)
Organizing Chair	Peng Zou (National University of Defense Technology, China)
Organizing Co-chair	Hao Li (Laboratoire de Recherche en Informatique, France)

Program Committee

Nancy Amato (Texas A&M University, USA)
Tetsuo Asano (Japan Advanced Institute of Science and Technology, Japan)
Mikhail Atallah (Purdue University, USA)
David A. Bader (Georgia Institute of Technology, USA)
Chandra Chekuri (University of Illinois at Urbana-Champaign, USA)
Jianer Chen (Texas A&M University, USA)
Kyung-Yong Chwa (Korea Advanced Institute of Science Technology, Korea)
Bhaskar DasGupta (University of Illinois at Chicago, USA)
Qizhi Fang (Ocean University of China, China)
Michael R. Fellows (University of Newcastle, Australia)
Michael T. Goodrich (University of California Irvine, USA)
Horst W. Hamacher (University of Kaiserslautern, Germany)
Pinar Heggernes (University of Bergen, Norway)
Kazuo Iwama (Kyoto University, Japan)
Der-Tsai Lee (Institute of Information Science Academia Sinica, Taiwan)
Erricos J. Kontoghiorghes (Birkbeck College, University of London, UK)
Lian Li (Hefei University of Technology, China)
Xiang-Yang Li (Illinois Institute of Technology, USA)
Andy McLennan (University of Queensland, Australia)
Detlef Seese (Universität Karlsruhe, Germany)
Kasturi Varadarajan (University of Iowa, USA)
Guoqing Wu (Wuhan University, China)

Jinhui Xu (State University of New York at Buffalo, USA)
Jinyun Xue (Jiangxi Normal University, China)
Chee Yap (New York University, USA)
Ramin Zabih (Cornell University, USA)
Louxin Zhang (National University of Singapore, Singapore)

Referees

Virat Agarwal
Sang Won Bae
Kung-Sik Chan
Aparna Chandramowlishwaran
Xi Cheng
Manisha Gajbe
Yong Guan
AnChen Hsiao
Seunghwa Kang
Mong-jen Kao
Cheng-Chung Li
Chung-Shou Liao
Tien-Ching Lin
Yun Liu
Daniel Lokshtanov
Jun Long
Kamesh Madduri
Xufei Mao
Amrita Mathuriya
Daniel Meister
Jun Ni
Peter Noel
Bonizzoni Paola
Frank Ruskey
Shaojie Tang
Chung-Hung Tsai
Andreas Wahle
Hom-Kai Wang
Yajun Wang
Xiaobing Wu
Yongan Wu
Ping Xu
Xiaohua Xu
Lei Xu
Teng-Kai Yu
Honghai Zhang
Guomin Zhang
Yongding Zhu

Table of Contents

Fixed Point Computation and Equilibrium
Abstract of Keynote Talk

Xiaotie Deng

Department of Computer Science
City University of Hong Kong, Hong Kong
deng@cs.cityu.edu.hk

The rise of the Internet has created a surge of human activities that make computation, communication and optimization of participating agents accessible at micro-economic levels. Fundamental equilibrium problems of games and markets, including algorithms and complexity as well as applications have become active topics for complexity studies. Algorithmic Game Theory has emerged as one of the highly interdisciplinary fields, in response to (and sometimes anticipating) the need of this great revolution, intersecting Economics, Mathematics, Operations Research, Numerical Analysis, and Computer Science.

The mathematical model underlying various forms of equilibrium is the fixed point concept. The discovery, and applications, of the close relationship between the fixed point and equilibrium concepts has played a major role in shaping Mathematical Economics. In computation, it continues to influence our understanding of complexity and algorithmic design for equilibrium problems. In this talk, I will discuss some recent development in fixed point computation, together with application problems, such as sponsored search auctions and envy-free cake cuttings.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, p. 1, 2008.

Computer Science in the Information Age
Abstract of Keynote Talk

John Hopcroft

Department of Computer Science
Cornell University, USA
jeh@cs.cornell.edu

The last forty years have seen computer science evolve as a major academic discipline. Today the field is undergoing a major change. Some of the drivers of this change are the internet, the world wide web, large sensor networks, large quantities of information in digital form and the wide spread use of computers for accessing information. This change is requiring universities to revise the content of computer science programs. This talk will cover the changes in the theoretical foundations needed to support information access in the coming years.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, p. 2, 2008.

Knowledge-Based Approaches to Quantitative Medical Image Analysis and Image-Based Decision Making

Abstract of Keynote Talk

Milan Sonka

Department of Electrical and Computer Engineering
University of Iowa
Iowa City, IA 52242, USA
milan-sonka@uiowa.edu

Widespread use of three-dimensional tomographic imaging scanners and other imaging modalities has revolutionized medical care as we know it today. The ever-increasing sizes of acquired data volumes are making conventional visual analysis of image data and consequent image-based decision making time consuming, tedious, and difficult to perform at the time available in busy clinical practice. The field of quantitative medical imaging, which has considerably matured in the past decade, is increasingly promising to solve many current problems that radiologists, cardiologists, orthopedists, and many other physicians are facing on a daily basis.

Accurate and reliable segmentation and subsequent quantitative description of multi-dimensional and/or multi-modality medical image data is one of the primary pre-requisites to more complex medical image analyses and decision making. The presentation will give a broad overview of the state of the art of medical image analysis and will focus on several inter-disciplinary projects requiring a direct and close collaboration between physicians, computer scientists, biostatisticians, and medical image analysis researchers. In the biomedical research context, spanning from cell images to small and large animals and to humans, a number of knowledge-based medical image analysis methods and approaches will be presented and their utility demonstrated. Examples of research applications as well as commercially available solutions will be presented and discussed.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, p. 3, 2008.

Optimal Field Splitting, with Applications in Intensity-Modulated Radiation Therapy*

Danny Z. Chen and Chao Wang**

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
{chen,cwang1}@cse.nd.edu

Abstract. We consider an interesting geometric partition problem called *field splitting*, which arises in intensity-modulated radiation therapy (IMRT). IMRT is a modern cancer treatment technique that delivers prescribed radiation dose distributions, called *intensity maps* (IMs) and defined on uniform grids, to target tumors via the help of a device called the *multileaf collimator* (MLC). The delivery of each IM requires a certain amount of *beam-on time*, which is the total time when a patient is exposed to actual irradiation during the delivery. Due to the *maximum leaf spread constraint* of the MLCs (i.e., the size and range of an MLC are constrained by its mechanical design), IMs whose widths exceed a given threshold value cannot be delivered by the MLC as a whole, and thus must be split into multiple subfields (i.e., subgrids) so that each subfield can be delivered separately by the MLC. In this paper, we present the first efficient algorithm for computing an optimal field splitting that guarantees to minimize the total beam-on time of the resulting subfields subject to a new constraint that the maximum beam-on time of each individual subfield is no larger than a given a threshold value. Our basic idea is to formulate this field splitting problem as a special integer linear programming problem. By considering its dual problem, which turns out to be a shortest path problem on a directed graph with both positive and negative edge weights, we are able to handle efficiently the upper-bound constraint on the allowed beam-on time of each resulting individual subfield. We implement our new field splitting algorithm and give some experimental results on comparing our solutions with those computed by the previous methods.

1 Introduction

In this paper, we consider an interesting geometric partition problem called *field splitting*, which arises in intensity-modulated radiation therapy (IMRT). IMRT is a modern cancer treatment technique that aims to deliver highly conformal

* This research was supported in part by the National Science Foundation under Grant CCF-0515203.

** Corresponding author.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 4–15, 2008.

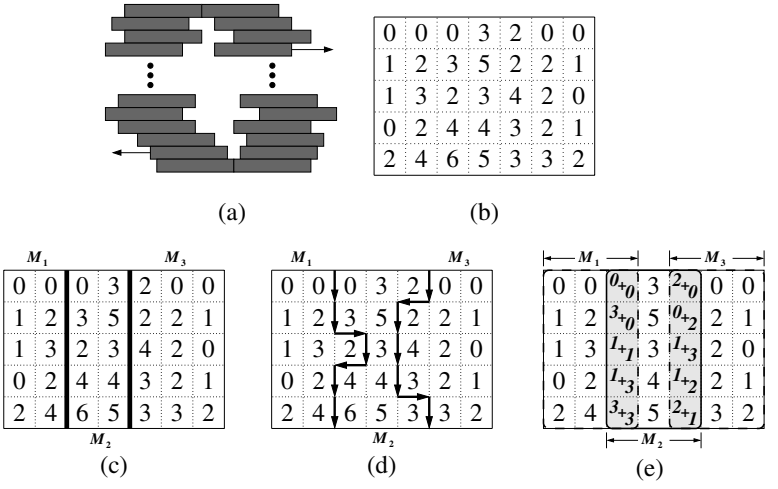


Fig. 1. (a) An MLC. (b) An IM. (c)-(e) Examples of splitting an IM into three subfields, M_1, M_2, and M_3, using vertical lines, y-monotone paths, and with overlapping, respectively. The dark cells in (e) show the overlapping regions of the subfields; the prescribed dose value in each dark cell is divided into two parts and allocated to two adjacent subfields.

prescribed radiation dose distributions, called *intensity maps* (IMs), to target tumors while sparing the surrounding normal tissue and critical structures. An IM is a dose prescription specified by a set of nonnegative integers on a uniform 2-D grid (see Figure 1(b)). The value in each grid cell indicates the intensity level of prescribed radiation at the body region corresponding to that IM cell. The delivery is done by a set of cylindrical radiation beams orthogonal to the IM grid.

One of the current most advanced tools for IM delivery is the *multileaf collimator* (MLC) [14,15]. An MLC consists of many pairs of tungsten alloy leaves of the same rectangular shape and size (see Figure 1(a)). The opposite leaves of each pair are aligned to each other, and can move left or right to form a y-monotone rectilinear polygonal beam-shaping region (the cross-section of a cylindrical radiation beam is shaped by such a region). All IM cells exposed under a radiation beam receive a uniform radiation dose proportional to the exposure time. The mechanical design of the MLCs restricts what kinds of beam-shaping regions are allowed [14]. A common constraint is called the **maximum leaf spread**: No leaf can move away from the vertical center line of the MLC by more than a threshold distance (e.g., 12.5 *cm* for Elekta MLCs). This means that an MLC cannot enclose any IM whose width exceeds a threshold value.

A key criterion for measuring the quality of IMRT treatment is the **beam-on time**, which is the total time while a patient is exposed to actual irradiation. Minimizing the beam-on time without compromising the prescribed dose distributions is an effective way to enhance the efficiency of the radiation machine, called the *monitor-unit (MU) e ciency* in medical literature, and to reduce the

patient's risk under irradiation [2]. The beam-on time also constitutes a good portion of the total treatment time [1,3,14,15] in IMRT. Thus, minimizing the beam-on time lowers the treatment cost of each patient, enhances the treatment quality, and increases the patient throughput of the hospitals.

On one of the most popular MLC systems called Varian, for example, the maximum leaf spread constraint limits the maximum allowed field width to about 15 cm. Hence, this necessitates a large-width IM field to be split into two or more adjacent subfields, each of which can be delivered separately by the MLC subject to the maximum leaf spread constraint [7,9,16]. But, such IM splitting may result in a prolonged beam-on time and thus adversely affect the treatment quality. The **field splitting problem**, roughly speaking, is to split an IM of a large width into multiple subfields whose widths are all no bigger than a threshold value, such that the total beam-on time of the resulting subfields is minimized. While splitting an IM into multiple subfields, it is also clinically desirable to keep the beam-on time of each resulting individual subfield under some threshold value. The reason is that, during the delivery of each subfield, the patient body may move, and the longer the beam-on time of a subfield, the higher the chance of body motion. Imposing an upper bound onto the beam-on time of each subfield is helpful to ensuring the accuracy of the IMRT treatment. This upper-bound constraint on the beam-on time of the resulting subfields has not been considered by previous field splitting algorithms before, which we will study in this paper.

Based on the MLC beam-shaping features, there are three ways to split an IM (see Figures 1(c)-1(e)): (1) splitting using vertical lines; (2) splitting using y-monotone paths; (3) splitting with overlapping. Note that in versions (1) and (2), an IM cell belongs to exactly one subfield; but in version (3), a cell can belong to two adjacent subfields, with a nonnegative value in each of these two subfields, and in the resulting sequence of subfields, each subfield is allowed to overlap only with the subfield immediately before and the one immediately after it. Clearly, splitting using y-monotone paths is a generalization of splitting using vertical lines (and its solution quality can in fact be considerably better), while splitting with overlapping is in a sense a generalization of splitting using y-monotone paths.

To characterize the **minimum beam-on time** (**MBT**), Engel [8] showed that for an IM M of size $m \times n$, when n is no larger than the maximum allowed field width w, the minimum beam-on time (MBT) of M is captured by the following formula:

$$MBT(M) = \max_{i=1}^{m}\{M_{i,1} + \sum\nolimits_{j=2}^{n} \max\{0, M_{i,j} - M_{i,j-1}\}\} \quad (1)$$

Engel also described a class of algorithms achieving this minimum value. Geometrically, if we view each row of an IM as representing a directed left-to-right x-monotone rectilinear curve f, called the *dose profile curve* (see Figure 2(a)), then the MBT of an IM row is actually the total sum of lengths of all the upward edges on f. The meaning of Formula (1) may be explained as follows: Each IM row is delivered by one pair of MLC leaves; for an upward edge e, say, of length l, on the dose profile curve of the IM row, the tip of the left MLC leaf for this row must stay at the x-coordinate of e for at least l time units (assuming one unit

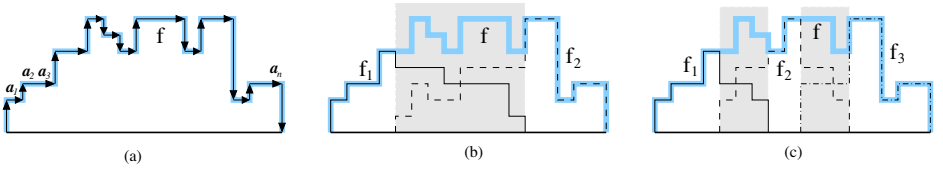


Fig. 2. (a) The dose profile curve of one row of an IM. The MBT (minimum beam-on time) of the IM row is equal to the sum of the lengths of all upward edges on the curve. (b) Splitting the IM row in (a) into two subfields with overlapping. (c) Splitting the IM row in (a) into three subfields with overlapping.

of dose is delivered in one unit of time) while the beam is on, so that the dose difference resulted at the x-coordinate of e is l. The *max* operator in the above formula reflects the fact that since all MLC leaf pairs are in use simultaneously during the delivery of the IM, the overall MBT of a subfield is determined by the maximum MBT value over all the m rows of the subfield.

A few known field splitting algorithms aim to minimize the *total MBT*, i.e., the *sum* of the MBTs, of the resulting subfields. Kamath *et al.* [11] first gave an $O(mn^2)$ time algorithm to split a size $m \times n$ IM using vertical lines into at most three subfields (i.e., $n \leq 3w$ for their algorithm, where w is the maximum allowed field width). Wu [17] formulated the problem of splitting an IM of an arbitrary width into $k \geq 3$ subfields using vertical lines as a k-link shortest path problem, yielding an $O(mnw)$ time algorithm. Kamath *et al.* [10] studied the problem of splitting an IM into at most three *overlapping* subfields, minimizing the total MBT of the resulting subfields; they proposed a greedy algorithm that produces optimal solutions in $O(mn)$ time when the overlapping regions of the subfields are fixed. Chen and Wang [5] gave an $O(mn + m\Delta^{d-2})$ time algorithm for optimally splitting an IM into d overlapping subfields whose overlapping regions need not be fixed, where $\Delta = w\lceil n/w \rceil - n + 1$ and $d = \lceil n/w \rceil \geq 1$ is the minimum number of subfields required to deliver M subject to the maximum allowed field width w, and an $O(mn + m^{d-2}\Delta^{d-1} \log(m\Delta))$ time algorithm for optimally splitting an IM into d subfields with y-monotone paths. Wu *et al.* [18] gave an $O(mn)$ time algorithm for the field splitting problem that minimizes the *total complexity* (a criterion closely related to the total beam-on time) of the resulting subfields, and showed how to minimize the maximum complexity of the resulting subfields in linear time when the overlapping regions of the subfields are fixed. Chen *et al.* [4] also considered field splitting problems based on other clinical criteria.

In this paper, we study the following **constrained field splitting with overlapping (CFSO)** problem. Given an IM M of size $m \times n$, a maximum field width $w > 0$, and a threshold beam-on time $T_{th} > 0$, split M into $d = \lceil n/w \rceil$ overlapping subfields $M_1, M_2, \ldots, M_d$, each with a width $\leq w$, such that (1) the MBT of each individual subfield M_i is no bigger than T_{th}, and (2) the total MBT of the d subfields is minimized (e.g., Figures 2(b)-2(c)). Here, d is the minimum number of subfields required to deliver M subject to the maximum allowed field width w.

To our best knowledge, no previous algorithms consider computing an optimal field splitting of IMs with the minimum total MBT subject to the upper-bound constraint on the MBT of each resulting subfield. The CFSO problem is clinically useful, as we discussed earlier in this section.

Our results in this paper are summarized as follows.

1. We present the first efficient algorithm for computing an optimal solution of the constrained field splitting problem (CFSO) defined in this section. Based on the integer linear programming framework of Chen and Wang [5] and graph optimization techniques, we extend the approach in [5] to handling the upper-bound constraint on the beam-on times of the individual subfields. Our algorithm runs in $O(mn + (md^2 + d^3)\Delta^{d-2})$ time, where $\Delta = w\lceil n/w \rceil - n + 1$ $(= O(w))$ and $d = \lceil n/w \rceil \geq 1$. It should be pointed out that in the current clinical settings, the value of d is usually not bigger than 3. Thus our algorithm is an optimal $O(mn)$ time solution for most of the current practical clinical applications.
2. One special feature of our field splitting algorithm is that it can compute a trade-off curve between the total MBT of the resulting subfields and the threshold value T_{th} for the MBT of each individual subfield. Since the value of T_{th} directly affects the treatment quality, this feature actually gives a trade-off between the treatment efficiency and treatment quality.
3. We implement our new field splitting algorithm and provide some experimental results on comparing our solutions with those computed by the previous methods using both clinical IMs and randomly generated IMs.

The basic idea for our CFSO algorithm is to formulate the problem as a special integer linear programming problem. By considering its dual problem, which turns out to be a shortest path problem on a directed graph with both positive and negative edge weights, we are able to handle efficiently the upper-bound constraint on the allowed beam-on time of each resulting individual subfield.

2 Constrained Field Splitting with Overlapping (CFSO)

In this section, we present our algorithm for solving the constrained field splitting with overlapping (CFSO) problem.

2.1 Notation and Definitions

We say that intervals $[\mu_1, \nu_1], [\mu_2, \nu_2], \ldots, [\mu_d, \nu_d]$ $(d \geq 2)$ form an *interweaving list* if $\mu_1 < \mu_2 \leq \nu_1 < \mu_3 \leq \nu_2 < \mu_4 \leq \cdots < \mu_{k+1} \leq \nu_k < \cdots < \mu_d \leq \nu_{d-1} < \nu_d$. For a subfield S restricted to begin from column $\mu + 1$ and end at column ν $(\mu + 1 \leq \nu)$, we call $[\mu, \nu]$ the *bounding interval* of S. We say that subfields $S_1, S_2, \ldots, S_d$ form a *chain* if their corresponding bounding intervals form an interweaving list (i.e., subfields S_k and S_{k+1} are allowed to overlap, for every $k = 1, 2, \ldots, d-1$). Further, $t = (t_1, t_2, \ldots, t_d) \in \mathbb{Z}_+^d$ is called the *MBT tuple* of a chain of d subfields $S_1, S_2, \ldots, S_d$ if t_k is the MBT of S_k $(k = 1, 2, \ldots, d)$.

For two tuples $t = (t_1, t_2, \ldots, t_d)$ and $\tau = (\tau_1, \tau_2, \ldots, \tau_d)$, we say $t \leq \tau$ if $t_k \leq \tau_k$ for every $k = 1, 2, \ldots, d$.

2.2 The General Row Splitting (GRS) Problem

We first review a basic case of the CFSO problem, called the **general row splitting (GRS) problem**, which is defined as follows: Given a vector (row) $\alpha \in \mathbb{Z}_+^n$ ($n \geq 3$), a d-tuple $\tau \in \mathbb{Z}_+^d$ ($d \geq 2$), and an interweaving interval list IL: $I_k = [\mu_k, \nu_k]$, $k = 1, 2, \ldots, d$, with $\mu_1 = 0$ and $\nu_d = n$, split α into a chain of d overlapping subfields $S_1, S_2, \ldots, S_d$ such that the bounding interval of each S_k is I_k ($k = 1, 2, \ldots, d$) and the MBT d-tuple $t = (t_1, t_2, \ldots, t_d)$ of the resulting subfield chain (i.e., t_k is the MBT of the subfield S_k) satisfies $t \leq \tau$. Denote by $GRS(\alpha, \tau, IL)$ the GRS problem on the instance α, τ, and IL.

Chen and Wang [5] gave the following result on the GRS problem, which will be useful to our CFSO algorithm.

Theorem 1. ([5]) *$GRS(\alpha, \tau, IL)$ has a solution if and only if $\tau \in P \cap \mathbb{Z}^d$, where*

$$P = \left\{ (t_1, t_2, \ldots, t_d) \in \mathbb{R}^d \;\middle|\; \sum_{q=k}^{k'} t_q \geq \eta_{k'} - \xi_{k-1}, \forall (k, k') : 1 \leq k \leq k' \leq d \right\}$$

and η_k's and ξ_k's are defined by

$$\eta_k = \begin{cases} c_1 & k = 1 \\ \sum_{q=1}^{k-1} \rho_q + \sum_{q=1}^{k} c_q & 2 \leq k \leq d \end{cases}$$

$$\xi_k = \begin{cases} \sum_{q=1}^{k} \rho_q + \sum_{q=1}^{k} c_q - \alpha_{\nu_k+1} & 1 \leq k \leq d-1 \\ \sum_{q=1}^{d-1} \rho_q + \sum_{q=1}^{d} c_q & k = d \end{cases}$$

with

$$c_k = \begin{cases} \alpha_1 + \sum_{j=2}^{\mu_2} \max\{0, \alpha_j - \alpha_{j-1}\} & k = 1 \\ \sum_{j=\nu_{k-1}+2}^{\mu_{k+1}} \max\{0, \alpha_j - \alpha_{j-1}\} & 2 \leq k \leq d-1 \\ \sum_{j=\nu_{k-1}+2}^{n} \max\{0, \alpha_i - \alpha_{j-1}\} & k = d \end{cases}$$

$$\rho_k = \max\{\alpha_{\nu_k+1}, \sum_{j=\mu_{k+1}+1}^{\nu_k+1} \max\{0, \alpha_j - \alpha_{j-1}\}\} \qquad 1 \leq k \leq d-1$$

Moreover, given any $\tau \in P \cap \mathbb{Z}^d$, we can solve $GRS(\alpha, \tau, IL)$ in $O(n)$ time.

Note that P defined in Theorem 1 is a polytope in $\mathbb{R}^d$ specified by a polynomial number (in terms of d) of linear constraints.

2.3 The Constrained Field Splitting with Fixed Overlapping (CFSFO) Problem

In this section, we study a special case of the constrained field splitting with overlapping (CFSO) problem, i.e., the case when the sizes and positions of the d sought subfields are all fixed. Precisely, the **constrained field splitting with fixed overlapping (CFSFO) problem** is: Given an IM M of size $m \times n$, an interweaving list IL of d intervals $I_1, I_2, \ldots, I_d$, and a threshold beam-on time $T_{th} > 0$, split M into a chain of d overlapping subfields $M_1, M_2, \ldots, M_d$, such that (1) I_k is the bounding interval of M_k (for each $k = 1, 2, \ldots, d$), (2) the MBT of every subfield M_k ($1 \leq k \leq d$) is no bigger than T_{th}, and (3) the total MBT of the d resulting subfields is minimized.

Denote by $CFSFO(M, IL, T_{th})$ the CFSFO problem on the instance M, IL, and T_{th}. Recall that the MBT of a subfield M_k is the maximum MBT value

over all the m rows of M_k. Therefore, for any feasible solution $(M_1, M_2, \ldots, M_d)$ of $CFSFO(M, IL, T_{th})$, let $\tau := (MBT(M_1), MBT(M_2), \ldots, MBT(M_d))$ be the corresponding MBT tuple, and denote by $M^{(i)}$ (resp., $M_k^{(i)}$) the i-th row of M (resp., M_k). For each $i = 1, 2, \ldots, m$, $(M_1^{(i)}, M_2^{(i)}, \ldots, M_d^{(i)})$ is clearly a solution of the GRS instance $GRS(M^{(i)}, \tau, IL)$. By Theorem 1, we have $\tau \in P_i \cap \mathbb{Z}^d$, where $P_i = \left\{(t_1, t_2, \ldots, t_d) \in \mathbb{R}^d \;\middle|\; \sum_{q=k}^{k'} t_q \geq \eta_{k'}^{(i)} - \xi_{k-1}^{(i)}, \forall (k, k') : 1 \leq k \leq k' \leq d\right\}$ (here, $\eta_k^{(i)}$'s and $\xi_k^{(i)}$'s are defined as in Theorem 1 except that α is substituted by $M^{(i)}$). Since the MBT of each individual subfield M_k is no bigger than T_{th}, we have $\tau \leq (T_{th}, T_{th}, \ldots, T_{th})$. It is thus easy to show that the CFSFO problem can be transformed to the following integer linear programming (ILP) problem:

$$\begin{aligned}
&\min \textstyle\sum_{k=1}^{d} \tau_k \\
&\text{subject to} \\
&\textstyle\sum_{q=k}^{k'} \tau_q \geq \max_{i=1}^{m} \{\eta_{k'}^{(i)} - \xi_{k-1}^{(i)}\}, \quad \forall (k, k') : 1 \leq k \leq k' \leq d \\
&\tau_k \leq T_{th}, \quad k = 1, 2, \ldots, d \\
&\tau = (\tau_1, \tau_2, \ldots, \tau_d) \in \mathbb{Z}^d
\end{aligned}$$

Clearly, the time taken by the transformation process is dominated by the time for computing the right hand side of the above ILP. For a fixed i, it is easy to show that all the $\eta_k^{(i)}$'s and $\xi_k^{(i)}$'s can be computed in $O(n)$ time. Once all the $\eta_k^{(i)}$'s and $\xi_k^{(i)}$'s are computed for every i, given (k, k'), $\max_{i=1}^{m} \{\eta_{k'}^{(i)} - \xi_{k-1}^{(i)}\}$ can be computed in $O(m)$ time. Therefore the time complexity of this transformation is $O(mn + md^2)$.

To solve this ILP problem efficiently, note that the constraint matrix of the ILP is a (0,1) interval matrix (i.e., each row of the constraint matrix is of the form $(0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0)$ with the 1's appearing consecutively) and is thus totally unimodular [12]. Based on well-known theory of linear programming and integer linear programming [12], this ILP problem can be solved optimally as the following linear programming (LP) problem, denoted by LP_I, by disregarding the integer solution constraint of the ILP:

$$\begin{aligned}
&\min \textstyle\sum_{k=1}^{d} \tau_k \\
&\text{subject to} \\
&\textstyle\sum_{q=k}^{k'} \tau_q \geq \max_{i=1}^{m} \{\eta_{k'}^{(i)} - \xi_{k-1}^{(i)}\}, \quad \forall (k, k') : 1 \leq k \leq k' \leq d \\
&\tau_k \leq T_{th}, \quad k = 1, 2, \ldots, d
\end{aligned}$$

We introduce a variable π_0, and define $\pi_k = \pi_0 + \sum_{q=1}^{k} \tau_q$, for each $k = 1, 2, \ldots, d$. Then LP_I is equivalent to the following LP problem, denoted by LP_{II}:

$$\begin{aligned}
&\max \pi_0 - \pi_d \\
&\text{subject to} \\
&\pi_k - \pi_{k'} \leq -\max_{i=1}^{m} \{\eta_{k'}^{(i)} - \xi_{k-1}^{(i)}\}, \quad \forall (k, k') : 0 \leq k < k' \leq d \\
&\pi_k - \pi_{k-1} \leq T_{th}, \quad k = 1, 2, \ldots, d
\end{aligned}$$

As shown in [13], the dual LP of LP_{II} is an s-t shortest path problem on a weighted directed graph G, that is, seeking a *simple* shortest path in G from a vertex s to another vertex t. (A path is *simple* if it does not contain any cycle.)

The weighted directed graph G for LP_{II} is constructed as follows. G has $d+1$ vertices v_0 $(=s)$, $v_1, \ldots, v_d$ $(=t)$. For each pair (k, k') with $0 \le k < k' \le d$, put a directed edge from v_k to $v_{k'}$ with a weight $-\max_{i=1}^{m}\{\eta_{k'}^{(i)} - \xi_{k-1}^{(i)}\}$; for each pair $(k, k-1)$ with $1 \le k \le d$, put a directed edge from v_k to v_{k-1} with a weight T_{th}. Clearly, G is a directed graph with $O(d)$ vertices and $O(d^2)$ edges. Since the edges of G may have both positive and negative weights, there are two possible cases to consider.

Case (1): G has no negative cycles. In this case, a simple s-to-t shortest path in G can be computed in $O(d^3)$ time by the Bellman-Ford algorithm [6]. Further, an optimal MBT tuple τ^* for the original ILP problem can be computed also in $O(d^3)$ time. We can then obtain the corresponding optimal splitting of M for $CFSFO(M, IL, T_{th})$ by solving m GRS instances, i.e., $GRS(M^{(i)}, \tau^*, IL)$, $i = 1, 2, \ldots, m$, which, by Theorem 1, takes a total of $O(mn)$ time.

Case (2): G has negative weight cycles. This case occurs when T_{th} is too small to be feasible. For this case, the Bellman-Ford algorithm on G, in $O(d^3)$ time, detects a negative weight cycle and reports the non-existence of a simple s-to-t shortest path. Correspondingly, the original CFSFO problem has no feasible solution for this case.

Hence, we have the following result for the CFSFO problem.

Theorem 2. *Given an IM M of size $m \times n$, an interweaving list IL of d intervals, and a threshold beam-on time $T_{th} > 0$, the problem $CFSFO(M, IL, T_{th})$ can be solved in $O(mn + md^2 + d^3)$ time.*

2.4 The Constrained Field Splitting with Overlapping (CFSO) Problem

We now solve the general CFSO problem on a size $m \times n$ IM M, for which the sizes and positions of the sought subfields are *not* fixed. One observation we use is that we can assume that the size of each resulting subfield is $m \times w$, where w is the maximum allowed field width. This is because we can always introduce columns filled with 0's to the ends of the subfield without changing its MBT. Also, note that among the d sought subfields, the positions of the leftmost and rightmost ones are already fixed. Based on these observations, it is sufficient to consider only $O(\Delta^{d-2})$ possible subfield chains, where $\Delta = dw - n + 1 = w\lceil n/w \rceil - n + 1$. Based on Theorem 2, the CFSO problem can be solved in $O((mn + md^2 + d^3)\Delta^{d-2})$ time, by solving $O(\Delta^{d-2})$ CFSFO problem instances.

We can further improve the time complexity of this CFSO algorithm, by exploiting the following observations.

(1) For each of the $O(\Delta^{d-2})$ CFSFO problem instances (on $O(\Delta^{d-2})$ possible subfield chains), we can stop (and continue with the next CFSFO instance, if any) once we obtain the corresponding optimal MBT tuple for that CFSFO instance. We need not find the corresponding actual optimal splitting of M, and this saves $O(mn)$ time (for solving the m GRS instances involved). Of course, we still need $O(mn)$ time to produce the optimal splitting of M for the CFSO problem once we determine the best subfield chain out of the $O(\Delta^{d-2})$ chains.

(2) For each CFSFO problem instance, its ILP transformation can be carried out in only $O(md^2)$ time, instead of $O(mn+md^2)$ time, by performing an $O(mn)$ time preprocess on M. More specifically, for each row $M^{(i)}$ $(i = 1, 2, \ldots, m)$ of M, we compute the prefix sums $M_j^{(i)}$ of $M^{(i)}$ $(j = 1, 2, \ldots, n)$. After this preprocess, given a subfield chain IL, for a fixed i, all the $\eta_k^{(i)}$'s and $\xi_k^{(i)}$'s can be computed in only $O(d)$ time. Hence, the ILP transformation takes $O(md + md^2) = O(md^2)$ time.

Hence in this way, in our CFSO algorithm, every CFSFO problem instance takes only $O(md^2 + d^3)$ time to solve. Note that it still takes $O(d^3)$ time for solving the ILP. Thus, we have the following result for the CFSFO problem.

Theorem 3. *Given an IM M of size $m \times n$, a maximum allowed field width $w > 0$, and a threshold beam-on time $T_{th} > 0$, the CFSO problem on M, w, and T_{th} can be solved in $O(mn + (md^2 + d^3)\Delta^{d-2})$ time, where $d = \lceil n/w \rceil \geq 1$ and $\Delta = w\lceil n/w \rceil - n + 1$.*

We should mention that in the current clinical settings, the value of d is normally not bigger than 3. Thus our above CFSO algorithm is an optimal $O(mn)$ time solution for most of the current practical clinical applications.

3 Implementation and Experiments

To examine the performance of our new CFSO algorithm, we implement the algorithm using the C programming language on Linux systems. We experiment with it using the following kinds of data: (1) 118 IMs of various sizes for 17 clinical cancer cases obtained from the Department of Radiation Oncology, the University of Maryland School of Medicine, and (2) 500 randomly generated IMs. The widths of the tested IMs range from 15 to 40, and the maximum intensity level of each IM is normalized to 100. The maximum allowed subfield widths (i.e., the maximum leaf spreads of the MLC) we consider are 14 and 17.

Figures 3(a)-3(d) five some of the field splitting results using our CFSO algorithm on four IMs for two clinical cancer cases and two randomly generated IMs, respectively. A distinct feature of our CFSO algorithm, as shown by Figure 3, is that it can compute a trade-off between the threshold MBT value and the total MBT of the resulting subfields. That is, we can find a field splitting which minimizes the total MBT subject to a given MBT upper bound on each resulting individual subfield, and *vice versa*. This trade-off feature is in fact between the treatment efficiency and treatment quality and can be useful clinically.

We also compare our CFSO algorithm with the following previously known field splitting software/algorithms:

1. CORVUS 5.0, which is a current most popular commercial radiation treatment planning software (developed by the NOMOS Corporation).
2. The field splitting algorithm by Kamath *et al.* [11] and Wu [17] (denoted by FSSL), which splits using vertical lines.

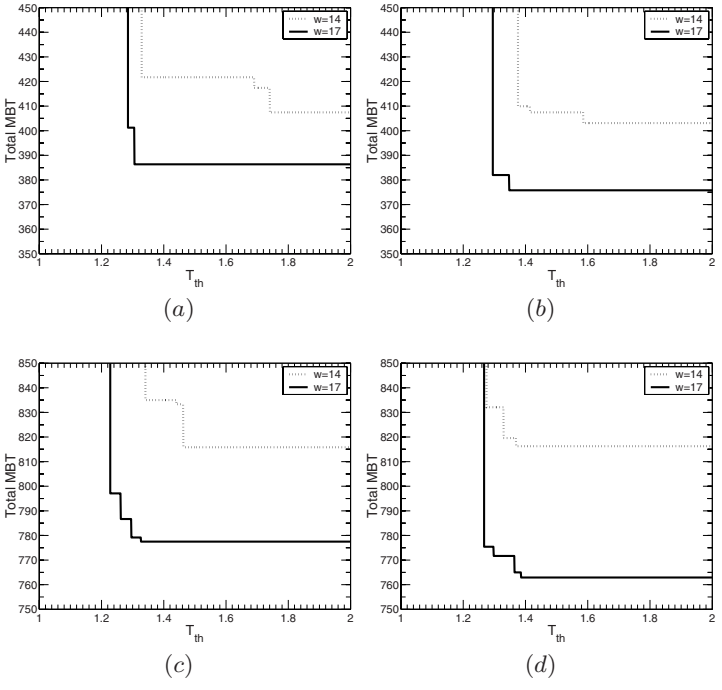


Fig. 3. The trade-off curves between the threshold MBT value T_{th} and the total MBT of the resulting subfields on four tested IMs. (a)-(b) illustrate the results on one head-and-neck cancer case and one prostate cancer case, respectively. (c)-(d) illustrate the results on two randomly generated IMs. In each figure, the solid and dotted lines show the trade-off curves for $w = 14$ and $w = 17$, respectively, where w is the maximum allowed subfield width. The threshold MBT T_{th} shown in these figures is a scaled value (i.e., the original threshold value T_{th} divided by a base value, where the base value is set to be the ideal MBT of the overall input IM calculated by Equation (1) divided by the number d of the resulting subfields).

3. The field splitting algorithm by Chen and Wang [5] (denoted by FSMP), which splits using y-monotone paths.
4. The field splitting algorithm by Chen and Wang [5] (denoted by FSO), which splits with overlapping.

More specifically, we consider the total MBT and the maximum subfield MBT of the output subfields of all these splitting approaches. For the CFSO algorithm, we let its maximum subfield MBT value of the resulting subfields be the lowest threshold MBT value T_{th} such that the total MBT corresponding to that threshold MBT value T_{th} on the same IM computed by the CFSO algorithm is the same as the total MBT achieved by the FSO algorithm. That is, when both the CFSO and FSO algorithms produce exactly the same total MBTs, we compare their maximum subfield MBT values of the resulting subfields (or compare the maximum subfield MBT values of the FSO algorithm with the threshold MBT values of our CFSO algorithm).

Table 1. Comparison results of the total MBT and the maximum subfield MBT on (a) 118 clinical IMs and (b) 500 randomly generated IMs. The maximum leaf spreads we use are 14 and 17 (the 1st column). The data sets are grouped based on the numbers of the resulting subfields from the splittings (the 2nd column). The total MBT and maximum subfield MBT shown are the average values computed for each data sets using the field splitting algorithms under comparison. The data under subcolumns FSCV are results produced by CORVUS 5.0. Each threshold MBT value T_{th} for the CFSO algorithm is chosen to be the smallest possible maximum subfield MBT value such that the total MBT corresponding to that maximum subfield MBT value (computed by the CFSO algorithm) is the same as the total MBT achieved by the FSO algorithm.

w	d	Total MBT				Maximum Subfield MBT				
		FSCV	FSSL	FSMP	FSO CFSO	FSCV	FSSL	FSMP	FSO	CFSO
14	2	353.4	322.2	301.8	288.2	192.9	252.8	252.4	181.8	148.8
	3	492.8	419.5	397.8	371.9	197.5	210.9	208.4	167.6	138.6
17	2	400.1	385.0	376.0	359.4	215.9	319.1	319.4	228.6	179.8
	3	492.8	451.1	419.3	398.4	197.5	220.0	211.0	157.6	148.9

(a)

w	d	Total MBT				Maximum Subfield MBT				
		FSCV	FSSL	FSMP	FSO CFSO	FSCV	FSSL	FSMP	FSO	CFSO
14	2	650.5	619.7	593.6	576.1	340.8	321.5	389.8	385.0	294.9
	3	995.5	939.3	875.4	839.9	355.3	405.1	401.0	325.6	315.4
17	2	738.8	706.3	676.8	661.0	385.4	449.0	446.5	380.2	337.7
	3	1064	989.3	901.5	847.3	378.8	457.7	462.1	345.8	316.1

(b)

Table 1 compares the average total MBTs of these algorithms for splitting the IMs in the data sets into two or three subfields. For all our data sets, the total MBTs of the four previous methods, CORVUS 5.0 (denoted by FSCV), FSSL, FSMP, and FSO, are in decreasing order. As shown in Table 1, the maximum subfield MBTs of the CFSO algorithm are uniformly smaller than the other four splitting methods for all the data sets, which demonstrates that our CFSO algorithm can minimize the total MBT while effectively controlling the maximum subfield MBT. In terms of the maximum subfield MBT values, our CFSO algorithm shows an average improvement of 13.6% over the FSO algorithm.

Note that when splitting an IM of size $m \times n$ into two or three subfields (i.e., $d = 2, 3$), our CFSO algorithm runs in an optimal $O(mn)$ time. Experimentally, the total execution time of the CFSO algorithm on all the 618 IMs that we use is only 35.77 seconds (on average, 0.0579 second per IM) on an HP xw6400 Workstation with a 2 GHz Intel Core 2 Duo processor and 2 GB memory running Redhat Enterprise Linux 4.

References

1. Ahuja, R.K., Hamacher, H.W.: A Network Flow Algorithm to Minimize Beam-on Time for Unconstrained Multileaf Collimator Problems in Cancer Radiation Therapy. Networks 45, 36–41 (2005)
2. Boland, N., Hamacher, H.W., Lenzen, F.: Minimizing Beam-on Time in Cancer Radiation Treatment Using Multileaf Collimators. Networks 43(4), 226–240 (2004)
3. Boyer, A.L.: Use of MLC for Intensity Modulation. Med. Phys. 21, 1007 (1994)
4. Chen, D.Z., Healy, M.A., Wang, C., Wu, X.: A New Field Splitting Algorithm for Intensity-Modulated Radiation Therapy. In: Proc. of 13th Annual International Computing and Combinatorics Conference, pp. 4–15 (2007)
5. Chen, D.Z., Wang, C.: Field Splitting Problems in Intensity-Modulated Radiation Therapy. In: Proc. 12th Annual Int. Symp. on Algorithms and Computation, pp. 690–700 (2006)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. McGraw-Hill, New York (2001)
7. Dogan, N., Leybovich, L.B., Sethi, A., Emami, B.: Automatic Feathering of Split Fields for Step-and-Shoot Intensity Modulated Radiation Therapy. Phys. Med. Biol. 48, 1133–1140 (2003)
8. Engel, K.: A New Algorithm for Optimal Multileaf Collimator Field Segmentation. Discrete Applied Mathematics 152(1-3), 35–51 (2005)
9. Hong, L., Kaled, A., Chui, C., Losasso, T., Hunt, M., Spirou, S., Yang, J., Amols, H., Ling, C., Fuks, Z., Leibel, S.: IMRT of Large Fields: Whole-Abdomen Irradiation. Int. J. Radiat. Oncol. Biol. Phys. 54, 278–289 (2002)
10. Kamath, S., Sahni, S., Li, J., Palta, J., Ranka, S.: A Generalized Field Splitting Algorithm for Optimal IMRT Delivery Efficiency. In: The 47th Annual Meeting and Technical Exhibition of the American Association of Physicists in Medicine (AAPM) (2005) Also, Med. Phys. 32(6), 1890 (2005)
11. Kamath, S., Sahni, S., Ranka, S., Li, J., Palta, J.: Optimal Field Splitting for Large Intensity-Modulated Fields. Med. Phys. 31(12), 3314–3323 (2004)
12. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. John Wiley, Chichester (1988)
13. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, New Jersey (1982)
14. Webb, S.: The Physics of Three-Dimensional Radiation Therapy. Institute of Physics Publishing, Bristol (1993)
15. Webb, S.: The Physics of Conformal Radiotherapy — Advances in Technology. Institute of Physics Publishing, Bristol (1997)
16. Wu, Q., Arnfield, M., Tong, S., Wu, Y., Mohan, R.: Dynamic Splitting of Large Intensity-Modulated Fields. Phys. Med. Biol. 45, 1731–1740 (2000)
17. Wu, X.: Efficient Algorithms for Intensity Map Splitting Problems in Radiation Therapy. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 504–513. Springer, Heidelberg (2005)
18. Wu, X., Dou, X., Bayouth, J.E., Buatti, J.M.: New Algorithm for Field Splitting in Radiation Therapy. In: Proc. 13th Annual Int. Symp. on Algorithms and Computation, pp. 692–703 (2007)

A Practical Parameterized Algorithm for Weighted Minimum Letter Flips Model of the Individual Haplotyping Problem*

Minzhu Xie[1,2], Jianxin Wang[1,**], Wei Zhou[1], and Jianer Chen[1,3]

[1] School of Information Science and Engineering,
Central South University, Changsha 410083, P.R. China
jxwang@mail.csu.edu.cn
[2] College of Physics and Information Science,
Hunan Normal University, Changsha 410081, P.R. China
[3] Department of Computer Science,
Texas A&M University, College Station, TX 77843, USA
http://netlab.csu.edu.cn

Abstract. Given a set of DNA sequence fragments of an individual with each base of every fragment attached a confidence value, the weighted minimum letter flips model (WMLF) of the individual haplotyping problem is to infer a pair of haplotypes by flipping a number of bases such that the sum of the confidence values corresponding to the flipped bases is minimized. WMLF is NP-hard. This paper proposes a parameterized exact algorithm for WMLF of time $O(nk_2 2^{k_2} + mlogm + mk_1)$, where m is the number of fragments, n is the number of SNP sites, k_1 is the maximum number of SNP sites that a fragment covers, and k_2 is the maximum number of fragments that cover a SNP site. Since in real biological experiments, both k_1 and k_2 are small, the parameterized algorithm is efficient in practical application.

1 Introduction

A *single nucleotide polymorphism* (SNP) is a single base mutation of a DNA sequence that occurs in at least 1% of the population. SNPs are the predominant form of human genetic variation and more than 3 million SNPs are distributed throughout the human genome [1, 2]. Detection of SNPs helps identifying biomedically important genes for diagnosis and therapy, is used in identification of individual and descendant, and can also be used in the analysis of genetic relations of populations.

* This research was supported in part by the National Natural Science Foundation of China under Grant Nos. 60433020 and 60773111, the National Basic Research 973 Program of China No.2008CB317107, the Program for New Century Excellent Talents in University No. NCET-05-0683, the Program for Changjiang Scholars and Innovative Research Team in University No. IRT0661, and the Scientific Research Fund of Hunan Provincial Education Department under Grant No.06C526.

** Corresponding author.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 16–27, 2008.

In humans and other diploid organisms, chromosomes are paired up. A *haplotype* describes the SNP sequence of a chromosome, while a *genotype* describes the conflated data of the SNP sequences on a pair of chromosomes. In Fig. 1, the haplotypes of the individual are (A, T, A, C, G) and (G, C, A, T, G), and the genotype is (A/G, C/T, A/A, C/T, G/G).

A SNP site where both haplotypes have the same nucleotide is called a *homozygous* site and a SNP site where the haplotypes are different is called a *heterozygous* site. To reduce the complexity, a haplotype can be represented as a string over a two-letter alphabet {0, 1} rather than the four-letter alphabet{A, C, G, T} , where '0' denotes the major allele and '1' denotes the minor. In Fig. 1, the haplotypes can be represented by "01011" and "10001".

A . . T . . A . . C . . G .

G . . C . . A . . T . . G .

Fig. 1. SNPs

Haplotyping, i.e., identification of chromosome haplotypes, plays an important role in SNP applications [3]. Stephens *et al.* [4] identified 3899 SNPs that were present within 313 genes from 82 unrelated individuals of diverse ancestry. Their analysis of the pattern of haplotype variation strongly supports the recent expansion of human populations. Based on linkage studies of SNPs and the association analysis between haplotypes and type 2 diabetes, Horikawa *et al.* [5] localized the gene *NIDDM1* to the distal long arm of chromosome 2 and found 3 SNPs in *CAPN10* associated with type 2 diabetes.

Haplotyping has been time-consuming and expensive using biological techniques. Therefore, effective computational techniques have been in demand for solving the haplotyping problem. A number of combinatorial versions of the haplotyping problem have been proposed, and they generally fall into two classes: the individual haplotyping and the population haplotyping, i.e., haplotyping based on SNP fragments or genotype samples [6].

The current paper is focused on the *Weighted Minimum Letter Flips* (WMLF) model of the individual haplptyping problem, which is NP-hard [7]. The paper is organized as follows: Section 2 introduces the individual haplotyping problem; in Section 3, we propose a practical parameterized algorithm for WMLF; and in Sections 4 and 5, experiment results and coclusion are presented.

2 Individual Haplotyping Problem

The individual haplotyping problem was first introduced by Lancia et al. [8]: Given a set of aligned SNP fragments from the two copies of a chromosome, infer two haplotypes. The aligned SNP fragments come from DNA shotgun sequencing or other sequencing experiments. The individual haplotyping problem aims to partition the set of SNP fragments into two subset with each subset determining a haplotype.

The aligned SNP fragments of an individual can be represented as an $m \times n$ SNP matrix M over the alphabet $\{0, 1, -\}$, in which n columns represent a sequence of SNPs according to the order of sites in a chromosome and m rows represent m fragments. Fig. 2 shows a 7×6 SNP matrix. In the matrix M,

the ith row's value at the jth column is denoted by $M_{i,j}$, which equals the ith fragment's value at the jth SNP site. If the value of the ith fragment at the jth SNP site misses (i.e. there is a hole in the fragment) or the ith fragment doesn't cover the jth SNP site, then $M_{i,j}$ takes the value "−" (the value "−" will be called the *empty value*).

The following are some definitions related to the SNP matrix M.

We say that the ith row *covers* the jth column if there are two indices k and r such that $k \leq j \leq r$, and both $M_{i,k}$ and $M_{i,r}$ are not empty. In other words, the ith row covers the jth column if either $M_{i,j}$ is not empty or there are a column on the left and another column on the right such that the ith row takes non-empty values at these columns.

SNPs

Fragments						
	–	–	–	–	1	0
	–	0	1	–	–	0
	0	1	1	0	–	–
	1	0	1	–	0	1
	–	1	0	–	–	–
	–	–	0	1	–	–
	–	–	–	0	1	0

Fig. 2. SNP Matrix

The set of (ordered) rows covering the jth column is denoted by $rowset(j)$. The first and the last column that the ith row covers are denoted by $left(i)$ and $right(i)$ respectively.

For example, as to the SNP matrix in Fig. 2, row 2 covers columns 2, 3, 4, 5 and 6, and $rowset(5)=\{1, 2, 4, 7\}$.

If $M_{i,j} \neq \text{“}-\text{”}, M_{k,j} \neq \text{“}-\text{”}$ and $M_{i,j} \neq M_{k,j}$, then the ith row and the kth row of M are said to *conflict* at the column j. If the ith and kth rows of M do not conflict at any column then they are *compatible*.

A SNP matrix M is *feasible* if its rows can be partitioned into two subsets such that the rows in each subset are all compatible.

Obviously, a SNP matrix M is feasible if and only if there are two haplotypes such that every row of M is compatible with one of the two haplotypes. And we claim that M can be *derived* from the two haplotypes.

Since a row of M comes from one of a pair of chromosomes, if there are no DNA sequencing errors, we can always derive M from the haplotypes of the pair of chromosomes. However, DNA sequencing errors are unavoidable and it is hard to decide which copy of chromosome a SNP fragment comes from, therefore the individual haplotyping problem is complex.

Based on different optimal criteria, there have been various computational models of the problem. There are some typical models [8, 9]: *Minimum fragment Removal* (MFR), *Minimum SNPs Removal* (MSR), *Minimum Error Correction* (MEC). Among the models above, MEC is considered to have most biological meaning. MEC is also called as *Mininum Letter Flips* (MLF) [10] and have been extended by including different extra information.

Since a DNA sequencer can provide one confidence value for each base [7], the confidence values corresponding an $m \times n$ SNP matrix M can organized as an $m \times n$ weight matrix W. The element of W at row i and column j, the confidence value of $M_{i,j}$, is denoted by $W_{i,j}$. If $M_{i,j}$='-', $W_{i,j}$=0. By including a weighted matrix, Greenberg et al. [10] introduced the *weighted minimum letter flips* model, and Zhao et al. [7] formulated it as follows:

Weighted Minimum Letter Flips (WMLF): Given a SNP matrix M a weighted matrix W, flip a number of elements ('0' into '1' and vice versa) of M so that the resulting matrix is feasible and the sum of confidence values of the elements in W corresponding the flipped elements in M is minimized.

Let S be a element subset of M. If after flipping the elements in S, M is feasible, S is a *feasible flipped element subset* of M. The sum of confidence values of the elements in S is defined as the *flipping cost* of S, and is denoted by $C(S)$, i. e. $C(S) = sum_{M_{i,j} \in S} W_{i,j}$.

Among all possible feasible flipped element subset of M, if S minimizes $C(S)$, we call $C(S)$ a solution to WMLF. For briefness, given a SNP matrix M and a corresponding weight matrix W, we denote a solution to WMLF by WMLF(M, W).

For the WMLF model, Zhao et al. [7] have proved it to be NP-hard, and designed a dynamic clustering algorithm. However, to the best of our knowledge, there has been no practical exact algorithm for it. In the following section, we will propose a practical parameterized exact algorithm for WMLF.

3 A Parameterized Algorithm for WMLF

By carefully studying related properties of fragment data, we have found the following fact. In all sequencing centers, due to technical limits, the sequencing instruments such as ABI 3730 and MageBACE can only sequence DNA fragments whose maximum length is about 1000 nucleotide bases. Since the average distribution density of SNPs is about 1 SNP per 1kb bases, the maximum number of SNP sites that a fragment covers is small, and usually smaller than 10 according to the current research results [3, 11].

Moreover, in DNA sequencing experiments, the fragment coverage is also small. In Celera's whole-genome shotgun assembly of the human genome, the fragment average coverage is 5.11 [1], and in the human genome project of the International Human Genome Sequencing Consortium, the fragment average coverage is 4.5 [12]. Huson *et al.* [13] have analyzed the fragment data of the human genome project of Celera's, and given a fragment coverage plot. Although the fragment covering rate is not the same at all sites along the whole genome, the plot shows that most sites are covered by 5 fragments, and that the maximum of fragments covering a site is no more than 19. Therefore, for an SNP site, compared with the total number of fragments, the number of fragments covering the SNP site is very small.

Based on the observations above, we introduce the following parameterized condition.

Definition 1. The (k_1, k_2) *parameterized condition*: the number of SNP sites covered by a single fragment is bounded by k_1, and each SNP site is covered by no more than k_2 fragments.

Accordingly, in a SNP matrix satisfying the (k_1, k_2) parameterized condition, each row covers at most k_1 columns and each column is covered by at most k_2 rows.

For an $m \times n$ SNP matrix M, the parameters k_1 and k_2 can be obtained by scanning all rows of M. In the worst case, $k_1 = n$ and $k_2 = m$. But as to the fragment data of Celera's human genome project, k_2 is no more than 19 [13].

Before describing our parameterized algorithm, there are some definitions.

For a feasible flipped element subset S of a SNP matrix M, after flipping the elements in S, all the rows of M can be partitioned into two classes H_0 and H_1, such that every two rows in the same class are compatible.

Definition 2. Let R be a subset of rows in a SNP matrix M. A *partition function* P on R maps each row in R to one of the values $\{0, 1\}$.

Suppose that R contains $h > 0$ rows, a partition function P on R can be denoted by an h-digit binary number in $\{0, 1\}$, where the i-th digit is the value of P on the ith row in R. If $R = \varnothing$, we also define a unique partition function P, which is denoted by -1.

For briefness, a partition function defined on $rowset(j)$ is called a partition function at column j. For a SNP matrix M satisfying the (k_1, k_2) parameterized condition, there are at most 2^{k_2} different partition functions at column j.

Let R be a set of rows of the matrix M, and P be a partition function on R. For a subset R' of R, the partition function P' on R' obtained by restricting P on the subset R' is called the *projection* of P on R', and P is called an *extension* of P' on R.

For briefness, let $M[:, j]$ be the SNP matrix consisting of the first j columns of M.

Definition 3. Fix a j. Let P be a partition function on a row set R. Defined $V_E[P, j]$ to be any subset S of elements of $M[:, j]$ that satisfies the following condition: After flipping the elements of S, there is a partition (H_0, H_1) of all rows in M such that any two rows in the same class do not conflict at any column from 1 to j, and for any row $i \in R$, row i is in the class H_q if and only if $P(i) = q$, for $q \in \{0, 1\}$.

Given a partition function P at column j, the rows covering column j can be partitioned into (H_0, H_1) by P according to the following rule: for each row $i \in rowset(j)$, $i \in H_q$ if $P(i) = q$. In order to make the rows in the same class don't conflict at column j, the values at column j of some rows have to be flipped to avoid confliction. For $q \in \{0, 1\}$, let $v_q = 0$ or 1. For any row $i \in H_q$, if $M_{i,j} = v_q$, $M_{i,j}$ is to be flipped. In consequence, we obtain a flipped elements set *Flips*, and $C(Flips) = \sum_{q=0..1} w(P, j, q, v_q)$, where $w(P, j, q, v_q) = \sum_{i:i \in rowset(j), P(i)=q, M_{i,j}=v_q} (W_{i,j})$.

Let $Minor(P, j, 0)$ and $Minor(P, j, 1)$ denote the value of v_0 and the value of v_1 minimizing $w(P, j, 0, v_0) + w(P, j, 1, v_1)$, respectively. Let $Flips(P, j)$ denote the flipped elements set $\{M_{i,j} \mid M_{i,j} = Minor(P, j, P(i))\}$.

Fig. 3 gives a function $CompFlipsW(j, P, Flips, C)$ to compute $Flips(P, j)$ and $C(Flips(P, j))$, whose time complexity is $O(k_2)$ for a SNP matrix M satisfying the (k_1, k_2) parameterized condition.

```
CompFlipsW(j, P, Flips, C)
//Flips denotes Flips(P, j), and C denotes C(Flips(P, j))
{   for q, v = 0, 1 do w_{q,v} = 0;   //w_{q,v} denotes w(P, j, q, v_q)
    tmp= P; //tmp is a binary number
    for each row i in rowset(j) (according to the order) do
    {   q = the least significant bit of tmp;   //q = P(i)
        tmp right shift 1 bit; v = M_{i,j};
        if v ≠ − then w_{q,v} = w_{q,v} + W_{i,j}; }
    v_0 = v_1 = 1;   //v_q denotes Minor(P, j, q) for q = 0, 1
    if w_{0,0} + w_{1,0} < w_{0,v_0} + w_{1,v_1} then
    {   v_0 = v_1 = 0;   }
    if w_{0,0} + w_{1,1} < w_{0,v_0} + w_{1,v_1} then
    {   v_0 = 0;   v_1 = 1;   }
    if w_{0,1} + w_{1,0} < w_{0,v_0} + w_{1,v_1} then
    {   v_0 = 1;   v_1 = 0;   }
    tmp= P;   Flips = ∅;   C = 0;
    for each row i in rowset(j) (according to the order) do
    {   q = the least significant bit of tmp;
        tmp right shift 1 bit;
        if v_q = M_{i,j} then { C = C + W_{i,j}; Flips = Flips ∪ M_{i,j}; } } }
```

Fig. 3. *CompFlipsW*

Fix a $m \times n$ SNP matrix M, a corresponding weight matrix W, and a partition function P on the set R of all rows in M. Let the projection on $rowset(j)$ of P be P^j. It is easy to verify that the following theorem is true.

Theorem 1. Fix l $(1 \leq l \leq n)$. $C(V_E[P, l])$ is minimized if and only if $V_E[P, l] = \cup_{j=1..l}(Flips(P^l, l))$.

Therefore, we have the following equation.

$$\text{WMLF}(M, W) = min_{P:\text{a partition function on }} {}_R C(\cup_{j=1..n}(Flips(P^j, j))),$$

which means a algorithm of time complexity $O(nk_2 2^m)$. To reduce the time complexity, we consider a partition function P at column j.

Definition 4. Fix a j. Let P be a partition function at column j. Defined $S_E[P, j]$ to be any $V_E[P, j]$ that minimizes the flipping cost $C(V_E[P, j])$ of $V_E[P, j]$. And $E[P, j]$ is defined to be $C(S_E[P, j])$.

Given an $m \times n$ SNP matrix M and a corresponding weight matrix W, from Definitions 3 and 4, it is easy to verify that the following equation holds true:

$$\text{WMLF}(M, W) = \min_{P:\, P \text{ is a partition function at column } n} (E[P, n]) \qquad (1)$$

For a partition function P at column 1, according to Theorem 1, the following equations hold true.

$$S_E[P,1] = \mathit{Flips}(P,1) \tag{2}$$

$$E[P,1] = C(\mathit{Flips}(P,1)) \tag{3}$$

In order to present our algorithm, we need to extend the above concepts from one column to two columns as follows. Let the set of all rows that cover both columns j_1 and j_2 be $R_c(j_1,j_2)$.

Definition 5. Fix a j. Let P' be a partition function on $R_c(j,j+1)$. Defined $S_B[P',j]$ to be any $V_E[P',j]$ that minimizes the flipping cost $C(V_E[P',j])$ of $V_E[P',j]$. And $B[P',j]$ is defined to be $C(S_B[P',j])$.

Given a j and a partition function P' on $R_c(j,j+1)$. If $E[P,j]$ and $S_E[P,j]$ are known for each extension P of P' on $\mathit{rowset}(j)$, $B[P',j]$ and $S_B[P',j]$ can be calculated by the following equations:

$$B[P',j] = \min_{P\text{: } P \text{ is an extension of } P' \text{ on } \mathit{rowset}(j)} (E[P,j]) \tag{4}$$

$$S_B[P',j] = S_E[P,j] \mid P \text{ minimizes } E[P,j] \tag{5}$$

Inversely, for any partition function P on $\mathit{rowset}(j)$, because $R_c(j-1,j)$ is a subset of $\mathit{rowset}(j)$, the project P' of P on $\mathit{rowset}(j)$ is unique. When $S_B[P',j-1]$ and $B[P',j-1]$ are known, according to Theorem 1, $E[P,j]$ and $S_E[P,j]$ can be calculated by the following equations:

$$S_E[P,j] = S_B[P',j-1] \cup \mathit{Flips}(P,j) \tag{6}$$

$$E[P,j] = B[P',j-1] + C(\mathit{Flips}(P,j)) \tag{7}$$

Based on the equations above, WMLF(M,W) can be obtained as follows: firstly, $S_E[P,1]$ and $E[P,1]$ can be obtained according to Equations (2) and (3) for all partition functions P at column 1; secondly, $B[P',1]$ and $S_B[P',1]$ can be obtained by using Equations (4) and (5) for all partition functions P' on $R_c(1,2)$; thirdly, $S_E[P,2]$ and $E[P,2]$ can be obtained by using Equations (6) and (7) for all partition functions P on $\mathit{rowset}(2)$; and so on, at last $E[P,n]$ and $S_E[P,n]$ can be obtained for all partition functions P at column n. Once $E[P,n]$ and $S_E[P,n]$ for all possible P are known, a solution to the WMLF problem can be obtained by using Equation (1). Please see Fig. 4 for the details of our P-WMLF algorithm.

Theorem 2. If M satisfies the (k_1,k_2) parameterized condition, the P-WMLF algorithm solves the WMLF problem in time $O(nk_2 2^{k_2} + mlogm + mk_1)$ and space $O(mk_1 2^{k_2} + nk_2)$.

Algorithm P-WMLF
input: an $m \times n$ SNP matrix M, an $m \times n$ weight matrix W
output: a solution to WMLF
1. **initiation**: sort the rows in M in ascending order such that for any two rows i_1 and i_2, if $i_1 < i_2$, then $left(i_1) \leq left(i_2)$; for each column l, calculate an ordered set $rowset(l)$ and the number $H[l]$ of the rows that cover column l; $j = 1$;
2. **for** $P = 0..2^{H[j]} - 1$ **do** // partition function is coded by a binary number
2.1. $CompFlipsW(j, P, Flips, C)$;
//$E[P]$ and $S_E[P]$ denote $E[P, j]$ and $S_E[P, j]$, respectively.
2.2. $E[P] = C$; $S_E[P] = Flips$; //Eqs. (2), (3),
3. **while** $j < n$ **do** //recursion based on Eqs. (4)-(7)
// MAX denotes the maximal integer.
3.1. calculate N_c, the number of rows that cover both columns j and $j + 1$, and a vector **Bits** such that **Bits**[i]=1 denotes the ith row of $rowset(j)$ covers column $j + 1$;
3.2. **for** $P' = 0..2^{N_c} - 1$ **do** $B[P']$=MAX;
3.3. **for** $P = 0..2^{H[j]} - 1$ **do**
3.3.1. calculate the project P' of P on $R_c(j, j + 1)$ using **Bits**.
3.3.2. **if** $B[P'] > E[P]$ **then**
$B[P'] = E[P]$, $S_B[P'] = S_E[P]$; //Eqs. (4), (5)
3.4. $j++$; //next column
3.5. **for** $P' = 0..2^{N_c} - 1$ **do**
3.5.1. **for** each extensions P of P' on $rowset(j)$ **do**
3.5.1.1. $CompFlipsW(j, P, Flips, C)$;
3.5.1.2. $E[P] = C + B[P']$; $S_E[P] = S_B[P'] \cup Flips$; //Eqs. (6), (7)
//Eq. (1)
4. output the minimal $E[P]$ and the corresponding $S_E[P]$ ($P = 0..2^{H[n]} - 1$).

Fig. 4. P-WMLF Algorithm

Proof. Given an $m \times n$ SNP matrix M satisfying the (k_1, k_2) parameterized condition, consider the following storage structure: each row keeps the first and the last column that the row covers, i.e. its left and right value, and its values at the columns from its left column to its right column. In such a storage structure, M takes space $O(mk_1)$. It is easy to see that *rowset* takes space $O(nk_2)$, H takes space $O(n)$, E and B take space $O(2^{k_2})$, and S_E and S_B take space $O(mk_1 2^{k_2})$. In summary, the space complexity of the algorithm is $O(mk_1 2^{k_2} + nk_2)$.

Now we discuss the time complexity of the algorithm. In Step 1, sorting takes time $O(mlogm)$. All *rowsets* can be obtained by scanning the rows only once, which takes time $O(mk_1)$. For any column j, because that no more than k_2 row covers covering it, $H[j] \leq k_2$, the function $CompFlipsW$ takes time $O(k_2)$, and Step 2 takes time $O(k_2 2^{k_2})$. In Step 3.1, scanning $rowset(j)$ and $rowset(j + 1)$ simultaneously can obtain N_c and **Bits**, and takes time $O(k_2)$. Step 3.2 takes time $O(2^{k_2})$, and Step 3.3 takes time $O(k_2 2^{k_2})$. In Step 3.5, for each P', there are $2^{H[j]-N_C}$ extensions of P' on $rowset(j)$. Given P', an extension of P' can be

obtained by a bit-or operation in time $O(1)$, because after the sorting in Step 1, the rows that cover column j, but do not cover column $j-1$ are all behind the rows in $R_c(j-1,j)$. In all, Step 3.5 takes time $O(k_2 2^{N_c} 2^{k_2-N_c})$. Then Step 3 is iterated $n-1$ times and takes time $O(nk_2 2^{k_2})$. Step 4 takes time $O(2^{k_2})$. In summary, the time complexity of the algorithm is $O(nk_2 2^{k_2} + mlogm + mk_1)$. This completes the proof. □

4 Experimental Results

In the experiments, we compare the running time and the reconstruction rate of haplotypes of P-WMLF and Zhao *et al.*'s dynamic clustering algorithm DC-WMLF [7]. The reconstruction rate of haplotypes is defined as the ratio of the number of the SNP sites that correctly inferred out by an algorithm to the total number of the SNP sites of the haplotypes [7].

The haplotype data can be obtained by two methods [7]: the first is to get real haplotypes from public domain, and the second is to generate simulated haplotypes by computer.In our exeperiments, the real haplotypes was obtained from the file genotypes_chr1_CEU_r21_nr_fwd_phased.gz [1] , which was issued in July 2006 by the International HapMap Project [2]. The file contains 120 haplotypes on chromosome 1 of 60 individuals of the CEU with each haplotype containing 193333 SNP sites. From the 60 individuals, select a individual at random. Then begining with a random SNP site, a pair of haplotypes of a given length can be obtained from the haplotypes of the selected individual.

The simulated haplotypes can be generated as follows ([14, 15]). At first a haplotype h_1 of length n is generated at random, then another haplotype h_2 of the same length is generated by flipping every char of h_1 with the probability of d.

As to framgent data, to the best of our knowledge, real DNA fragments data in the public domain are not available, and references [7] and [15] used computer-generated simulated fragment data. After obtaining a pair of real or simulated haplotypes, in order to make the generated fragments have the same statistical features as the real data, a widely used shotgun assembly simulator Celsim ([16]) is invoked to generate m fragments whose lengths are between $lMin$ and $lMax$. At last the output fragments are processed to plant reading errors with probability e and empty values with probability p. Please refer to [15] and [16] for the details about how to generate artificial fragment data.

In our experiments, the parameters are set as follows: the probability d used in generating simulated haplotypes is 20%, fragment coverage rate $c = 10$, the minimal length of fragment $lMin = 3$, the maximal length of fragment $lMax = 7$ and empty value probability $p = 2\%$.

The weight matrix W corresponding to the fragments is generated by the method of [7]: the entries of W are normally distributed with mean μ and variance $\sigma^2 = 0.05$. For a correct SNP site, $\mu = 0.9$, and for an error SNP site,$\mu = 0.8$.

[1] From http://www.hapmap.org/downloads/phasing/2006-07_phaseII/phased/

P-WMLF and DC-WMLF are implemented in C++. We ran our experiments on a Linux server (4 Intel Xeon 3.6GHz CPU and 4GByte RAM) with the length of haplotype n, the number of fragment m ($m = 2 \times n \times c/(lMax + lMin)$) and the reading error probability e varied. The data of Table 1 and Fig. 5 are the average over 100 repeated experiments with the same parameters.

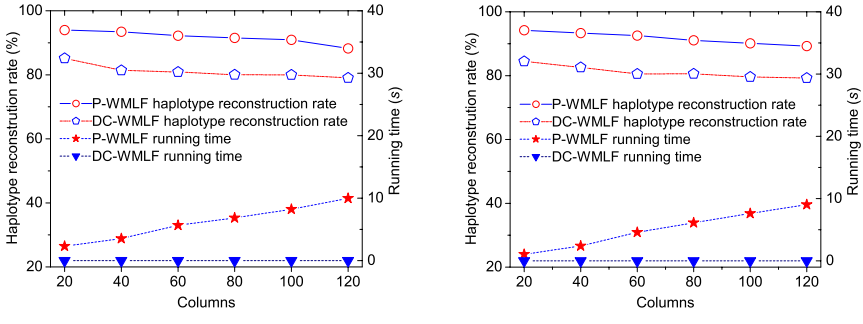
When $n = 100$, the experiment results on both real haplotype data and simulated haplotype data show that when e increases, the haplotype reconstruction rates of the algorithms decrease, and though DC-WMLF is faster than P-WMLF, P-WMLF is more accurate in haplotype reconstruction rate than DC-WMLF. The results is given in Table 1.

Table 1. Comparison of performances of P-WMLF and DC-WMLF with e varied

	Haplotype reconstruction rate (%)		Average running time (s)	
e (%)	P-WMLF	DC-WMLF	P-WMLF	DC-WMLF
1	93.35 (93.25)	80.16 (80.30)	6.48 (5.90)	0.006 (0.006)
3	91.32 (91.32)	80.11 (80.05)	7.22 (6.16)	0.006 (0.006)
5	90.03 (90.60)	79.78 (79.89)	8.22 (7.68)	0.006 (0.006)
7	88.22 (88.97)	79.42 (79.39)	9.26 (7.96)	0.006 (0.006)

The data not enclosed in brackets are the experiment results coming from the experiments on the real haplotype data, and the data in brackets show the ones on the simulated haplotypes data. In the experiments, $n = 100$ and $m = 200$.

With $e = 5\%$, when n increases, the experiment results show again that P-WMLF is more accurate in haplotype reconstruction rate than DC-WMLF, and that though P-WMLF is slower than DC-WMLF, P-WMLF can work out with an exact solution to WMLF in 10 seconds when $n = 120$. The results is illustrated by Fig. 5.



(a) On the real haplotype data (b) On the simulated haplotype data

Fig. 5. The performance comparison of P-WMLF and DC-WMLF when n increases

5 Conclusion

Haplotyping plays a more and more important role in some regions of genetics such as locating of genes, designing of drugs and forensic applications. WMLF is an important computational model of inferring the haplotypes of an individual from one's aligned SNP fragment data with confidence values. Because reading errors cannot be avoid in the DNA sequencing process, WMLF is a NP-hard problem. To solve the problem, Zhao *et al.* [7] proposed a dynamic clustering algorithm DC-WMLF. Being a heuristic algorithm, it cannot to ensure the accuracy in haplotype reconstruction. Based on the fact that the maximum number of fragments covering a SNP site is small (usually no more than 19 [13]), the current paper introduced a new practical parameterized exact algorithm P-WMLF to solve the problem. With the fragments of maximum length k_1 and the maximum number k_2 of fragments covering a SNP site, the P-WMLF algorithm can solve the problem in time $O(nk_2 2^{k_2} + mlogm + mk_1)$ and in space $O(mk_1 2^{k_2} + nk_2)$. Extensive experiments show that P-WMLF has higher haplotype reconstruction rate than DC-WMLF, and it is practical even if it is slower than DC-WMLF.

References

[1] Venter, J.C., Adams, M.D., Myers, E.W., et al.: The sequence of the human genome. Science 291(5507), 1304–1351 (2001)
[2] The International HapMap Consortium: A haplotype map of the human genome. Nature 437(7063), 1299–1320 (2005)
[3] Gabriel, S.B., Schaffner, S.F., Nguyen, H., et al.: The structure of haplotype blocks in the human genome. Science 296(5576), 2225–2229 (2002)
[4] Stephens, J.C., Schneider, J.A., Tanguay, D.A., et al.: Haplotype variation and linkage disequilibrium in 313 human genes. Science 293(5529), 489–493 (2001)
[5] Horikawa, Y., Oda, N., Cox, N.J., et al.: Genetic variation in the gene encoding calpain-10 is associated with type 2 diabetes mellitus. Nature Genetics 26(2), 163–175 (2000)
[6] Zhang, X.S., Wang, R.S., Wu, L.Y., Chen, L.: Models and algorithms for haplotyping problem. Current Bioinformatics 1(1), 105–114 (2006)
[7] Zhao, Y.Y., Wu, L.Y., Zhang, J.H., Wang, R.S., Zhang, X.S.: Haplotype assembly from aligned weighted snp fragments. Computational Biology and Chemistry 29(4), 281–287 (2005)
[8] Lancia, G., Bafna, V., Istrail, S., Lippert, R., Schwartz, R.: SNPs Problems, Complexity, and Algorithms. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 182–193. Springer, Berlin (2001)
[9] Lippert, R., Schwartz, R., Lancia, G., Istrail, S.: Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. Brief. Bioinform 3(1), 1–9 (2002)
[10] Greenberg, H.J., Hart, W.E., Lancia, G.: Opportunities for combinatorial optimization in computational biology. INFORMS J. Comput. 16(3), 211–231 (2004)
[11] Hinds, D.A., Stuve, L.L., Nilsen, G.B., Halperin, E., Eskin, E., Ballinger, D.B., Frazer, K.A., Cox, D.R.: Whole-genome patterns of common dna variation in three human populations. Science 307(5712), 1072–1079 (2005)

[12] International Human Genome Sequencing Consortium: Initial sequencing and analysis of the human genome. Nature 409(6822), 860–921 (2001)
[13] Huson, D.H., Halpern, A.L., Lai, Z., Myers, E.W., Reinert, K., Sutton, G.G.: Comparing Assemblies Using Fragments and Mate-Pairs. In: Gascuel, O., Moret, B.M.E. (eds.) WABI 2001. LNCS, vol. 2149, pp. 294–306. Springer, Berlin (2001)
[14] Wang, R.S., Wu, L.Y., Li, Z.P., Zhang, X.S.: Haplotype reconstruction from snp fragments by minimum error correction. Bioinformatics 21(10), 2456–2462 (2005)
[15] Panconesi, A., Sozio, M.: Fast Hare: A Fast Heuristic for Single Individual SNP Haplotype Reconstruction. In: Jonassen, I., Kim, J. (eds.) WABI 2004. LNCS (LNBI), vol. 3240, pp. 266–277. Springer, Heidelberg (2004)
[16] Myers, G.: A dataset generator for whole genome shotgun sequencing. In: Lengauer, T., Schneider, R., Bork, P., Brutlag, D.L., Glasgow, J.I., Mewes, H.W., Zimmer, R. (eds.) Proc. ISMB, pp. 202–210. AAAI Press, California (1999)

SlopeMiner: An Improved Method for Mining Subtle Signals in Time Course Microarray Data

Kevin McCormick, Roli Shrivastava, and Li Liao*

Computer and Information Sciences, University of Delaware
Newark, DE 19716, USA
lliao@cis.udel.edu

Abstract. This paper presents an improved method, SlopeMiner, for analyzing time course microarray data by identifying genes that undergo gradual transitions in expression level. The algorithm calculates the slope for the slow transition between the expression levels of data, matching the sequence of expression level for each gene against temporal patterns having one transition between two expression levels. The method, when used along with StepMiner - an existing method for extracting binary signals, significantly increases the annotation accuracy.

Keywords: Data mining, Time Course, DNA Microarray, Regression.

1 Introduction

DNA microarray is a high throughput technology that can measure the expression levels for a large number of genes simultaneously, and therefore has become a very useful tool in studying gene regulation [1, 2]. Analyzing the expression data, such as identifying co-regulated genes and further inferring regulatory networks inside the cell, has received a lot of attention in the bioinformatics community [3-9]. In order to enable more sophisticated analyses and to better interpret the results coming out of these analyses, it is critical to have a sensitive and accurate way to extract signals, as the gene expression data are typically noisy and prone to misclassification. For example, a basic question to ask is whether a gene is up- or down-regulated under different conditions and/or at different time points. The answer may not be always clear cut, as indicated in Fig. 1. Recently, a statistical method, called StepMiner, has been developed to cope with this issue in order to assist biologists in understanding the temporal progression of genetic events and biological processes following a stimulus, based on gene expression microarray data [10]. At the most basic level, StepMiner identifies genes which undergo one or more binary transitions over short time courses. It directly addresses one of the more basic questions one can ask of time course data: 'Which genes are up- or down-regulated as a result of the stimulus?' and 'When does the gene transition to up- or down-regulated?' StepMiner utilizes a procedure called the adaptive regression where every possible placement of the transition between time points is evaluated and the algorithm chooses the best fit [11].

* Corresponding author.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 28 – 34, 2008.

Although StepMiner is shown to be capable of identifying one- and two-step binary signals, there are limitations to its applications. Firstly, StepMiner is limited to identifying step changes, i.e., the transitions between up and down regulations are instantaneous. While this may be a reasonable assumption when the expression data are collected sparsely, i.e., at large time intervals, the actual transitions of expression level occur in a finite amount of time, which is determined by many factors including the kinetics of the processes that are involved in expression regulation. As shown in the right panel of Fig. 1, the transition from the down regulation to the up regulation is gradual, and forcing a sharp turn at a midpoint can be misleading about when the transition really begins to occur. Secondly, StepMiner requires high signal-noise ratio to perform accurately. In Ref [10], it is reported that the minimum height of step has to be 3 times the noise in order to achieve 95% accuracy. Besides, the method cannot handle very short or long time courses effectively.

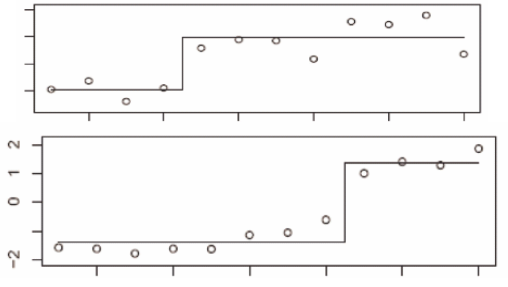


Fig. 1. Examples of extracting binary signal (solid line) of up/down regulation from noisy time course microarray data (dots)

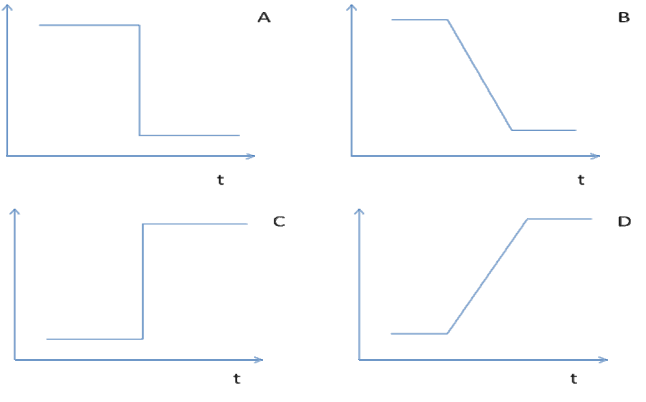


Fig. 2. Schematic illustration of expression patterns in microarray data. Panels A and C show sharp transition for down and up regulation respectively, whereas Panels B and D show the slow transition in the two respective cases.

In this paper we try to develop a more refined method to mitigate some of these problems. Specifically we focused on cases when the transition is a slow one, i.e., a sharp turning point does not exist, as illustrated in panels B and D of Fig. 2. As shown in the following sections, our new method not only can more accurately identify the gradual transitions but also is more sensitive in picking up weak signals by requiring lower minimum height to noise ratios. The paper is organized as following. In section 2, we described our refined method in detail. In section 3, we designed experiments to test the performance of the new methods on a set of simulated data, and made side by side comparisons between StepMiner and SlopeMiner on the same data sets. Discussions and conclusions are given in section 4.

2 Method

The main objective of this paper is to develop a new method that is able to overcome some of the limitations of StepMiner so that more subtle signals can be extracted from the time-course microarray data. Specifically, we focus on capturing the slow transition of expression levels between the up and down regulation, and we call this new method SlopeMiner.

The algorithm and the principle used are similar to those in StepMiner. Given a set of n time points and values X_1, X_2, X_3, …, X_n, we have to fit a one-slope (analogous to one-step) function to those time points. Suppose the fitted values are X_1^{fit}, X_2^{fit} X_3^{fit}, …, X_n^{fit}. Using these values we calculate the square error (SSE). The algorithm computes SSE with linear regression for all combinations of slope positions fitting the time course data. The various statistical parameters remain defined as the same as in Ref [10]. The total sum of squares is defined as

$$SSTOT = \sum_{i=1}^{n} (X_i - \overline{X})^2 \tag{1}$$

where $\overline{X}$ is the mean of data at n original time points. The sum of squares error SSE is defined as

$$SSE = \sum_{i=1}^{n} (X_i - X_i^{fit})^2 . \tag{2}$$

And the regression sum of squares SSR is given as

$$SSR = \sum_{i=1}^{n} (X_i^{fit} - \overline{X})^2 = SSTOT - SSE . \tag{3}$$

Let m be the degrees of freedom of one-slope function, the regression mean square is then given as

$$\text{MSE} = \text{SSE}/(n-m), \tag{4}$$

and the error mean square MSE is given as

$$MSR = SSR/(m-1). \quad (5)$$

We used the same degree 3 as suggested in Sahoo *et al*'s work [10]. Regression test statistic is computed as, F = MSR/MSE. If $F_{n-m}^{m-1} \leq F$, then the fit is considered to be, statistically speaking, a good fit. To compute the F-distribution a P-value of 0.05 was adopted.

The algorithm was implemented with the following four steps.

1. Compute the SSE statistic for all combinations of slope between the time points.
2. For each slope position compute the SSE statistic.
3. The slope position that results in the minimum SSE is then used to compute the F statistic.
4. If $F_{n-m}^{m-1} \leq F$ then we say that the SlopeMiner has successfully fitted the data to one-slope function and has found a match.

3 Results

The algorithm was evaluated on simulated time course microarray data with 20 time points. The simulated data has been generated using a subroutine in MATLAB. Gaussian noise with a 0 mean has also been added with a height to noise ratio varied between 1 and 5. 100 sets of data were generated with a fixed height to noise ratio and a fixed starting and ending point of slope. To test the ability of identifying gradual transitions with varied slopes, 9 combinations of slope positions were used: the distance between the start point and end point of slope was varied from 4 to 20. The 9 combinations are listed in Table 1. Without loss of generality, all the simulated data was one-slope with UP-transition. A heat map for the simulated data is shown in Figure 3 for visualization, where the low values are coded with green and the high values with red. The data has been written in appropriate formats for analysis by both StepMiner and SlopeMiner.

In our experiments, the height-to-noise ratio was varied from 1 to 5. The proportion of correctly identified datasets was plotted as function of the height-to-noise ratio as shown Fig. 4. For comparison, the results for both StepMiner and SlopeMiner are shown. It can be seen clearly that the SlopeMiner outperforms StepMiner when the height-to-noise ratio is less than 3. For height-to-noise ratio greater than 3 both the tools classify more than 99% of data correctly. Furthermore, SlopeMiner can classify the data correctly with more than 90% accuracy even when the height to noise ratio is equal to 2.

As mentioned in Section 1, another limitation of StepMiner was that the method works best when the constant segments are of at least length 3 even for height-to-noise ratios of 3 or higher. To test if this is improved with SlopeMiner we analyzed the proportion of correctly identified data sets as a function of distance between start and end points of slope. The results are shown in Fig. 5. We see that for height-to-noise ratio $\geq$ 3, SlopeMiner can correctly identify slopes even when the distance

Table 1. Combinational schemes for the simulated expression data

Starting point	Ending point
1	20
2	19
3	18
4	17
5	16
6	15
7	14
8	13
9	12

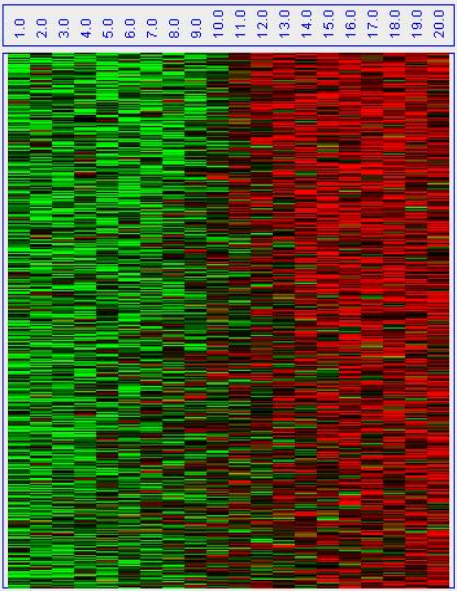


Fig. 3. Heat map for the simulated expression data. Green color for down regulated level and red color for up regulated level.

between start and end points = 18, i.e. when there is barely any constant segment. As the height/noise is decreased to 2 a constant part of length 1 is required. And for height/noise = 1 a constant segment of at least length 5 is needed. This is a significant improvement over StepMiner because now a transition can be identified even when the slope is very slow as it spans a large time interval. Therefore, with SlopeMiner, the limitation of StepMiner for requiring a "sharp-transition" has been mitigated in most general cases that are practically possible.

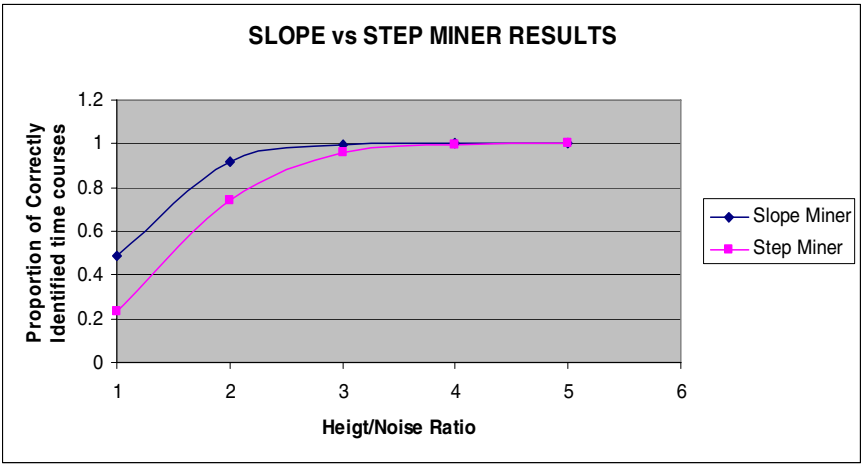


Fig. 4. Comparison of Performance between StepMiner and SlopeMiner

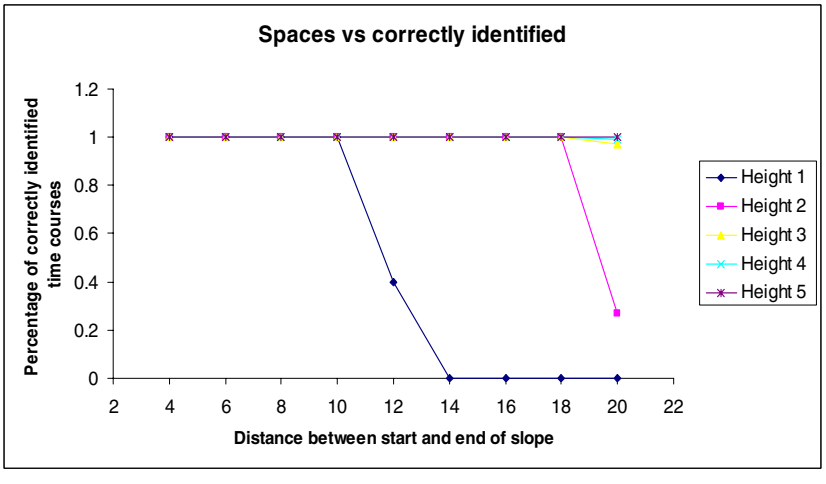


Fig. 5. Performance as affected by the height-to-noise ratio and slope

4 Conclusions

In this paper, we presented an improved method for identifying patterns in gene expression data with slow gradual transitions between down and up regulations. Using the adaptive regression technique, the method achieve 90% accuracy for noisy data (height-to-ratio of 2), which is a significant improvement from the 75% accuracy by StepMiner. In addition, the method is sensitive to slow transition data, with a minimum requirement of the constant segments. However, the current implementation does not handle multi-step transitions, and is limited to two level regulations. To eliminate these constraints should be the future work.

Acknowledgements. The authors are grateful to Debashish Sahoo for making available the StepMiner package.

References

1. Schena, M., Shalon, D., Davis, R.W., Brown, P.O.: Quantitative monitoring of gene expression patterns with a complementary DNA microarray. Science 270, 467–470 (1995)
2. Lashkari, D.A., DeRisi, J.L., McCusker, J.H., Namath, A.F., Gentile, C., Hwang, S.Y., Brown, P.O.: Yeast microarray for genome wide parallel genetic and gene expression analysis. PNAS 94, 13057–13062 (1997)
3. Aach, J., Church, G.M.: Aligning gene expression time series with time warping algorithms. Bioinformatics 17, 495–508 (2001)
4. Amato, R., Ciaramella, A., Deniskina, N., Del Mondo, C., di Bernardo, D., Donalek, C., Longo, G., Miele, G., et al.: A multi-step approach to time series analysis and gene expression clustering. Bioinformatics 22, 589–596 (2006)
5. Bar-Joseph, Z.: Analyzing time series gene expression data. Bioinformatics 20, 2493–2503 (2004)
6. Eisen, M.B., Spellman, P.T., Brown, P.O., Bostein, D.: Cluster analysis and display of genome-wide expression patterns. PNAS 95, 14863–14868 (1998)
7. De Jong, H.: Modeling and simulation of genetic regulatory systems: a literature review. J. Comput. Biol. 9, 67–103 (2002)
8. Martin, S., Zhang, Z., Martino, A., Faulon, J.-L.: Boolean dynamics of genetic regulatory networks inferred from microarray time series data. Bioinformatics 23, 866–874 (2007)
9. Luan, Y., Li., H.: Clustering of time-course gene expression data using a mixed-effects model with B-splines. Bioinformatics 19, 474–482 (2003)
10. Sahoo, D., Dill, D.L., Tibshirani, R., Plevritis, S.K.: Extracting binary signals from microarray time-course data. Nucleic Acids Research 35, 3705–3712 (2007)
11. Owen, A.: Discussion: Multivariate adaptive regression splines. Ann. Stat. 19, 102–112 (1991)

A PTAS for the k-Consensus Structures Problem Under Euclidean Squared Distance

Shuai Cheng Li, Yen Kaow Ng, and Louxin Zhang

[1] David R. Cheriton School of Computer Science, University of Waterloo, Waterloo ON N2L 3G1 Canada
scli@cs.uwaterloo.ca
[2] Department of Computer Science and Communication Engineering, Kyushu University, Fukuoka 819-0395, Japan
kalngyk@tcslab.csce.kyushu-u.ac.jp
[3] Department of Mathematics, National University of Singapore, Singapore 117543
matzlx@nus.edu.sg

Abstract. In this paper we consider a basic clustering problem that has uses in bioinformatics. A *structural fragment* is a sequence of ℓ points in a 3D space, where ℓ is a fixed natural number. Two structural fragments f_1 and f_2 are equivalent iff $f_1 = f_2 \cdot R + \tau$ under some rotation R and translation τ. We consider the *distance* between two structural fragments to be the sum of the Euclidean squared distance between all corresponding points of the structural fragments. Given a set of n structural fragments, we consider the problem of finding k (or fewer) structural fragments $g_1, g_2, \ldots, g_k$, so as to minimize the sum of the distances between each of $f_1, f_2, \ldots, f_n$ to its nearest structural fragment in $g_1, \ldots, g_k$. In this paper we show a PTAS for the problem through a simple sampling strategy.

1 Introduction

In this paper we consider the problem of clustering similar sequences of 3D points. Two such sequences are considered the same if they are equivalent under some rotation and translation. The scenerio which we consider is as follows. Suppose there is an original sequence of points that gave rise to a few variations of itself, through slight changes in some or all of its points. Now given these variations of the sequence, we are to reconstruct the original sequence. A likely candidate for such an original sequence would be a sequence which is "nearest" in terms of some distance measure, to the variations.

Now consider a more complicated scenerio where there had been not only one, but k original sequences of the same length. Given a set of variations of these original sequences, we are to find the original sequences. We formulate the problem as follows. Given n sequences of points $f_1, f_2, \ldots, f_n$, we are to find a set of k sequences $g_1, \ldots, g_k$, such that the sum of distances

$$\sum_{1 \le i \le n} \min_{1 \le j \le k} dist(f_i, g_j)$$

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 35–44, 2008.

is minimized. In this paper we consider the case where *dist* is the sum of Euclidean squared distances between each of the points in the two sequences f_i and g_k. The "square" in the distance measure is necessary to the method given in this paper (more on this in Discussions). On the other hand, it should be easy to adapt the method to other distance measures that fulfill this condition.

Such a problem has potential use in clustering protein structures. A protein structure is typically given as a sequence of points in 3D space, and for various reasons, there are typically minor variations in their measured structures. Our problem can be used in the case where we have a set of measurements of a few protein structures and are to reconstruct the original structures.

In this paper, we show that there is a polynomial-time approximation scheme (PTAS) for the problem, through a sampling strategy.

2 Preliminaries

Throughout this paper we let ℓ be a fixed non-zero natural number. A *structural fragment* is a sequence of ℓ 3D-points. The *mean square distance (MS)* between two structural fragments $f = (f[1], \ldots, f[\ell])$ and $g = (g[1], \ldots, g[\ell])$, is defined to be

$$MS(f, g) = \min_{R \in \mathcal{R}, \tau \in \mathcal{T}} \sum_{i=1}^{\ell} \parallel f[i] - (R \cdot g[i] + \tau) \parallel^2$$

where $\mathcal{R}$ is the set of all rotation matrices, $\mathcal{T}$ the set of all translation vectors, and $\parallel x - y \parallel$ is the Euclidean distance between $x, y \in \mathbb{R}^3$.

The root of the MS measure, $RMS(f, g) = \sqrt{MS(f, g)}$ is a measure that has been extensively studied. Note that $R \in \mathcal{R}$, $\tau \in \mathcal{T}$ that minimize $\sum_{i=1}^{\ell} \parallel f[i] - (R \cdot g[i] + \tau) \parallel^2$ to give us $MS(f, g)$ will also give us $RMS(f, g)$, and vice versa. Since given any f and g, there are closed form equations [1,5] for finding R and τ that give $RMS(f, g)$, $MS(f, g)$ can be computed efficiently for any f and g.

Furthermore, it is known that to minimize $\sum_{i=1}^{\ell} \parallel f[i] - (R \cdot g[i] + \tau) \parallel^2$, the centroid of f and g must coincide [1]. Due to this, without loss of generality we assume that all structural fragments have centroids at the origin. Such transformations can be done in time $O(n\ell)$. After such transformations, in computing $MS(f, g)$, only the parameter $R \in \mathcal{R}$ need to be considered, that is,

$$MS(f, g) = \min_{R \in \mathcal{R}} \sum_{i=1}^{\ell} \parallel f[i] - R \cdot g[i] \parallel^2$$

Suppose that given a set of n structural fragments $f_1, f_2, \ldots, f_n$, we are to find k structural fragments $g_1, \ldots, g_k$, such that each structural fragment f_i is "near", in terms of the MS, to at least one of the structural fragments in $g_1, \ldots, g_k$. We formulate such a problem as follows:

k-Consensus Structural Fragments Problem Under MS	
Input:	n structural fragments $f_1, \ldots f_n$, and a non-zero natural number $k < n$.
Output:	k structural fragments $g_1, \ldots g_k$, minimizing the cost $\sum_{i=1}^{n} \min_{1 \le j \le k} MS(f_i, g_j)$.

In this paper we will demonstrate that there is a PTAS for the problem.

We use the following notations: Cardinality of a set A is written $|A|$. For a set A and non-zero natural number n, A^n denotes the set of all length n sequences of elements of A. Let elements in a set A be indexed, say $A = \{f_1, f_2, \ldots, f_n\}$, then $A^{m!}$ denotes the set of all the length m sequences $f_{i_1}, f_{i_2}, \ldots, f_{i_m}$, where $1 \le i_1 \le i_2 \le \ldots \le i_m \le n$. For a sequence S, $S(i)$ denotes the i-th element in S, and $|S|$ denotes its length.

3 PTAS for the k-Consensus Structural Fragments

The following lemma, from [4], is central to the method.

Lemma 1 ([4]). *Let $a_1, a_2, \ldots, a_n$ be a sequence of real numbers and let $r \in N$, $1 \le r \le n$. Then the following equation holds:*

$$\frac{1}{n^r} \sum_{1 \le i_1, i_2, \ldots, i_r \le n} \sum_{i=1}^{n} \left(\frac{a_{i_1} + a_{i_2} + \cdots + a_{i_r}}{r} - a_i\right)^2 = \frac{r+1}{r} \sum_{i=1}^{n} \left(\frac{a_1 + a_2 + \cdots + a_n}{n} - a_i\right)^2$$

□

Let $P_1 = (x_1, y_1, z_1), P_2 = (x_2, y_2, z_2), \ldots, P_n = (x_n, y_n, z_n)$ be a sequence of 3D points.

$$\begin{aligned}
&\frac{1}{n^r} \sum_{1 \le i_1, i_2, \ldots, i_r \le n} \sum_{i=1}^{n} \| \frac{P_{i_1} + P_{i_2} + \cdots + P_{i_r}}{r} - P_i \|^2 \\
&= \frac{1}{n^r} \sum_{1 \le i_1, \ldots, i_r \le n} \sum_{i=1}^{n} \left(\frac{x_{i_1} + \ldots + x_{i_r}}{r} - x_i\right)^2 + \left(\frac{y_{i_1} + \ldots + y_{i_r}}{r} - y_i\right)^2 + \left(\frac{z_{i_1} + \ldots + z_{i_r}}{r} - z_i\right)^2 \\
&= \frac{r+1}{r} \sum_{i=1}^{n} \left(\frac{x_1 + \ldots + x_n}{n} - x_i\right)^2 + \left(\frac{y_1 + \ldots + z_n}{n} - z_i\right)^2 + \left(\frac{z_1 + \ldots + z_n}{n} - z_i\right)^2 \\
&= \frac{r+1}{r} \sum_{i=1}^{n} \| \frac{P_1 + P_2 + \cdots + P_n}{n} - P_i \|^2
\end{aligned}$$

One can similarly extend the equation for structural fragments. Let $f_1, \ldots, f_n$ be n structural fragments, the equation becomes:

$$\frac{1}{n^r} \sum_{1 \le i_1, \ldots, i_r \le n} \sum_{i=1}^{n} \| \frac{f_{i_1} + \cdots + f_{i_r}}{r} - f_i \|^2 = \frac{r+1}{r} \sum_{i=1}^{n} \| \frac{f_1 + \cdots + f_n}{n} - f_i \|^2 \quad (1)$$

The equation says that there exists a sequence or r structural fragments f_{i_1}, $f_{i_2}, \ldots, f_{i_r}$ such that

$$\sum_{i=1}^{n} \| \frac{f_{i_1} + \cdots + f_{i_r}}{r} - f_i \|^2 \leq \frac{r+1}{r} \sum_{i=1}^{n} \| \frac{f_1 + \cdots + f_n}{n} - f_i \|^2$$

Our strategy uses this fact —in essentially the same way as in [4]— to approximate the optimal solution for the k-consensus structural fragments problem. That is, by exhaustively sampling every combination of k sequences, each of r elements from the space $\mathcal{R}' \times \{f_1, \ldots, f_n\}$, where $f_1, \ldots, f_n$ is the input and $\mathcal{R}'$ is a fixed selected set of rotations, which we next discuss.

3.1 Discretized Rotation Space

Any rotation can be represented by a normalized vector u and a rotation angle θ, where u is the axis around which an object is rotated by θ. If we apply (u, θ) to a vector v, we obtain vector $\hat{v}$, which is:

$$\hat{v} = u(v \cdot u) + (v - w(v \cdot w)) \cos\theta + (v \times w) \sin\theta$$

where $\cdot$ represents dot product, and $\times$ represent cross product.

By the equation, one can verify that a change of ϵ in u will result in a change of at most $\alpha_1 \epsilon |v|$ in $|\hat{v}|$ for some computable $\alpha_1 \in \mathbb{R}$; and a change of ϵ in θ will result in a change of at most $\alpha_2 \epsilon |v|$ in $|\hat{v}|$ for some computable $\alpha_2 \in \mathbb{R}$. Now any rotation along an axis through the origin can be written in the form $(\theta_1, \theta_2, \theta_3)$, where $\theta_1, \theta_2, \theta_3 \in [0, 2\pi]$ are respectively a rotation along each of the x, y, z axes. Similarly, changes of ϵ in θ_1, θ_2 and θ_3 will result in a change of at most $\alpha\epsilon|v|$, for some computable $\alpha \in \mathbb{R}$.

We discretize the values that each θ_i, $1 \leq i \leq 3$ may take within the range $[0, 2\pi]$ into a series of angles of angular difference ϑ. There are hence at most $O(1/\vartheta)$ of such values for each θ_i, $1 \leq i \leq 3$. Let $\mathcal{R}'$ denote the set of all possible discretized rotations $(\theta_1, \theta_2, \theta_3)$. Note that $|\mathcal{R}'|$ is of order $O(1/\vartheta^3)$.

Let $\boldsymbol{d}$ be the diameter of a ball that is able to encapsulate each of $f_1, f_2, \ldots, f_n$. Hence any distance between two points among $f_1, \ldots, f_n$ is at most $\boldsymbol{d}$. In this paper we assume $\boldsymbol{d}$ to be constant with respect to the input size. Note that for a protein structure, $\boldsymbol{d}$ is of order $O(\ell)$ [3]. For any $b \in \mathbb{R}$, we can choose ϑ so small that for any rotation R and any point $p \in \mathbb{R}^3$, there exists $R' \in \mathcal{R}'$ such that $\| R \cdot p - R' \cdot p \| \leq \alpha\vartheta\boldsymbol{d} \leq b$.

3.2 A Polynomial Time Algorithm with Cost $((1+\epsilon)\boldsymbol{D_{opt}} + \boldsymbol{c})$

Consider the number of $F_1, F_2, \ldots, F_m$ in (2.1) that are possible. Let each F_j be represented by a length r string of $n+1$ symbols, n of which each represents one of $f_1, \ldots, f_n$, while the remaining symbol represents "nothing". It is clear that for any A_j, any $F_j \in A_j^{r!}$, or $F_j \in A_j^{|A_j|!}$ (where $|A_j| \leq r$), can be represented by one such string. Furthermore, any $F_1, F_2, \ldots, F_m$ can be completely represented

by k such strings — that is, to represent the case where $m < k$, $k - m$ strings can be set to "nothing" completely. From this, we can see that there are at most $(n+1)^{rk} = O(n^{rk})$ possible combinations of $F_1, F_2, \ldots, F_m$.

Approximation Algorithm k-Consensus Structural Fragments
Input: structural fragments $f_1, \ldots f_n$, natural numbers $k < n$ and $r \geq 1$.
Output: up to k structural fragments $g_1, \ldots g_k$.

(1) Let $D_{min} = \infty$, Consensus= $\emptyset$.
(2) For every possible set of $m \leq k$ disjoint sets $A_1, \ldots, A_m \subseteq \{f_1, \ldots, f_n\}$
(2.1) For every possible $F_1, F_2, \ldots, F_m$, where
$F_j \in A_j^{r!}$ if $|A_j| > r$, otherwise
F_j is the (unique) sequence in $A_j^{|A_j|!}$ that contains all the elements of A_j.
(2.2) For every possible sequence $\Theta_1, \Theta_2, \ldots, \Theta_m$, where
$\Theta_j \in \mathcal{R}'^{|F_j|}$ for $1 \leq j \leq m$.
(2.3) For $j = 1$ to m, find u_j, the average structural fragment
for $\Theta_j(1) \cdot F_j(1)$,
$\Theta_j(2) \cdot F_j(2)$,
$\vdots$
$\Theta_j(|F_j|) \cdot F_j(|F_j|)$.
(2.4) For $i = 1$ to n, find $d_i = \min\{\| u_j - R \cdot f_i \|^2 \mid 1 \leq j \leq m, R \in \mathcal{R}'\}$.
(2.5) If $\sum_{i=1}^{n} d_i < D_{min}$,
set D_{min} to $\sum_j d_j$ and Consensus to $\{u_1, \ldots, u_k\}$.
(3) Output Consensus.

For each of these combinations, there are $|\mathcal{R}'|^{rk}$ possible combinations of $\Theta_1, \Theta_2, \ldots, \Theta_m$ at (2.2), hence resulting in $O((n|\mathcal{R}'|)^{rk})$ iterations to run for (2.3) to (2.5). Since (2.3) can be done in $O(rk\ell)$, (2.4) in $O(nk|\mathcal{R}'|\ell)$, and (2.5) in $O(n)$ time, the algorithm completes in $O(k\ell(r + n|\mathcal{R}'|)(n|\mathcal{R}'|)^{rk})$ time.

We argue that D_{min} eventually is at most $(r+1)/r$ of the optimal solution plus a factor. Suppose the optimal solution results in the $m \leq k$ disjoint clusters $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_m \subseteq \{f_1, \ldots, f_n\}$.

For each $\mathbf{A}_j$, $1 \leq j \leq m$, let $\boldsymbol{u}_j$ be a structural fragment which minimizes $\sum_{f \in \mathbf{A}_j} MS(\boldsymbol{u}_j, f)$. Furthermore, for each $f \in \mathbf{A}_j$, let $\boldsymbol{R}_f$ be a rotation where

$$\boldsymbol{R}_f \in \arg\min_{R \in \mathcal{R}} \| \boldsymbol{u}_j - R \cdot f \|^2$$

and let

$$\mathbf{D}_j = \sum_{f \in \mathbf{A}_j} \| \boldsymbol{u}_j - \boldsymbol{R}_f \cdot f \|^2 \quad \text{(Hence the optimal cost, } \mathbf{D} = \textstyle\sum_{j=1}^{m} \mathbf{D}_j\text{.)}$$

By the property of the MS measure, it can be shown that $\boldsymbol{u}_j$ is the average of $\{\boldsymbol{R}_f \cdot f \mid f \in \mathbf{A}_j\}$. For each $\mathbf{A}_j$ where $|\mathbf{A}_j| > r$, by Equation 1,

$$\frac{1}{|\mathbf{A}_j|^r} \sum_{F_j \in \mathbf{A}_j^r} \sum_{f \in \mathbf{A}_j} \parallel \frac{\boldsymbol{R}_{F_j(1)} \cdot F_j(1) + \cdots + \boldsymbol{R}_{F_j(r)} \cdot F_j(r)}{r} - \boldsymbol{R}_f \cdot f \parallel^2 = \frac{r+1}{r} \mathbf{D}_j$$

For each such $\mathbf{A}_j$, let $\boldsymbol{F}_j \in \mathbf{A}_j^r$ be such that

$$\sum_{f \in \mathbf{A}_j} \parallel \frac{\boldsymbol{R}_{\boldsymbol{F}_j(1)} \cdot \boldsymbol{F}_j(1) + \cdots + \boldsymbol{R}_{\boldsymbol{F}_j(r)} \cdot \boldsymbol{F}_j(r)}{r} - \boldsymbol{R}_f \cdot f \parallel^2 \leq \frac{r+1}{r} \mathbf{D}_j$$

Without loss of generality assume that each $\boldsymbol{F}_j \in \mathbf{A}_j^{r!}$. Let

$$\boldsymbol{\mu}_j = \begin{cases} \frac{\boldsymbol{R}_{\boldsymbol{F}_j(1)} \cdot \boldsymbol{F}_j(1) + \cdots + \boldsymbol{R}_{\boldsymbol{F}_j(r)} \cdot \boldsymbol{F}_j(r)}{r} & \text{if } |\mathbf{A}_j| > r \\ \frac{\boldsymbol{R}_{\boldsymbol{F}_j(1)} \cdot \boldsymbol{F}_j(1) + \overset{r}{\cdots} + \boldsymbol{R}_{\boldsymbol{F}_j(|\mathbf{A}_j|)} \cdot \boldsymbol{F}_j(|\mathbf{A}_j|)}{|\mathbf{A}_j|} & \text{otherwise} \end{cases}$$

Then we may write,

$$\sum_{j=1}^{m} \sum_{f \in \mathbf{A}_j} \parallel \boldsymbol{\mu}_j - \boldsymbol{R}_f \cdot f \parallel^2 \leq \frac{r+1}{r} \mathbf{D} \tag{2}$$

For each rotation $\boldsymbol{R}_f$, let R_f be a closest rotation to $\boldsymbol{R}_f$ within $\mathcal{R}'$. Also, let

$$\mu_j = \begin{cases} \frac{R_{\boldsymbol{F}_j(1)} \cdot \boldsymbol{F}_j(1) + \cdots + R_{\boldsymbol{F}_j(r)} \cdot \boldsymbol{F}_j(r)}{r} & \text{if } |\mathbf{A}_j| > r \\ \frac{R_{\boldsymbol{F}_j(1)} \cdot \boldsymbol{F}_j(1) + \overset{r}{\cdots} + R_{\boldsymbol{F}_j(|\mathbf{A}_j|)} \cdot \boldsymbol{F}_j(|\mathbf{A}_j|)}{|\mathbf{A}_j|} & \text{otherwise} \end{cases}$$

Since we exhaustively sample all possible $F_j \in A_j^{r!}$ for all possible A_j and for all $R \in \mathcal{R}'$, it is clear that:

$$D_{min} \leq \sum_{j=1}^{m} \sum_{f \in \mathbf{A}_j} \parallel \mu_j - R_f \cdot f \parallel^2 \tag{3}$$

We will now relate the LHS of Equation 2 with the RHS of Equation 3. The RHS of Equation 3 is

$$\begin{aligned}
&\sum_{j=1}^{m} \sum_{f \in \mathbf{A}_j} \| \mu_j - R_f \cdot f \|^2 \\
=&\sum_{j=1}^{m} \sum_{f \in \mathbf{A}_j} \| \mu_j + (\boldsymbol{\mu}_j - \boldsymbol{\mu}_j) + (\boldsymbol{R}_f \cdot f - \boldsymbol{R}_f \cdot f) - R_f \cdot f \|^2 \\
\leq&\sum_{j=1}^{m} \sum_{f \in \mathbf{A}_j} (\| \boldsymbol{\mu}_j - \boldsymbol{R}_f \cdot f \| + (\| \mu_j - \boldsymbol{\mu}_j \| + \| \boldsymbol{R}_f \cdot f - R_f \cdot f \|))^2 \\
=&\sum_{j=1}^{m} \sum_{f \in \mathbf{A}_j} \| \boldsymbol{\mu}_j - \boldsymbol{R}_f \cdot f \|^2 + (\| \mu_j - \boldsymbol{\mu}_j \| + \| \boldsymbol{R}_f \cdot f - R_f \cdot f \|)^2 \\
&\qquad\qquad + 2 \| \boldsymbol{\mu}_j - \boldsymbol{R}_f \cdot f \| (\| \mu_j - \boldsymbol{\mu}_j \| + \| \boldsymbol{R}_f \cdot f - R_f \cdot f \|) \\
\leq&\sum_{j=1}^{m} \sum_{f \in \mathbf{A}_j} \| \boldsymbol{\mu}_j - \boldsymbol{R}_f \cdot f \|^2 + 8n\ell b
\end{aligned}$$

Hence by Equation 2, D_{min} is at most $(r+1)/r = 1+1/r$ of the optimal solution plus a factor $c = 8n\ell b$. Let $\epsilon = 1/r$,

Theorem 1. *For any $c, \epsilon \in \mathbb{R}$, a $((1+\epsilon)D_{opt} + c)$-approximation solution for the k-consensus structural fragments problem can be computed in*

$$O(k\ell(\frac{1}{\epsilon} + n|\mathcal{R}'|)(n|\mathcal{R}'|)^{\frac{k}{\epsilon}})$$

time.

The factor c in Theorem 1 is due to error introduced by the use of discretization in rotations. If we are able to estimate a lowerbound of D_{opt}, we can scale this error by refining the discretization such that c is an arbitrarily small factor of D_{opt}. To do so, in the next section we show a lowerbound to D_{opt}.

3.3 A Polynomial Time 4 Approximation Algorithm

We now show a 4-approximation algorithm for the k-consensus structural fragments problem. We first show the case for $k = 1$, and then generalizes the result to all $k \geq 2$.

Let the input n structural fragments be f_1, f_2, ..., f_n. Let f_a, $1 \leq a \leq n$ be the structural fragment where

$$\sum_{1 \leq j \leq n \wedge j \neq a} MS(f_a, f_j)$$

is minimized. Note that f_a can be found in time $O(n^2\ell)$, since for any $1 \leq i, j \leq n$, $MS(f_i, f_j)$ (more precisely, $RMS(f_i, f_j)$) can be computed in time $O(\ell)$ using closed form equations from [5].

We argue that f_a is a 4-approximation. Let the optimal structural fragment be f_{opt}, the correspoding distance D_{opt}, and let f_b $(1 \leq b \leq n)$ be the fragment where $MS(f_b, f_{opt})$ is minimized.

We first note that the cost of using f_a as solution, $\sum_{i \neq a} MS(f_a, f_i) \leq \sum_{i \neq b} MS(f_b, f_i)$. To continue we first establish the following claim.

Claim. $MS(f, f') \leq 2(MS(f, f'') + MS(f'', f'))$.

Proof. In [2], it is shown that

$$RMS(f, f') \leq RMS(f, f'') + RMS(f'', f')$$

Squaring both sides gives

$$MS(f, f') \leq MS(f, f'') + MS(f'', f') + 2RMS(f, f'')RMS(f'', f')$$

Since

$$2RMS(f, f'')RMS(f'', f') \leq MS(f, f'') + MS(f'', f')$$

we have $MS(f, f') \leq 2(MS(f, f'')+MS(f'', f'))$. ∎

By the above claim,

$$\begin{aligned}
\sum_{i \neq b} MS(f_b, f_i) &\leq 2 \sum_{i \neq b} (MS(f_b, f_{opt}) + MS(f_{opt}, f_i)) \\
&= 2 \sum_{i \neq b} MS(f_b, f_{opt}) + 2 \sum_{i \neq b} MS(f_i, f_{opt}) \\
&\leq 2 \sum_{i \neq b} MS(f_b, f_{opt}) + 2D_{opt} \\
&\leq 2 \sum_{j \neq b} MS(f_j, f_{opt}) + 2D_{opt} \\
&\leq 2D_{opt} + 2D_{opt} = 4D_{opt}
\end{aligned}$$

Hence $\sum_{i \neq a} MS(f_a, f_i) \leq 4D_{opt}$. We now extend this to k structural fragments.

4-Approximation Algorithm k-Consensus Structural Fragments
Input: structural fragments $S = \{f_1, \ldots f_n\}$, natural number $k < n$.
Output: up to k structural fragments A.

(1) For every set $A \subseteq S$ of up to k structural fragments, do
(2) Compute cost$(A) = \sum_{f \in S-A} \min_{f' \in A} MS(f, f')$
(3) Output A with the least cost(A).

We first pre-compute $MS(f, f')$ for every pair of $f, f' \in S$, which takes time $O(n^2\ell)$. Then, at step (1), there are at most $O(n^k)$ combinations of A, each which takes $O(nk)$ time to compute at step (2). Hence in overall, we can perform the computation within time of $O(n^2\ell + kn^{k+1})$. To see that the solution is

a 4-approximation, let $S_1, S_2, \ldots, S_m$ where $m \leq k$ be an optimal clustering. Then, by our earlier argument, there exists $f_{i_1} \in S_1$, $f_{i_2} \in S_2$, ..., $f_{i_m} \in S_m$ such that each f_{i_x} is a 4-approximation for S_x, and hence $(f_{i_1}, f_{i_2}, \ldots, f_{i_m})$ is a 4-approximation for the k-consensus structural fragments problem. Since the algorithm exhaustively search for every combination of up to k fragments, it gives a solution at least as good as $(f_{i_1}, f_{i_2} \ldots, f_{i_m})$, and hence is a 4-approximation algorithm.

Theorem 2. *A 4-approximation solution for the k-consensus structural fragments problem can be computed in time $O(n^2\ell + kn^{k+1})$.*

3.4 A $(1+\epsilon)$ Polynomial Time Approximation Scheme

Recall that the algorithm in Section 3.2 has cost $D \leq (1+\epsilon)D_{opt} + 8n\ell b$ where $b = \alpha\vartheta\boldsymbol{d}$. From Section 3.3 we have a lowerbound $\boldsymbol{D}_{opt}$ of D_{opt}. We want $8n\ell b \leq \epsilon \boldsymbol{D}_{opt} \leq \epsilon D_{opt}$. To do so, it suffices that we set $\vartheta \leq \epsilon \boldsymbol{D}_{opt}/(8n\ell\alpha\boldsymbol{d})$. This results in an $|\mathcal{R}'|$ of order $O(1/\vartheta^3) = O((n\ell\boldsymbol{d})^3)$. Substituting this in Theorem 1, and combining with Theorem 2, we get the following.

Theorem 3. *For any $\epsilon \in \mathbb{R}$, a $((1+\epsilon)D_{opt})$-approximation solution for the k-consensus structural fragments problem can be computed in*

$$O(n^2\ell + kn^{k+1} + k\ell(\frac{2}{\epsilon} + n\lambda)(n\lambda)^{\frac{2k}{\epsilon}})$$

time, where $\lambda = (n\ell\boldsymbol{d})^3$.

4 Discussions

The method in this paper depends on Lemma 1. For this reason, the technique does not extend to the problem under distance measures where Lemma 1 cannot be applied, for example, the RMS measure. However, should Lemma 1 apply to a distance measure, it should be easy to adapt the method here to solve the problem for that distance measure.

One can also formulate variations of the k-consensus structural fragments problem. For example,

k-Closest Structural Fragments Problem Under MS	
Input:	n structural fragments $f_1, \ldots f_n$, and a non-zero natural number $k < n$.
Output:	k structural fragments $g_1, \ldots g_k$, minimizing the threshold $\max_{1 \leq i \leq n} \min_{1 \leq j \leq k} MS(f_i, g_j)$.

The cost function of the k-consensus structural fragments problem has some resemblance to that of the k-means problem, while the cost function of the k-closest structural fragments resembles the (absolute) k-center problem.

References

1. Arun, K.S., Huang, T.S., Blostein, S.D.: Least-squares fitting of two 3-d point sets. IEEE Trans. Pattern Anal. Mach. Intell. 9(5), 698–700 (1987)
2. Boris, S.: A revised proof of the metric properties of optimally superimposed vector sets. Acta Crystallographica Section A 58(5), 506 (2002)
3. Hao, M., Rackovsky, S., Liwo, A., Pincus, M.R., Scheraga, H.A.: Effects of compact volume and chain stiffness on the conformations of native proteins 89, 6614–6618 (1992)
4. Qian, J., Li, S.C., Bu, D., Li, M., Xu, J.: Finding Compact Structural Motifs. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 142–149. Springer, Heidelberg (2007)
5. Umeyama, S.: Least-squares estimation of transformation parameters between two point patterns. IEEE Trans. Pattern Anal. Mach. Intell. 13(4), 376–380 (1991)

Haplotype Assembly from Weighted SNP Fragments and Related Genotype Information

Seung-Ho Kang, In-Seon Jeong, Mun-Ho Choi, and Hyeong-Seok Lim

Dept. of Computer Science, Chonnam National University,
Yongbong-dong 300, Buk-gu, Gwangju 500-757, Korea
kinston@gmail.com, isjung0@hotmail.com, howork@paran.com
hslim@chonnam.ac.kr

Abstract. The algorithms that are based on the Weighted Minimum Letter Flips (WMLF) model are more accurate in haplotype reconstruction than those based on the Minimum Letter Flips (MLF) model, but WMLF is effective only when the error rate in SNP fragments is low. In this paper, we first establish a new computational model that employs the related genotype information as an improvement of the WMLF model and show its NP-hardness, and then we propose an efficient genetic algorithm to solve for the haplotype assembly problem. The results of experiments on a real data set indicate that the introduction of genotype information to the WMLF model is quite effective in improving the reconstruction rate especially when the error rate in SNP fragments is high.

Keywords: haplotype assembly problem, WMLF model, genetic algorithm.

1 Introduction

With complete genome sequences for humans now available, the investigation of genetic differences is one of the main topics in genomics[9]. Complete sequencing confirms that we all share some 99% identity at the DNA level, so there are small regions of differences that are responsible for our diversities. It is known that these few regions of differences in DNA sequences are responsible for the genetic diseases[7]. The most abundant source of genetic variation in the human genome is represented by single nucleotide polymorphism (SNP). SNPs are believed to be the most frequent form of genetic variability, and its understanding will increase our ability to treat human diseases, design drugs, and create new medical applications.

A SNP is a variation of a single nucleotide at a fixed point of the DNA sequence and in a bounded range of possible values, and each variant is called an allele. In particular, each SNP shows a variability of only two different alleles. We denote each allele by 0 (wild type) and 1 (mutant type). The sequence of SNPs in a specific chromosome is called a haplotype. In diploid organisms such as humans, genomes are organized into pairs of chromosomes, so there are two copies of the haplotypes for each of the SNP sequences. A genotype is the conflation of two haplotypes on the homologous chromosomes. For a genotype, when a pair of alleles

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 45–54, 2008.

at a SNP site is made of two identical types, this site is called homozygous and denoted by 0 or 1; otherwise it is called heterozygous and denoted by 2. Haplotypes play a more important role than genotypes in disease association studies[6]. However, current sequencing techniques can detect the presence of SNP sites, but they cannot tell which copy of a pair of chromosomes the alleles belong to. Hence, it is more difficult to determine haplotypes than to determine genotypes. To help overcome this difficultly, two classes of problems are defined from the viewpoint of computation. One is the population haplotyping problem and the other is the individual haplotyping problem. The former, which is called the haplotype inference problem, is to infer a set of haplotypes from the genotype set of a population. The latter, which is called the haplotype assembly problem, is to obtain a pair of haplotypes by assembling the aligned SNP fragments, each of which can contain errors. This paper focuses on the haplotype assembly problem. The haplotype assembly problem is concerned with the partitioning of the aligned SNP fragments into two sets, with each set determining a haplotype. There are several models based on different error assumptions[10]. Among these models, the Minimum Letter Flips (MLF) model and its weighted version, the weighted MLF (WMLF) model are proposed based on the assumption that all SNP fragments are from one organism, and they involve some sequencing errors. The WMLF model has another assumption that for each SNP site, there is a weight for its flipping, which represents the confidence level of the sequencer machine's reading. Both models have been proven to be NP-hard even if the SNP fragment matrix is gapless[1,12]. Zhao et al. showed that the WMLF model has a higher accuracy for haplotype reconstruction than the MLF model[12]. However, both models are effective when the SNP fragments have a low error rate. In order to improve the quality of the MLF model, the idea that genotype information can be employed has been proposed[9]. Since genotype data can be much more easily obtained, it is a practical and important strategy. Several methods that use genotype information in the MLF model have been proposed[8,9,11], but there has been no method that employs the genotype information of an individual in the WMLF model, which is known to be more accurate. Here we propose a new WMLF/GI model that adds genotype information to the existing WMLF model and a genetic algorithm to solve the problem.

The paper is organized as follows. In Section 2, we give the problem definition and its hardness together with the mathematical model formulation. In Section 3, a genetic algorithm is designed to solve the WMLF model with genotype information. Finally, the experimental results and conclusions are shown in Sections 4 and 5, respectively.

2 Formulation and Problem Definition

In this section, we propose a new computational model as an extension of the WMLF model. The proposed model employs the genotype information to improve the accuracy and speed of reconstruction of a pair of haplotypes from sequenced SNP fragments.

Suppose that there are m SNP fragments obtained from the two copies of a chromosome and the length of each fragment is n, and each SNP can take either one of the two values 0 (wild type) or 1 (mutant type). These fragment data are represented by an $m \times n$ matrix M over {0, 1, -}, which called the SNP matrix. The symbol - is used to represent a SNP that is not covered by a fragment and we say that it is missing or call it a gap. Each row of the matrix corresponds to a SNP fragment (denoted by f_i) and each column corresponds to a SNP site of fragments. The sequencer machine attaches a confidence level to each sequenced value in these fragments, which represents the probability that the value is correctly read. The confidence levels of these vales can be represented by an $m \times n$ weight matrix W. The element of W, w_{ij}, denotes the confidence level of the value at the corresponding SNP site f_{ij} of matrix M, and if there is the symbol - at f_{ij}, then we assign it a 0. The confidence level (weight) of a value is assumed to be between 0 and 1. Since two different sites can have the same letters by flipping the letter with the smaller weight, the distance between the SNP sites of two fragments f_i and f_j is defined by their weights as follows:

$$d(f_{ik}, f_{jk}) = \begin{cases} min\{w_{ik}, w_{jk}\}, & \text{if } f_{ik} \neq -, f_{jk} \neq -, \text{ and } f_{ik} \neq f_{jk} \\ 0 \quad , & \text{otherwise} \end{cases} . \quad (1)$$

If one site is of a haplotype, then their distance is the weight of the fragment site i.e. $d(f_{ik}, h_{jk}) = w_{ik}$. The distance between f_i and f_j is then defined as the weighted sum of letter flips of the two fragments

$$D(f_i, f_j) = \sum_{k=1}^{n} d(f_{ik}, f_{jk}) \ . \quad (2)$$

If $D(f_i, f_j) > 0$, it means that the two fragments f_i and f_j come from different copies of a chromosome or there are errors in sequencing. We call them conflicting, otherwise we call them agreeable. Similarly, the distance between a haplotype h_i and an SNP fragment f_j is defined in a similar way:

$$D(h_i, f_j) = \sum_{k=1}^{n} d(h_{ik}, f_{jk}) \ . \quad (3)$$

If the SNP fragments can be partitioned into two disjoint sets such that all of the SNP fragments in the same set are pairwise agreeable, then the SNP matrix is said to be feasible.

In order to improve the accuracy of the WMLF model, we introduce a new model that employs the genotype information, which can be obtained easily. For a genotype $g = (g_1, g_2, ..., g_n)$, we assign 0 to g_i when the i-th SNP site is wild type homozygous, and we assign it a 1 when it is mutant type homozygous. If the site is heterozygous, we assign it a 2. A pair of haplotypes h_1 and h_2 is said

to be compatible with genotype g if the following conditions hold for each SNP site k:

$$\begin{cases} \text{if } g_k \neq 2, & h_{1k} = h_{2k} = g_k \\ \text{if } g_k = 2, & h_{1k} = 0, h_{2k} = 1 \text{ or } h_{1k} = 1, h_{2k} = 0 \end{cases} . \tag{4}$$

The new model WMLF/GI is defined as follows:

Definition 1 (WMLF/GI). *Given a SNP matrix M with a weight matrix W and a genotype g, flip some elements (0 into 1 and vice versa) so that the weighted sum of the flips reaches a minimum and the resulting matrix is both feasible and g-compatible, i.e. the SNP fragments can be divided into two disjoint sets of pairwise agreeable fragments that determine a pair of haplotypes that is compatible with g.*

WMLF/GI is to flip some elements (satisfying the condition that the weighted sum is minimized) under the guidance of the genotype information so that the modified SNP fragments can be divided into two disjoint sets, and these two sets are used to determine a pair of haplotypes.

Theorem 1. *The WMLF/GI problem is NP-hard.*

Proof. The MLF/GI problem was proven to be NP-hard by Zhang et al [11]. To prove that the WMLF/GI problem is NP-hard, we will reduce the MLF/GI problem to the WMLF/GI problem.

We can transform an arbitrary instance of MLF/GI into an instance of WMLF /GI just by adding a special weight matrix W^*. Without loss of generality, we can map the value 1 to the elements of W^* when the corresponding letters are 0 or 1 in the SNP matrix, and if the element is a gap, we assign the value 0.

In the MLF/GI model, the distance between fragments or between a fragment and a haplotype is measured by the hamming distance, i.e. the number of mismatch characters. Since every element of W^* is 1 except for the gaps, the distance measures of MLF/GI are equal to formulas (2) and (3). Therefore, if a partition P^* has a minimum weighted sum of the flips and the resulting matrix is both feasible and g-compatible in the MLF/GI model, then it is also an optimal solution to the WMLF/GI model.

Conversely, an instance of WMLF/GI with weight matrix W^* is a special case of WMLF/GI. Therefore the converse holds also. This reduction can obviously be performed in polynomial time. □

3 A Heuristic Method Based on a Genetic Algorithm

In this section, we propose a genetic algorithm for solving the WMLF/GI problem. Genetic algorithms[4] are useful meta-heuristic algorithms and they have found successful application in many areas including those of computational biology.

3.1 The Hypothesis Space

In the genetic algorithm, an individual in a population is expressed by a bit vector. It represents a partition of SNP fragments in a SNP matrix and a feasible solution to the WMLF/GI problem. The length of an individual in the hypothesis space is the number of SNP fragments. The value 0 or 1 at the i-th position of an individual characterizes the partition membership of the i-th SNP fragment. Thus, all of the bit vectors having length m constitute the hypothesis space:

$$H = \{(f_1, f_2, ..., f_m) | f_i \in \{0, 1\}, i = 1, 2, ..., m\} \ .$$

Therefore the size of the hypothesis space is 2^m .

3.2 Construction of the Initial Population Under the Guidance of Genotype Information

Each individual in the initial population is generated by randomly giving either one of the two values 0 or 1 to each element f_i, but we can improve the efficiency of the genetic algorithm by employing genotype information in the construction of the initial population. If g_i is 0 or 1, it means that the i-th SNP site of all fragments must be 0 or 1 and we can modify the site with this value. If $g_i = 2$, then we cannot determine the value of the site. Therefore, if we modify only the sites of all fragments corresponding to the site of the genotype that has the value 0 or 1, then we will obtain a new SNP fragment matrix M^*. If a pair of fragments f_i and f_j in M^* has $D(f_i, f_j) = 0$, we presume that they are generated from the same copy of a chromosome, so we assign the same value 0 or 1 to all similar fragments. Then the hypothesis space will be reduced and the speed of the convergence to the best feasible solution will be increased. We show the effectiveness of the modification in improving the convergence speed in Section 4.

3.3 Haplotype Assembly Rule from a Partition

We first describe how to generate a pair of haplotypes from a partition $P = (P_1, P_2)$ that corresponds to an individual under the guidance of genotype. Let $C_{0j}(P_l)$, $C_{1j}(P_l)$ denote the weighted sum of letter flips of the wild and mutant type respectively in column j when we focus on the class P_l, i.e., $C_{0j}(P_l) = \sum_{f_i \in P_l, f_{ij}=0} w_{ij}$ and $C_{1j}(P_l) = \sum_{f_i \in P_l, f_{ij}=1} w_{ij}$. If $g_j \neq 2$, then let $h_{1j} = h_{2j} = g_j$. If $g_j = 2$, then h_{1j} and h_{2j} are determined by the following formula:

$$h_{lj} = \begin{cases} 0, & \text{if } C_{0j}(P_l) > C_{1j}(P_l) \\ 1, & \text{otherwise} \end{cases} \ . \tag{5}$$

where $l = 1, 2$ and $j = 1, 2, ..., n$. If $h_{1j} \neq h_{2j}$, then h_1 and h_2 are compatible with g at the j-th SNP site. Otherwise, h_1 and h_2 are not compatible with g at the j-th SNP site and we must modify h_{1j} or h_{2j}. For $h_{1j} = h_{2j} = 1$, if $C_{1j}(P_1) - C_{0j}(P_1) < C_{1j}(P_2) - C_{0j}(P_2)$, then set $h_{1j} = 0$, otherwise set $h_{2j} = 0$. For $h_{1j} = h_{2j} = 0$, if $C_{0j}(P_1) - C_{1j}(P_1) < C_{0j}(P_2) - C_{1j}(P_2)$, then set $h_{1j} = 1$,

otherwise set $h_{2j} = 1$. After modification, h_1 and h_2 are compatible with g at every SNP site. It is also easy to see that for a fixed partition, a pair of haplotypes compatible with a given genotype generated by this method has a lower weighted sum of letter flips with the given fragments than haplotypes generated by any other means. After determining a pair of haplotypes from a partition by the above method, we need a weight function to compute the total weighted sum of letter flips needed for the corresponding partition and generated haplotypes. We define a flip weight function by the following formula:

$$FW(P) = \sum_{l=1}^{2} \sum_{f_i \in P_l} D(h_l, f_i) \ . \tag{6}$$

where h_1 and h_2 are generated by the above assembly rule from the partition P. The goal of the WMLF/GI model is again to find a best partition of M by using the flip weight function.

3.4 Designation of the Fitness Function

In the genetic algorithm, for every individual in a population, we need to perform an evaluation to determine the goodness of an individual. We can use the flip weight function defined above as the fitness function, but for the convenience of evaluation, the following fitness function is used:

$$Fit((f_1, f_2, ..., f_m)) = mn - FW(P) \ . \tag{7}$$

where P denotes the partition that corresponds to an individual $(f_1, f_2, ..., f_m)$ and $0 < Fit() \leq mn$. The fitness of an individual is reversely proportional to the corresponding weighted sum of flips needed for error correction. It is easy to see that a SNP matrix is feasible if and only if there exists an individual $(f_1, f_2, ..., f_m)$ in the hypothesis space such that $Fit((f_1, f_2, ..., f_m)) = mn$.

3.5 Genetic Operators

There are several kinds of selection operators. Among them we adopt a tournament selection to increase the convergence speed and the accuracy. In addition, in order to yield a more diverse population, we use a roulette wheel selection operator to select individuals to crossover, and we newly create a crossover technique that changes only the randomly chosen parts of the heterozygous site in light of the genotype information. We adopt a single-point mutation in our genetic algorithm.

The overall scheme of the algorithm is given in Table 1.

4 Analysis of the Experimental Results

The proposed algorithm is implemented in the C language and tested on a single 32-bit system (Pentium 4, 2.8 GHz with 1GB RAM).

Table 1. Genetic algorithm for WMLF/GI

Algorithm $GA for Haplotype Assembly$
Input: SNP fragments matrix M, weight matrix W, genotype g population size PS, crossover rate CR, mutation rate MR, the maximum number of population generation GN
Output: a pair of haplotypes h_1, h_2
Begin
Generate a random initial population $P_0, k = 0$;
Modify the initial population under the guidance of a genotype g;
while$(k < GN)$ *do*
Compute the fitness of each individual in P_k;
Select $(1 - CR) \times PS$ individuals in P_k and add them to P_{k+1} using the tournament selection operator;
Generate $CR \times PS$ offstrings from P_k and add them to P_{k+1} using the roulette wheel selection operator and newly designed crossover operator;
Mutate $MR \times PS$ individuals in P_{k+1};
$k = k + 1$;
end do
return a pair of haplotypes assembled by the best individual;
end

We use the correct rate (R_c) as the measure of the performance of our algorithm. The correct rate is universally adopted [8,9,11,12] and it helps to compare the performance of our algorithm with the others. It is defined as follows. Let $h^* = (h_1^*, h_2^*)$ be the original haplotypes of the given chromosomes, and $h = (h_1, h_2)$ be the haplotypes reconstructed by an algorithm. The R_c is

$$R_c(h, h^*) = 1 - \frac{min\{D(h_1, h_1^*) + D(h_2, h_2^*), D(h_1, h_2^*) + D(h_2, h_1^*)\}}{2n} \quad . \tag{8}$$

where $D(h, h^*)$ is the hamming distance between the two haplotypes.

4.1 Experiment on Data from Chromosome *5q31*

To compare the performance of the WMLF model and the WMLF/GI model, we use the same data from the public Daly set [2] that is used in [12]. The data consist of genotypes for 103 SNPs in a 500 kilobase region of chromosome *5q31* from mother-father-child trios. The haplotype pairs of the 129 children from the trios can be inferred from the genotypes of their parents through pedigree information and the non-transmitted chromosomes as an extra 129 haplotype pairs. Markers for which both alleles could not be inferred are marked as missing. From the resulting 258 haplotype pairs, the ones with more than 20% missing alleles are removed, leaving 147 haplotype pairs as the test set.

We also make instances with the same parameter set used in [12]. The SNP matrix of every example consists of $m = 100$ SNP fragments, each of which is

Table 2. A comparison between the WMLF model and the WMLF/GI model

	$R_e = 0.05$			$R_e = 0.2$			$R_e = 0.3$			$R_e = 0.4$		
R_m	WMLF (wang)	WMLF (GA)	WMLF/GI (GA)	(wang)	(GA)	(GA)	(wang)	(GA)	(GA)	(wang)	(GA)	(GA)
0.1	0.978	1.0	1.0	0.978	0.997	0.999	0.995	0.9834	0.994	-	0.912	0.935
0.2	0.975	1.0	1.0	0.988	0.996	0.999	0.993	0.978	0.991	-	0.895	0.934
0.3	0.978	0.999	1.0	0.988	0.993	0.998	0.988	0.975	0.989	-	0.875	0.924
0.4	0.978	0.999	1.0	0.994	0.990	0.998	0.981	0.970	0.982	-	0.854	0.923
0.5	0.978	0.999	1.0	0.997	0.986	0.996	0.972	0.962	0.962	-	0.83	0.919
0.6	0.981	0.997	0.999	0.993	0.979	0.992	0.950	0.952	0.953	-	0.806	0.915
0.7	0.982	0.992	0.999	0.982	0.970	0.984	0.923	0.913	0.933	-	0.777	0.906
0.8	0.990	0.982	0.996	0.963	0.957	0.959	0.864	0.858	0.927	-	0.738	0.905
0.9	0.959	0.960	0.971	0.886	0.927	0.927	0.772	0.785	0.911	-	0.702	0.902

Table 3. The comparison of convergence speed between genetic algorithms with/without genotype information

	R_e= 0.05		R_e= 0.2		R_e= 0.3		R_e= 0.4	
R_m	without GI	GI	without GI	GI	without GI	GI	without GI	GI
0.1	31.891	5.578	35.285	4.034	33.897	3.360	33.095	4.231
0.2	30.707	5.163	34.707	2.755	33.829	2.360	34.537	2.755
0.3	30.217	2.952	33.040	3.149	34.108	2.503	34.408	3.612
0.4	30.721	3.489	33.217	2.442	34.659	2.843	33.326	2.761
0.5	31.136	2.993	32.428	3.251	34.918	3.619	33.272	3.435
0.6	31.312	2.836	32.115	3.074	34.530	3.122	32.721	3.156
0.7	31.823	3.612	34.238	3.707	34.217	3.619	32.544	3.285
0.8	30.244	4.442	32.455	4.306	34.115	3.367	32.476	4.585
0.9	28.748	6.217	32.272	7.156	31.979	6.346	31.775	6.503

generated by randomly copying either of the two seed haplotypes. The gaps in every SNP fragment are also produced randomly at missing rate R_m. The SNP error in a correct SNP fragment is simulated by turning a 0 into 1 or vice versa at some SNP error rate R_e. The entries of the weight matrix W are normally distributed and the corresponding parameters are μ(mean) and σ^2(variance). Throughout the simulations, $\mu = 0.9$ for a correct SNP site and $\mu = 0.8$ for an error SNP site. In both of these cases, $\sigma^2 = 0.05$. For the parameters of the designed genetic algorithm, we set $PS = 400$, $CR = 0.8$, $MR = 0.01$, and $GN = 150$.

Table 2 compares the WMLF model (which consists of the Dynamic Clustering Algorithm [12] and genetic algorithm) and the WMLF/GI model with different parameter settings: $R_e = 0.05, 0.2, 0.3, 0.4$. In each of these three cases, R_m ranges from 0.1 to 0.9. The value of each entry of this table is the average of the correct rates over all of the 147 instances at each parameter setting. From Table 2, we can see that the accuracy of the haplotype assembly of the WMLF/GI model is

generally better than that of the WMLF model. Furthermore, the difference between them becomes significant as the SNP error rate and the gap rate increase. This indicates the validity of introducing the genotype information to the WMLF model to improve the accuracy.

Table 3 illustrates the effectiveness of the genotype information in the convergence speed of the genetic algorithm. We define the convergence generation number as the last generation number after which there is no more improvement in the correct rate through 150 generations in each instance. It shows the average convergence generation number of 147 instances for each parameter setting. It can be seen that the speed of convergence of our genetic algorithm is greatly improved by employing the genotype information without loss of accuracy.

5 Conclusion

The haplotype assembly problem is an important problem in computational biology. In this paper, we first proposed the WMLF/GI model as an improvement of WMLF and showed its NP-hardness. We also suggested a genetic algorithm to solve this problem more accurately and efficiently. This algorithm employs related genotype information not only in inferring a pair of haplotypes but also in building an initial population that increases the algorithms accuracy and speed of convergence. The computational results on a real data set show that the designed algorithm performs well and the WMLF/GI model assembles the haplotypes more accurately than the WMFL model especially when the SNP fragments have a high error rate and a high gap rate.

In future works, we will extend our experiments to other data sets and compare its performance with the MFL model that employs genotype information.

References

1. Cilibrasi, R., Iersel, L.V., Kelk, S., Tromp, J.: On the Complexity of Several Haplotyping Problems. In: Casadio, R., Myers, G. (eds.) WABI 2005. LNCS (LNBI), vol. 3692, pp. 128–139. Springer, Heidelberg (2005)
2. Daly, M.J., Rioux, J.D., Schaffner, S.F., Hudson, T.J., Lander, E.S.: High-resolution haplotype structure in the human genome. Nature Genetics 29, 229–232 (2001)
3. Greenberg, H.J., Hart, W.E., Lancia, G.: Opportunities for Combinatorial Optimization in Computational Biology. INFORMS Journal on Computing 16(3), 211–231 (2004)
4. Goldberg, D.E.: Genetic Algorithms in serarch, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
5. Rizzi, R., Bafna, V., Istrail, S., Lancia, G.: Practical Algorithms and Fixed-Parameter Tractability for the Single Individual SNP Haplotyping Problem. In: Guigó, R., Gusfield, D. (eds.) WABI 2002. LNCS, vol. 2452, pp. 29–43. Springer, Heidelberg (2002)
6. Stephens, J.C., et al.: Haplotype variation and linkage disequilibrium in 313 human genes. Science 293, 489–493 (2001)
7. Terwilliger, J.D., Weiss, K.M.: Linkage disequilibrium mapping of complex disease: fantasy or reality? Current Opinion in Biotechnology 9(6), 578–594 (1998)

8. Wang, Y., Feng, E., Wang, R., Zhang, D.: The haplotype assembly model with genotype information and iterative local-exhaustive search algorithm. Computational Biology and Chemistry 31, 288–293 (2007)
9. Wang, R.S., Wu, L.Y., Li, Z.P., Zhang, X.S.: Haplotype reconstruction from SNP fragments by minimum error correction. Bioinformatics 21(10), 2456–2462 (2005)
10. Zhang, X.S., Wang, R.S., Wu, L.Y., Chen, L.: Models and Algorithms for Haplotyping Problem. Current Bioinformatics 1, 105–114 (2006)
11. Zhang, X.S., Wang, R.S., Wu, L.Y., Zhang, W.: Minimum Conflict Individual Haplotyping from SNP Fragments and Related Genotype. Evolutionary Bioinformatics Online 2, 271–280 (2006)
12. Zhao, Y.Y., Wu, L.Y., Zhang, J.H., Wang, R.S., Zhang, X.S.: Haplotype assembly from aligned weighted SNP fragments. Computational Biology and Chemistry 29, 281–287 (2005)

Estimating Hybrid Frequency Moments of Data Streams

Extended Abstract*

Sumit Ganguly, Mohit Bansal, and Shruti Dube

Indian Institute of Technology, Kanpur

Abstract. A two-dimensional stream is a sequence of coordinate-wise updates to a two-dimensional array $(A_{i,j})_{1 \leq i,j \leq n}$. The hybrid frequency moments $F_{p,q}(A)$ is defined as $F_{p,q}(A) = \sum_{j=1}^{n} (\sum_{i=1}^{n} |A_{ij}|^p)^q$. For every $0 < \epsilon < 1$ and $p, q \in [0, 2]$, we present an $O(\epsilon^{-6} \text{poly}(\log n, \log m, \log(1/\epsilon)))$ space algorithm for the problem of estimating $F_{p,q}$, where, m is an upper bound on $\max_{i,j} |A_{i,j}|$.

1 Introduction

The data stream model of computation is an abstraction for a variety of practical applications arising in network monitoring, sensor networks, RF-id processing, database systems, online web-mining, etc.. A problem of basic utility and relevance in this setting is the following *hybrid frequency moments estimation* problem. Consider a networking application where a stream of packets with schema (*src-addr*, *dest-addr*, *nbytes*, *time*) arrives at a router. The problem is to warn against the following scenario arising out of a distributed denial of service attack, where, a few destination addresses receive messages from an unusually large number of distinct source addresses. This can be quantified as follows: let A be an $n \times n$ matrix where $A_{i,j}$ is the count of the number of messages from node i to node j. Then, $A^0_{i,j}$ is 1 if i sends a message to j and is 0 otherwise. Thus, $\sum_{i=1}^{n} A^0_{i,j}$ counts the number of distinct sources that send at least one message to j. Define the hybrid moment $F_{0,2}(A) = \sum_{j=1}^{n} (\sum_{i=1}^{n} A^0_{i,j})^2$. In an attack scenario, $F_{0,2}(A)$ becomes large compared to its average value. Since n can be very large (e.g., in the millions), it is not feasible to store and update the traffic matrix A at network line speeds. We propose instead to use the data streaming approach to this problem, namely, to design a sub-linear space data structure that, (a) processes updates to the entries of A, and, (b) provides a randomized algorithm for approximating the value of $F_{0,2}(A)$.

Given an n-dimensional vector from $\mathbb{Z}^n$ or $\mathbb{R}^n$, its pth frequency moment $F_p(a)$ is defined as $F_p(a) = \sum_{j=1}^{n} |a_i|^p$ and its pth norm is defined as $\|a\|_p = (\sum_{j=1}^{n} |a_j|^p)^{1/p}$. Given an $n \times n$ integer matrix A with columns $A_1, A_2, \ldots, A_n$,

* Full version may be found at "http://www.cse.iitk.ac.in/users/sganguly/mixed-full.pdf"

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 55–66, 2008.

the hybrid frequency moment $F_{p,q}(A)$ is defined as the qth moment of the n-dimensional vector $[F_p(A_1), F_p(A_2), \ldots, F_p(A_n)]$ [14]. That is,

$$F_{p,q}(A) = \sum_{j=1}^{n} \left(\sum_{i=1}^{n} A_{i,j}^{p} \right)^{q} = \sum_{j=1}^{n} \|A_j\|_p^{pq} \qquad p \geq 0, q \geq 0, p, q \in \mathbb{R} \ .$$

Data Stream Model. A data stream is an unbounded sequence σ of records of the form (pos, i, j, Δ), where, $i, j \in \{1, 2, \ldots, n\}$ and $\Delta \in \mathbb{Z}$ is the change to the value of $A_{i,j}$. The *pos* attribute is simply the sequence number of the record. Each input record (pos, i, j, Δ) changes $A_{i,j}$ to $A_{i,j} + \Delta$. In other words, the $A_{i,j}$ is the sum of the changes made to the (i, j)th entry since the inception of the stream:

$$A_{i,j} = \sum_{(pos,i,j,\Delta) \in \sigma} \Delta, \quad 1 \leq i, j \leq n \ .$$

In this paper, we consider the problems of estimating $F_{p,q}$ and allow general matrix streams, that is, matrix entries may be positive, zero or negative.

Prior work. Estimating hybrid frequency moments $F_{p,q}(A)$ is clearly a generalization of the problem of estimating the frequency moment $F_p(a)$ of an array a. The problem of estimating $F_p(a)$ has been extensively studied in the data stream model where the input is a stream of updates to the components of a. We say that a randomized algorithm computes an ϵ-approximation to a real valued quantity L if it returns $\hat{L}$ such that $|\hat{L} - L| < \epsilon L$, with probability $\geq \frac{3}{4}$.

Alon, Matias and Szegedy [1] present a seminal randomized sketch technique for ϵ approximation of $F_2(a)$ in the data streaming model using space $O(\epsilon^{-2} \log F_1(a))$ bits. Using the techniques of [1], it is easily shown that deterministically estimating $F_p(a)$ for any real $p \geq 0$ requires $\Omega(n)$ space. Hence, work in the area of sub-linear space estimation of moments has considered only randomized algorithms. Estimation of $F_0(a)$ was first considered by Flajolet and Martin in [7]; the work in [1] presents a modern version of this technique for estimating $F_0(a)$ to within a constant multiplicative factor and using space $O(\log n)$. Gibbons and Tirthapura [10] present an ϵ-approximation algorithm for $F_0(a)$ using space $O(\epsilon^{-2} \log F_1(a))$; this is further improved in [3]. The use of p-stable sketches was pioneered by Indyk [11] for estimating $F_p(a)$, for $0 < p \leq 2$, using space $O(\epsilon^{-2}(\log F_1(a))(\log \epsilon^{-1} + \log \log F_1(a) + \log n))$. Li [13] presents the geometric mean estimator based on stable sketches as an improvement over Indyk's median estimator. Indyk and Woodruff [12] present a near optimal space algorithm for estimating F_p, for $p > 2$. Woodruff [16] presents an $\Omega(\epsilon^{-2})$ space lower bound for the problem of estimating F_p, for all $p \geq 0$, implying that the stable sketches technique is space optimal up to logarithmic factors. A space lower bound of $\Omega(n^{1-2/p})$ was shown for the problem F_p in a series of developments [1,2,5]. Cormode and Muthukrishnan [6] present an algorithm for obtaining an ϵ-approximation for $F_{0,2}(A)$ using space $\tilde{O}(\sqrt{n})$. This is the only prior work on estimating hybrid moments of a matrix in the data stream model.

Contributions. We present a family of randomized algorithms for the problem of estimating hybrid moments $F_{p,q}(A)$ of a matrix A in the data stream model. For every $p, q \in [0, 2]$, we present an algorithm for computing an ϵ-approximation of $F_{p,q}$ using space $O(\epsilon^{-6}\ \mathrm{poly}(\log n, \log m, \log(1/\epsilon)))$ bits. For $p \in [0, 2]$ and $q > 2$, the algorithm can be extended using the Hss technique [4] to obtain an ϵ-approximation for $F_{p,q}$ using $\tilde{O}(n^{1-2/q})$ bits. The proposed algorithms improve upon the space requirement for the hybrid moments estimation problem.

Organization. The remainder of the paper is organized as follows. Section 2 presents some preliminaries on stable distributions and its applications to moment estimation. In Section 3, we present our algorithm for estimating $F_{p,q}$ for $p, q \in [0, 2]$.

2 Preliminaries

In this section, we review salient properties of stable distributions and briefly review Indyk's [11] and Li's [13] techniques for estimating moments of one-dimensional vectors in the data streaming model. *Notation*: Given a random variable y that follows a distribution D, we use the notation $y \sim D$ to denote this fact.

Stable Distributions

We will use random variables drawn from stable distributions. There are two popular parameterizations of stable distributions, see Nolan [15], namely, $S^0(q, \beta, \gamma, \delta)$ and $S^1(q, \beta, \gamma, \delta)$, where, $q \in (0, 2], \beta \in [-1, 1], \gamma \in \mathbb{R}^+$ and $\delta \in \mathbb{R}$. In these notations, q is the index of stability, β is the skew parameter, γ is the scale parameter and δ is the shift. Suppose that $X_0 \sim S^0(q, \beta, \gamma, \delta)$ and $X_1 \sim S^1(q, \beta, \gamma, \delta)$. Then, the distributions are related as follows: $\gamma^{-1}(X_{S^1} - \delta) \sim \gamma^{-1}(X_{S^0} - \delta) + \tan\frac{\pi q}{2}$. A well-known result of Levy shows that the tail of stable distributions is asymptotically distributed as Pareto distribution. Let $X \sim S^0(q, \beta, 1, 0)$ and $x > 0$. Then,

$$\Pr\{X < -x\} \approx C_q(1-\beta)x^{-q}, x \to \infty, \text{ for } q < 2, \beta > 1. \text{ [Levy 1934]} \qquad (1)$$

where, $C_q = \frac{\Gamma(q)}{\pi} \sin\frac{\pi q}{2}$. Indyk [11] proposed the use of stable sketches for estimating $F_q(a)$, $q \in [0, 2]$, in the streaming model. A stable sketch is a linear combination $X = \sum_{i=1}^{n} a_i s_i$, where, $s_i \sim S^1(q, 0, 1, 0)$ and the s_i's are independent. By property of stable distributions, $X \sim S^1(p, 0, \|a\|_q, 0)$. The problem now reduces to the estimation of the scale parameter of the distribution of X. Indyk proposed keeping $t = O(\frac{1}{\epsilon^2})$ independent stable sketches $X_1, X_2, \ldots, X_t$ and returning $\hat{F}_q(a) = \mathrm{median}_{r=1}^{t}|X_r|^q$. Li [13] uses the geometric means estimator $\hat{F}_q(a) = C_L \prod_{r=1}^{t}|X_r|^{q/t}$ and shows that this is unbiased for a proper choice of the constant C_L. Both estimators satisfy $|\hat{F}_q(a) - F_q(a)| \leq \epsilon \hat{F}_q(a)$, with probability $\frac{7}{8}$. We will refer to Indyk's median estimator or Li's geometric means estimator for $F_p(a)$ as $\mathrm{StableEst}^{(q)}(\{X_1, X_2, \ldots, X_t\})$, where, $q \in (0, 2]$ is the stability index.

Distribution Distances

The distance between distributions $D(X)$ and $D(Y)$ is defined as per standard convention as follows.

$$\|D(X) - D(Y)\| \stackrel{\text{def}}{=} \int_{t=-\infty}^{\infty} |f_X(t) - f_Y(t)|dt \ .$$

We have the following straightforward consequences. The proofs of Lemmas 1 and 2 are given in Appendix A.

Lemma 1. *Suppose that $X \sim D(X)$, $Y \sim D(Y)$, $\|D(X) - D(Y)\| \leq \delta$, $g : \mathbb{R} \to \mathbb{R}$ is differentiable, and $g'(x) = 0$ at a finite number of points. Then, $\|D_{g(X)} - D_{g(Y)}\| \leq \delta$.*

Lemma 1 generalizes to the case when X is a vector of random variables $X = (X_1, X_2, \ldots, X_k)$ and $g(X) = g(X_1, \ldots, X_k)$ is a real valued function, all of whose partial derivatives exist, and that is zero on a finite number of points in $\mathbb{R}^k$. Suppose that $X_1, X_2, \ldots, X_s$ are independent and identically distributed. Their joint distribution is denoted as $D(X_1, X_2, \ldots, X_s)(x_1, x_2, \ldots, x_s) = \prod_{j=1}^{s} f_{X_j}(x_j)$.

Lemma 2. *Suppose that random variables $X_1, X_2 \ldots, X_s$ are each identically and independently distributed as $D(X)$ and the random variables $Y_1, \ldots, Y_s$ are identically and independently distributed as $D(Y)$. Suppose that $\|D(X) - D(Y)\| \leq \delta$. Then, $\|D(X_1, X_2, \ldots, X_s) - D(Y_1, Y_2, \ldots, Y_s)\| \leq s\delta$.*

Hybrid Frequency Moments: Preliminaries

Estimation of hybrid moments generalizes the problem of estimating the regular moment F_p, in particular $F_{p,p}(A) = F_p(a)$ where a is the n^2-dimensional vector obtained by stringing out the matrix A row-wise (or column-wise). For brevity, we will assume that the matrix A is implicit and and refer to $F_{p,q}(A)$ simply as $F_{p,q}$. We first note that for the estimation of $F_{p,q}$ where $p, q \in [0, 2]$, it is sufficient to assume that $|p - q| \geq \frac{\epsilon}{4 \log F_{1,1}}$, otherwise, one can estimate $F_{p,q}$ as the regular moment $F_{p,p}(A)$ using existing techniques.

Fact 3. *Let $\epsilon < 1$ and $p, q \in [0, 2]$. If $|p - p'| \leq \frac{\epsilon}{4 \log F_{1,1}}$, then, $|F_{p,p} - F_{p,q}| \leq \epsilon F_{p,q}$. If $|q - q'| \leq \frac{\epsilon}{6 \log F_{1,1}}$, then, $|F_{q,q} - F_{p,q}| \leq \epsilon F_{p,q}$.* □

Proof (Of Fact 3). Let $|p - p'| = \epsilon_p$ and $|q - q'| = \epsilon_q$. By triangle inequality and Taylor series expansion up to first order term,

$$|F_{p',q} - F_{p,q}| \leq \epsilon_p q F_{\max(p,p'),q} \log F_{1,1}, \text{ and } |F_{p,q'} - F_{p,q}| \leq \epsilon_q F_{p,\max(q,q')} \log F_{1,1}.$$

If $p' < p$, then, $|F_{p',q} - F_{p,q}| \leq \epsilon_p q F_{p,q} \log F_{1,1}$. If $p' > p$ and $\epsilon_p q \log F_{1,1} < 1$,

$$|F_{p',q} - F_{p,q}| \leq \epsilon_p q F_{p',q} \log F_{1,1}, \text{ that is, } |F_{p',q} - F_{p,q}| \leq \frac{\epsilon_p q F_{p,q} \log F_{1,1}}{1 - \epsilon_p q \log F_{1,1}} \ .$$

Similarly, if $q' < q$, then, $|F_{p,q'} - F_{p,q}| \leq \epsilon_q F_{p,q} \log F_{1,1}$ and if $q' > q$ and $\epsilon_q \log F_{1,1} < 1$ then,

$$|F_{p,q'} - F_{p,q}| \leq \frac{\epsilon_q F_{p,q} \log F_{1,1}}{1 - \epsilon_q \log F_{1,1}} \ .$$

□

We will henceforth assume without loss of generality that in the estimation of $F_{p,q}$, $|p - q| \geq \frac{\epsilon}{4 \log F_{1,1}}$.

Simple Cases for Hybrid Moments Estimation

In this section, we study some specific combinations of the parameters p and q for which $F_{p,q}$ is easily estimated using variations of existing techniques. Consider estimation of $F_{0,1}$. This is easily accomplished by keeping a distinct sample [10] S of capacity s over the set $\{(i,k) : A_{i,k} \neq 0\}$. By property of distinct sample, if $s = O(\frac{1}{\epsilon^2})$, then, $|S| \cdot 2^l$ is an ϵ-close estimator to $|\{(i,k) : A_{i,k} \neq 0\}|$. The latter expression is exactly $F_{0,1}$. This implies that $F_{0,1}$ is estimated using space $O(\frac{1}{\epsilon^2} \log m)$ bits. To handle deletions, an updatable distinct sample can be used [8]. An estimate for $F_{0,q}$ for $q \in [0,1]$ can be obtained as follows. Keep a distinct sample S as above. For a column index k, estimate $\hat{f}_k$ as the number of distinct entries in S of the form (i,k). By distinct sample property, if $s = \Omega(\frac{1}{\epsilon^3})$ and $F_0(A_k) \geq \epsilon F_{0,1}(A)$, then, $\hat{f}_k \geq \frac{2\epsilon}{3}|S|$ with high probability, that is, columns with high relative $F_0(A_k)$ can be discovered with high probability. One can now use the HSS technique [9] to estimate $F_{0,p}(A)$ in the same way that F_1-based frequent items are used to estimate F_p. The space requirement of this technique becomes $\tilde{O}(\frac{1}{\epsilon^6})$.

3 Estimating Hybrid Frequency Moments $F_{p,q}$, $p, q \in [0,2]$

In this section, we present an estimation algorithm for $F_{p,q}$, $p, q \in [0,2]$.

3.1 Bi-linear Stable Sketches

Consider two families of fully independent stable variables $\{x_{i,j} : 1 \leq i \leq j \leq n\}$ and $\{\xi_j : 1 \leq j \leq n\}$, where, $x_{i,j} \sim S^1(p,0,1,0)$ and $\xi_j \sim S^1(q,\beta,\gamma,\delta)$ conditional on $\xi_j \geq 0$, for each $j \in \{1,\ldots,n\}$. The parameters, β,γ and δ are set in the analysis. The conditional property of ξ_j's is implemented as follows: if any of the ξ_j is negative, then, the entire family is discarded. The analysis will show that for the choice of the parameters $\mathsf{Pr}\{\xi_j \geq 0\} > 1 - \frac{1}{n^2}$. A p,q bi-linear stable sketch is defined as

$$X = \sum_{j=1}^{n} \sum_{i=1}^{n} A_{i,j} x_{i,j} \xi_j^{1/p} \ .$$

Corresponding to each stream update (pos, i, j, Δ), the bi-linear sketch is updated as follows:

$$X := X + \Delta \cdot x_{i,j} \cdot \xi_j^{1/p} \ .$$

A collection of $s_1 s_2$ bi-linear sketches $\{X_{u,v} \mid 1 \leq u \leq s_1, 1 \leq v \leq s_2\}$ is kept such that for each distinct value of v, the family of sketches $\{X_{u,v}\}_{u=1,2,\ldots,s_1}$ uses the independent family of stable variables $\{x_{i,j}(u,v)\}$ but uses the same family of stable variables $\{\xi_i(v)\}$. That is,

$$X(u,v) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} x_{i,j}(u,v)(\xi_i(v))^{1/p}, \qquad u = 1, \ldots, s_1, v = 1, \ldots, s_2 \ .$$

The estimate $\hat{F}_{p,q}$ is obtained using the following steps.

Algorithm. BILINSTABLE$(p, q, s_1, s_2, \{X(u,v)\}_{u\in[1,s_1],v\in[1,s_2]})$.

1. For $v = 1, 2, \ldots, s_2$, calculate $\hat{Y}(v)$ as follows.

$$\hat{Y}(v) = \text{StableEst}^{(p)}(\{X(u,v)\}_{u=1,\ldots,s_1}) \ .$$

2. Return the estimate $\hat{F}_{p,q}$ as follows.

$$\hat{F}_{p,q} = \frac{1}{\gamma^q} \cdot \text{StableEst}^{(q)}(\{\hat{Y}(v) \mid v = 1, \ldots, s_2\})$$

Fig. 1. Algorithm BILINSTABLE for estimating $F_{p,q}$

3.2 Analysis of Bi-linear Stable Sketches

In this section, we present an analysis of the bi-linear stable sketch algorithm. The analysis is divided into two basic cases, namely, (a) $p \in (0,2)$ and $q \in (0,1)$ and (b) $p \in (0,2]$ and $q \in (1,2]$. The cases, $p = 0$ or $q = 0$ or $q = 1$ are treated as singularities and considered separately. Recall that by Fact 3, we can assume that $|p - q| > \frac{\epsilon}{4 \log F_{1,1}}$.

Lemma 4. *For each* $0 < p \leq 2$ *and* $0 < q < 1$*, the estimator* BILINSTABLE$(p, q, s_1, s_2, \{X(u,v)\}_{u\in[1,s_1],v\in[1,s_2]})$ *with parameters* $\beta = 0$*,* $\delta = 0$*,* $\gamma = 1$*,* $s_2 = \Theta(\frac{1}{q^2\epsilon^2})$ *and* $s_1 = \Theta(\frac{1}{p^2\epsilon^2} \log \frac{1}{\epsilon q})$ *satisfies* $|\hat{F}_{p,q} - F_{p,q}| \leq 3\epsilon F_{p,q}$ *with probability* $\frac{3}{4}$. □

Proof. For $0 < q < 1$, $S^1(q, 0, 1, 0)$ is non-negative [15]. Therefore, $\xi_i \sim S^1(q, 0, 1, 0)$ and by property of StableEst, $\hat{Y}(v) = \lambda_v \sum_{j=1}^{n} (\|A_j\|_p)^p \xi_j$, where, $1-\epsilon \leq \lambda_v \leq 1+\epsilon$, with probability $1 - \delta'$, for $v \in [1 \ldots s_2]$. Denote the jth column of A as A_j, $j = 1, 2, \ldots, n$.

$$\hat{Y}(v) = \lambda_v \sum_{j=1}^{n} (\|A_j\|_p)^p \ \xi_j \sim S^1(q, 0, F_{p,q}(A) \cdot \lambda_v^{1/q}, 0) \ .$$

Suppose $1 - \epsilon \leq \lambda_v \leq 1 + \epsilon$ for each $v \in [1, s_2]$. If $s_2 = \Theta(\frac{1}{q^2\epsilon^2})$ then by property of StableEst [13], $|\hat{F}_{p,q} - \lambda F_{p,q}| \leq \epsilon \lambda F_{p,q}$ with probability $\frac{7}{8} - s_2\delta'$, for some λ

satisfying $1-\epsilon \leq \lambda \leq 1+\epsilon$. Therefore,

$$|\hat{F}_{p,q} - F_{p,q}| \leq F_{p,q}(\epsilon\lambda + (\lambda - 1)) < 3\epsilon F_{p,q} \quad \text{with prob. } \frac{6}{8} \ .$$

provided, $\delta' \leq \frac{1}{8s_2}$. Since, $s_2 = O(1/(q^2\epsilon^2))$, therefore, $s_1 = O(\frac{1}{\epsilon^2 p^2} \log \frac{1}{\delta'}) = O(\frac{1}{\epsilon^2 p^2} \log \frac{1}{\epsilon q})$. This proves the lemma. □

We now consider the analysis for $0 < p \leq 2$ and $1 \leq q < 2$.

Lemma 5. *Let m be an upper bound on the maximum absolute value of any matrix entry $A_{i,j}$. For each $0 < p \leq 2$ and $1 < q < 2$, the bi-linear stable sketch estimator* BilinStable(p, q, A, s_1, s_2) *given in Figure 1 with the parameters $\beta = 1$, γ given by (3) and $\delta = \frac{\epsilon}{2m}$ satisfies $|\hat{F}_{p,q} - F_{p,q}| < 5\epsilon F_{p,q}$, provided, $s_2 = \Theta(\frac{1}{q^2\epsilon^2})$ and $s_1 = \Theta\left(\frac{1}{p^2\epsilon^2}\left(\log \frac{1}{q\epsilon}\right)\right)$.*

Proof. Let $U \sim S^0(q, \beta, 1, 0)$ and $\xi \sim S^1(q, \beta, \gamma, \delta)$. There exists a constant $x_0(q, \beta)$ such that by Levy's result (1),

$$\mathsf{Pr}\{U < -x\} < 2C_q(1-\beta)x^{-q}, \text{ for } x > x_0(q,\beta) > 0. \tag{2}$$

Then, $\frac{\xi - \delta}{\gamma} \sim U + \tan\frac{\pi q}{2}$. Therefore,

$$\mathsf{Pr}\{\xi < 0\} = \mathsf{Pr}\left\{U < -\frac{\delta}{\gamma} + \tan\frac{\pi q}{2}\right\}$$

Choose δ, γ such that

$$\frac{\delta}{\gamma} - \tan\frac{\pi q}{2} > x_0(q,\beta) \text{ and } 2C_q(1-\beta)\left(\frac{\delta}{\gamma} - \tan\frac{\pi q}{2}\right)^{-q} < \frac{1}{8n^3} \ .$$

This is implied if γ satisfies

$$\gamma = \delta\left(1 + \max\left(x_0(q,\beta) - \tan\frac{\pi q}{2}, (4C_q n^2 s_2(1-\beta))^{-1/q} - \tan\frac{\pi q}{2}\right)\right)^{-1} \ . \tag{3}$$

Assume (3) holds for the rest of the discussion. Then,

$$\mathsf{Pr}\{\xi < 0\} = \mathsf{Pr}\left\{U < -\frac{\delta}{\gamma} + \tan\frac{\pi q}{2}\right\} < \mathsf{Pr}\{U < x_0(q,\beta)\} < \frac{1}{8n^3} \ .$$

Define η as the scale factor

$$\eta = (1 - \mathsf{Pr}\{\xi < 0\})^{-1} \leq \left(1 - \frac{1}{8n^3}\right)^{-1} \ . \tag{4}$$

Let $V \sim S^1(q, \beta, \gamma\eta, \delta)$, where, $\delta = 1$ and γ satisfies (3). Therefore,

$$|D(\xi) - D(V)| < \frac{1}{8n^3} \ . \tag{5}$$

By Lemma 2, the joint product distributions have a distance bounded as follows, where, the V_i's are each independently distributed as $S^1(q, \beta, \gamma, \delta)$ and the ξ_i's are independently distributed as $S^1(q, \beta, \gamma, \delta)$, conditional on the fact that $\xi_i \geq 0$.

$$\|D(\{\xi_i(v)\}_{i\in[1,n],v\in[1,s_2]}) - D(\{V_i(v)\}_{i\in[1,n],v\in[1,s_2]})\| < ns_2 \cdot \frac{1}{8n^3} = \frac{s_2}{8n^2} \ . \quad (6)$$

For $(1-\epsilon) \leq \lambda_v \leq (1+\epsilon)$ and $v \in [1, s_2]$, define

$$Z(v) = \lambda_v \eta \sum_{i\in[1,n]} \|A_i\|_p^p V_i(v) \text{ and } Z'(v) = \lambda_v \sum_{i\in[1,n]} \|A_i\|_p^p \xi_i(v), \quad v = 1, 2, \ldots, s_2. \quad (7)$$

The value of λ_v is chosen so that $Z'(v)$ is the same as the variable $\hat{Y}(v)$ used in the algorithm of Figure 1 with probability $1-\delta'$. This follows from the property of StableEst[11,13]. Furthermore, let $\eta = (1 - \mathsf{Pr}\left\{S^1(q, 1, \gamma, \delta) < 0\right\})^{-1}$, as given by (4). We now condition the remainder of the analysis on $Z'(v) = \hat{Y}(v)$, $v = 1, 2, \ldots, s_2$.

By Lemma 1, taking the function $g(X_1, \ldots, X_n) = \sum_{i\in[1,n]} \|A_i\|_p^p X_i$,

$$\|D(Z'(v)) - D(Z(v))\| \leq \|D(\{\xi_i(v)\}_{i\in[1,n],v\in[1,s_2]}) - D(\{V_i(v)\}_{i\in[1,n],v\in[1,s_2]})\| \leq \frac{s_2}{8n^2}, \qquad v = 1, 2, \ldots, s_2 \ .$$

By property of stable distributions,

$$Z(v) \sim S^1(q, \beta, (F_{p,q}(A))^{1/q}\gamma\eta\lambda_v, F_{1,1}(A)\delta) \ .$$

By Lemma 1 and Lemma 2,

$$|D(Z'(v)) - D(Z(v))| \leq \frac{s_2}{4n^2} \qquad v = 1, 2, \ldots, \lfloor s_2/2 \rfloor \ .$$

Finally, suppose we use Li's estimator of geometric means, since, the estimator is a continuous and differentiable function of its arguments and Lemma 1 is applicable [1]. Therefore, by Lemma 2,

$$\left| D(\text{StableEst}^q(\{Z'(v)\}_{v\in[1,s_2]})) - D(\text{StableEst}^q(\{Z(v)\}_{v\in[1,s_2]}) \right| \leq \frac{s_2^2}{4n^2} \quad (8)$$

Suppose $s_2 = \Theta(\frac{1}{q^2\epsilon^2})$. Using Li's estimator [13], we have with probability 7/8

$$\left|\text{StableEst}^q(\{Z(v)\}_{v\in[1,s_2]} - F_{p,q}(A)\gamma\eta\lambda_v\right| \leq F_{p,q}(A)\gamma\eta\lambda_v + F_{1,1}(A)\delta\eta \ .$$

By (8), and multiplying with scale factor γ^{-1}, it follows that

$$\left|\gamma^{-1}\text{StableEst}^q(\{Z'(v)\}_{v\in[1,s_2]}) - F_{p,q}(A)\eta\lambda_v\right| \leq \epsilon F_{p,q}(A)\eta\lambda_v + F_{1,1}(A)\frac{\delta\eta}{\gamma}$$

[1] Indyk's median estimator can also be used by appropriately strengthening Lemma 1.

with probability $\frac{7}{8} - \frac{s_2^2}{4n^2}$. Substituting $\lambda_v \in (1 \pm \epsilon)$, and adding the error probability $\mathsf{Pr}\left\{Z'(v) \neq \hat{Y}(v), v \in [1, s_2]\right\} \leq s_2\delta'$, thereby unconditioning the analysis on this event,

$$\left|\gamma^{-1}\text{StableEst}^q(\{Z'(v)\}_{v\in[1,s_2]}) - F_{p,q}(A)\right| \leq (3\epsilon + (1+\epsilon)(\eta-1))F_{p,q}(A) + F_{1,1}(A)\frac{\delta\eta}{\gamma}$$

with probability $\frac{7}{8} - \frac{s_2^2}{4n^2} - s_2\delta'$. Recall that $\eta \leq (1 - 1/(8n^3))^{-1} \leq (1 + 1/(4n^3))$ from (4) and $\epsilon \geq \frac{1}{n}$. Therefore,

$$\left|\gamma^{-1}\text{StableEst}^q(\{Z'(v)\}_{v\in[1,s_2]} - F_{p,q}(A)\right| \leq 4\epsilon F_{p,q}(A) + F_{1,1}(A)\frac{2\delta}{\gamma} \quad (9)$$

with prob. $\frac{7}{8} - \frac{s_2^2}{4n^2} - s_2\delta'$.

Choice of δ. By choosing $\delta < \frac{\epsilon F_{p,q}}{2F_{1,1}(A)}$, (9) becomes

$$\left|\gamma^{-1}\text{StableEst}^q(\{Z'(v)\}_{v\in[1,s_2]}) - F_{p,q}(A)\right| \leq 5\epsilon F_{p,q}(A)$$

with probability $\frac{7}{8} - \frac{s_2^2}{4n^2} - s_2\delta'$. The condition $\delta < \frac{\epsilon F_{p,q}(A)}{2F_{1,1}(A)}$ is simply enforced by letting $\delta < \frac{\epsilon F_{0,0}(A)}{2F_{1,1}(A)} \leq \frac{\epsilon}{2m}$, where, m is an upper bound on the maximum absolute value of a matrix element.

To bound the error probability, we let $\delta' < \frac{1}{8s_2}$. By property of StableEst, it is sufficient to let $s_2 = \Theta(\frac{1}{q^2\epsilon^2})$ and $s_1 = \Theta(\frac{1}{p^2\epsilon^2}\log\frac{1}{\delta'}) = \Theta\left(\frac{1}{p^2\epsilon^2}\left(\log\frac{1}{q} + \log\frac{1}{\epsilon}\right)\right)$. This then implies that

$$\left|\gamma^{-1}\text{StableEst}^q(\{Z'(v)\}_{v\in[1,s_2]}) - F_{p,q}(A)\right| \leq 5\epsilon F_{p,q}(A)$$

with probability $\frac{5}{8}$. Since, $\hat{F}_{p,q} = \gamma^{-1}\text{StableEst}^q(\{Z'(v)\}_{v\in[1,s_2]})$ the statement of the lemma follows. □

The Levy asymptotic result given by (1) holds for $0 < q < 2$. The case for $q = 2$ is simpler, since the distribution is the well-known Gaussian distribution with density function $f_{S^1(2,0,\gamma,\delta)}(x) = \frac{1}{\gamma\sqrt{2\pi}}e^{-(x-\delta)^2/\gamma^2}$. A standard tail inequality for Gaussian distributions implies that if $Z \sim S^1(2, 0, \gamma, \delta)$, then, $\mathsf{Pr}\{Z < 0\} < \frac{2\gamma}{\delta}e^{-\delta^2/(2\gamma^2)}$, provided, $\frac{\delta}{\gamma} = \Omega(1)$. Therefore, $\mathsf{Pr}\{Z < 0\} < \frac{1}{8n^3}$ is satisfied, if, $\delta = 1$ and $\gamma = O(\sqrt{\log n})$.

Singularities around $q = 1$ *and for* $p = 0$ *or* $q = 0$. There are two singularities that remain. First, the above method does not work for estimating $F_{p,q}$ when, either $q = 1$ or when either p or q is 0. The first case, namely, $q = 1$ is not solved using the above method since, $\tan\frac{\pi q}{2} = \infty$ and therefore, there is no solution to (3). The second problem case arises when either p or q is 0, since, stable

distributions are not known for these parameters. The solution to both these cases are obtained by approximating $F_{p,q}$ by $F_{p',q'}$, where, p' and q' are chosen to be close enough to p to q so that $|F_{p',q'} - F_{p,q}| \leq \epsilon F_{p,q}$.

Suppose $q = 1$. Then by Fact 3, $|F_{p,q'} - F_{p,q}| \leq |q - q'|F_{p,q} \log F_{p,q}$. Thus, by choosing $q' = 1 - \frac{\epsilon}{6}\log^{-1}(mn)$, where, mn is an upper bound on $F_{1,1}$, we will have $|F_{p,q'} - F_{p,1}| \leq \frac{\epsilon}{2}F_{p,1}$. Lemma 4 can now be used to estimate $F_{p,q'}$ to within a factor of $(1 \pm \frac{\epsilon}{2})$; the approximation is proved using triangle inequality. A problem of having to use very large values of γ arises because $\gamma = \Omega(x_0(q,\beta) - \tan\frac{\pi q}{2})$. If q is very close to 1, then, $x_0(q,\beta) = \Theta(\tan\frac{\pi q}{2})$ is very large. Since, $O(\log \gamma^{-1})$ bits is necessary for accurate computation, $\tan\frac{\pi q}{2}$ must be bounded. This is again solved by choosing $q' = 1 + \frac{\epsilon}{12}(\log^{-1}(mn)$, in case $q \in [1, q']$. By Fact 3, it follows that $|F_{p,q} - F_{p,q'}| \leq \frac{\epsilon}{2}F_{p,q}$. The precision required is $O(\log F_{1,1}\delta^{-1} \tan\frac{\pi q'}{2}) = O(\log(mn \cdot \frac{m}{\epsilon} \cdot \frac{\log(mn)}{\epsilon}) = O(\log\frac{mn}{\epsilon})$ bits. The case for the singularity $p = 0$ is handled similarly, since, by Fact 3, if $p' = \epsilon/(2(1 + q\log(mn)))$, $|F_{0,q} - F_{p',q}| \leq \frac{\epsilon}{2}F_{0,q}$. The case for $q = 0$ follows similarly from Fact 3.

The space requirement is more appropriately written as $O_{p,q}(\cdots)$ signifying the dependence on the parameters p and q, which are treated as constants for the sake of space complexity expressions. As analyzed by Li, the direct dependence on p, q is $O(\frac{1}{p^2q^2})$. However, for p (respectively q) equal to 0 or very close to 0, it suffices (by the arguments above) to let $p = \Omega(\frac{\epsilon}{\log F_{1,1}})$. We can now use the technique of Indyk [11] to reduce the number of random bits from n^2 bits to $O(S\log(nS))$, where, S is the space used by the algorithm assuming access to fully independent random bits. The constants in the space complexity expression are independent of p, q and n.

Theorem 1. *For $p, q \in [0, 2]$ and an a-priori upper bound m on $|A_{i,k}|$, for $1 \leq i, k \leq n$, there exists an algorithm that returns $\hat{F}$ satisfying $|\hat{F} - F_{p,q}| < \epsilon F_{p,q}$ using space $O(S\log(nS))$, where, $S = O\left(\frac{p'^2q'^2}{\epsilon^4}\left(\log\frac{q'}{\epsilon}\right)\left(\log\frac{nm}{\epsilon}\right)\right)$, $p' = \min\left(\frac{6\log(mn)}{\epsilon}, \frac{1}{p}\right)$, $q' = \min\left(\frac{6\log(mn)}{\epsilon}, \frac{1}{q}\right)$.* □

Remarks. Bi-linear stable sketches can be used to estimate $F_{p,q}$ for $p \in [0, 2]$ and $q > 2$ using space $\bar{O}(\frac{1}{\epsilon^{4+1/q}}n^{1-2/q})$ and can be found in the full version of this paper. The problem of finding space lower bounds for estimating $F_{p,q}$ is open. We conjecture that estimating $F_{0,2}$ requires $\Omega(\frac{1}{\epsilon^4})$ space.

References

1. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating frequency moments. J. Comp. Sys. and Sc. 58(1), 137–147 (1998)
2. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. In: Proceedings of ACM Symposium on Theory of Computing, Princeton, NJ, pp. 209–218 (2002)
3. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D., Trevisan, L.: Counting Distinct Elements in a Data Stream. In: Rolim, J.D.P., Vadhan, S.P. (eds.) RANDOM 2002. LNCS, vol. 2483. Springer, Heidelberg (2002)

4. Bhuvanagiri, L., Ganguly, S.: Estimating Entropy over Data Streams. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 148–159. Springer, Heidelberg (2006)
5. Chakrabarti, A., Khot, S., Sun, X.: Near-Optimal Lower Bounds on the Multi-Party Communication Complexity of Set Disjointness. In: Proceedings of 18th IEEE Conference on Computational Complexity, pp. 107–117. IEEE Computer Society, Los Alamitos (2003)
6. Cormode, G., Muthukrishnan, S.: Space Efficient Mining of Multigraph Streams. In: Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM 2005, pp. 271–282 (2005)
7. Flajolet, P., Martin, G.N.: Probabilistic Counting Algorithms for Database Applications. J. Comp. Sys. and Sc. 31(2), 182–209 (1985)
8. Ganguly, S.: Counting Distinct Items over Update Streams. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 505–514. Springer, Heidelberg (2005)
9. Ganguly, S., Cormode, G.: On Estimating Frequency Moments of Data Streams. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 479–493. Springer, Heidelberg (2007)
10. Gibbons, P.B., Tirthapura, S.: Estimating simple functions on the union of data streams. In: Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures, pp. 281–291. ACM, New York (2001)
11. Indyk, P.: Stable Distributions, Pseudo Random Generators, Embeddings and Data Stream Computation. In: Procedings of the 41st Symposium on Foundations of Computer Science (FOCS), pp. 189–197. IEEE Computer Society, Los Alamitos (2000)
12. Indyk, P., Woodruff, D.: Optimal Approximations of the Frequency Moments. In: Proceedings of the 37th ACM Symposium on Theory of Computing, 2005 (STOC), pp. 202–208 (2005)
13. Li, P.: Very Sparse Stable Random Projections, Estimators and Tail Bounds for Stable Random Projections (manuscript, 2006)
14. McGregor, A.: Open Problems In Data Streams And Related Topics: IITK Workshop On Algorithms For Data Streams (2006), http://www.cse.iitk.ac.in/users/sganguly/openproblems.pdf
15. Nolan, J.P.: Stable Distributions, Ch.1 (2006), http://academic2.american.edu/~jpnolan
16. Woodruff, D.P.: Optimal space lower bounds for all frequency moments. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 167–175. SIAM, Philadelphia (2004)

A Proofs

Proof (Of Lemma 1). $\mathsf{Pr}\{g(X) \in [u, u+du]\} = \mathsf{Pr}\left\{X \in g^{-1}[u, u+du]\right\}$. By assumption, $g^{-1}[u, u+du]$ is a finite union of disjoint maximal intervals, each of finite size $I_1, I_2, \ldots, I_w$ such that $g(I_j) = [u, u+du]$, where, $I_j = [g^{-1}(y), g^{-1}(y) + \frac{1}{g'(g^{-1}(y))}dy]$. Thus,

$$f_{g(X)}(u)du = \mathsf{Pr}\left\{X \in g^{-1}[u, u+du]\right\} = \sum_{j=1}^{w} \frac{f_X(g^{-1}(u))}{|g'(g^{-1}(u))|}du.$$

Therefore,

$$\begin{aligned}\|D_{g(X)} - D_{g(Y)}\| &= \int_{-\infty}^{\infty} |f_{g(X)}(u) - f_{g(Y)}(u)|du \\ &= \int_{-\infty}^{\infty} \frac{1}{|g'(g^{-1}(u))|} |f_X(g^{-1}(u)) - f_Y(g^{-1}(u))|du \\ &= \int_{-\infty}^{\infty} |f_X(t) - f_Y(t)|dt, \quad \text{substituting } t = g^{-1}(u) \\ &= \|D(X) - D(Y)\| \ .\end{aligned}$$

Proof (Of Lemma 2). We prove the lemma using induction. The base case $s = 1$ is given as a premise. Suppose $s = k + 1$ and assume that the statement of the lemma holds for any set of k independent and identically distributed variables.

$$\begin{aligned}&\|D(X_1, X_2, \ldots, X_{k+1}) - D(Y_1, Y_2, \ldots, Y_{k+1})\| \\ &= \int_{x_1=-\infty}^{\infty} \int_{x_2,\ldots,x_{k+1}=-\infty}^{\infty,\ldots,\infty} |f_{X_1}(x_1) f_{X_2}(x_2) \ldots f_{X_{k+1}}(x_{k+1}) - \\ &\qquad\qquad f_{Y_1}(x_1) f_{Y_2}(x_2) \ldots f_{Y_{k+1}}(x_{k+1})| dx_1 \ldots dx_{k+1} \\ &= \int_{x_1=-\infty}^{\infty} \int_{x_2,\ldots,x_{k+1}=-\infty}^{\infty,\ldots,\infty} dx_1 \ldots d_{x_{k+1}} \big| f_{X_1}(x_1) f_{X_2}(x_2) \ldots f_{X_{k+1}}(x_{k+1}) - \\ &\qquad\qquad f_{X_1}(x_1) f_{Y_2}(x_2) \ldots f_{Y_{k+1}}(x_{k+1}) + \\ &\qquad f_{X_1}(x_1) f_{Y_2}(x_2) \ldots f_{Y_{k+1}}(x_{k+1}) - f_{Y_1}(x_1) f_{Y_2}(x_2) \ldots f_{Y_{k+1}}(x_{k+1}) \big| \\ &\leq \int_{x_1=-\infty}^{\infty} |f_{X_1}(x_1)| dx_1 \int_{x_2,\ldots,x_{k+1}=-\infty}^{\infty,\ldots,\infty} |f_{X_2}(x_2) \ldots f_{X_{k+1}}(x_{k+1}) - \\ &\qquad\qquad f_{Y_2}(x_2) \ldots f_{Y_{k+1}}(x_{k+1})| dx_2 \ldots d_{x_{k+1}} \\ &+ \int_{x_1=-\infty}^{\infty} |f_{X_1}(x_1) - f_{Y_1}(x_1)| dx_1 \int_{x_2,\ldots,x_{k+1}=-\infty}^{\infty,\ldots,\infty} |f_{Y_2}(x_2) \ldots f_{Y_{k+1}}(x_{k+1})| dx_2 \ldots d_{x_{k+1}} \\ &= \|D(X_2, \ldots, X_{k+1}) - D(Y_2, \ldots, Y_{k+1})\| + \|D(X_1) - D(Y_1)\| \leq k\delta + \delta = (k+1)\delta \ .\end{aligned}$$

The simplification of the integral follows from independence and since, $|f_X(x)| = f_X(x)$ for any probability density function f_X, therefore,

$$\int_{x_j=-\infty}^{\infty} |f_{X_j}(x_j)| dx_j = \int_{x_j=-\infty}^{\infty} f_{X_j}(x_j) dx_j = 1 \ . \qquad \square$$

Constraint Bipartite Vertex Cover
Simpler Exact Algorithms and Implementations

Guoqiang Bai and Henning Fernau

Univ. Trier, FB IV—Abteilung Informatik, 54286 Trier, Germany
baiguoqiang@hotmail.com, fernau@uni-trier.de

Abstract. CONSTRAINT BIPARTITE VERTEX COVER is a graph-theoretical formalization of the spare allocation problem for reconfigurable arrays. We report on an implementation of a parameterized algorithm for this problem. This has led to considerable simplifications of the published, quite sophisticated algorithm. Moreover, we can prove that the mentioned algorithm could be quite efficient in practial situations.

1 Introduction

Problem Definition. In this paper, we are considering the following problem:

An instance of CONSTRAINT BIPARTITE VERTEX COVER (CBVC) is given by a bipartite graph $G = (V_1, V_2, E)$, and the parameter(s), positive integers k_1, k_2. The task is: Is there a *vertex cover* $C \subseteq V_1 \cup V_2$ with $|C \cap V_i| \leq k_i$ for $i = 1, 2$?

This graph-theoretic problem can be easily seen to be equivalent to the following problem, namely via the adjacency matrix of a bipartite graph:

An instance of SPARE ALLOCATION (SAP) is given by a $n \times m$ binary matrix A representing an erroneous chip with $A[r, c] = 1$ iff the chip is faulty at position $[r, c]$, and the parameter(s), positive integers k_1, k_2. The task is: Is there a *reconfiguration strategy* that repairs all faults and uses at most k_1 spare rows and at most k_2 spare columns?

Motivation and Previous Work. Kuo and Fuchs [13] provide a fundamental study of that problem. Put concisely, this "most widely used approach to reconfigurable VLSI" uses spare rows and columns to tolerate failures in rectangular arrays of identical computational elements, which may be as simple as memory cells or as complex as processor units. If a faulty cell is detected, the entire row or column is replaced by a spare one. The tacit (but unrealistic) assumption that spare rows and columns are never faulty can be easily circumvented by a parameter-preserving reduction, as exhibited by Handa and Haruki [11]. For technological reasons (avoiding superfluous redundancy [12] as well as too much expensive laser repair surgery), the number of spare rows and columns is very limited (rarely more than fourty), making this problem a natural candidate for a "fixed parameter approach" [5].

However, due to the $\mathcal{NP}$-hardness of the problem (shown in [13]), no polytime algorithms can be expected. In [9], it is shown that CONSTRAINT BIPARTITE

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 67–78, 2008.

VERTEX COVER can be solved slightly faster than $\mathcal{O}^*(1.4^k)$. However, that particular algorithm is derived via a very sophisticated analysis of local situations, quite typical for search-tree algorithms that were developed ten years ago. This means that a naive implementation would have to test all these local cases, which is quite a challenging and error-prone task by the sheer number of cases (more than 30). Since many of these cases are quite special, these cases would show up rarely, and therefore programming errors would be hard to detect. Our approach presented here means nearly a complete re-design of the published algorithm: as it is often the case now with "modern" exact algorithms, most of the burden is taken from the algorithm implementor and shifted on the shoulders of the algorithm analyzer.

Contributions. Our main contribution is to present within Sec. 2 implementable algorithm variants of the algorithm given in [9]; to the most refined variant, basically the same run time analysis applies. These variants have been implemented and tested, which will be described in Sec. 3. We observe that in practical instances (generated according to previously described schemes), the algorithms perform much better than it could be expected from theory. This shows that exact algorithms could be useful even in circumstances where real-time performance is important, as it is the case in industrial processes.

Notations. We need some non-standard notation: If $S = (C_1, C_2)$ is a CBVC solution of $G = (V_1, V_2, E)$, then $(|C_1|, |C_2|)$ is called the *signature* $\sigma(S)$ of that solution. We call a solution (C_1, C_2) *signature-minimal* if there is no solution S' with $\sigma(S') < \sigma(S)$; we will also call $\sigma(S)$ a minimal signature in this case. Here, we compare two vectors of numbers componentwisely. The *signature spectrum* of a CBVC instance (G, k_1, k_2) is the set of all minimal signatures (i_1, i_2) (that obey $(i_1, i_2) \leq (k_1, k_2)$). If the parameter is not given, we ignore the latter restriction. As can be seen, all our algorithms (written as decision algorithms for convenience) can be easily converted into algorithms that produce (solutions to) all minimal signatures.

2 Algorithm Variants

We are going to describe three algorithm variants that will be compared later in Sec. 3.

2.1 The Simplest Algorithm A1

It should be noted here that people developing algorithms for VLSI design actually discovered the $\mathcal{FPT}$ concept in the analysis of their algorithms, coming up with $\mathcal{O}(2^{k_1+k_2}k_1k_2 + (k_1+k_2)|G|)$ algorithms in [10,15]. They observed that "if k_1 and k_2 are small, for instance $\mathcal{O}(\log(|G|))$, then this may be adequate." [10, p. 157]. It was in the context of this problem that Evans [6] basically discovered *Buss' rule* to prove a problem kernel for CONSTRAINT BIPARTITE VERTEX COVER. Kuo and Fuchs called this step quite illustratively the *must-repair-analysis*: Whenever a

row contains more than k_2 faulty elements (or a column contains more than k_1 faulty elements, resp.), then that row (column, resp.) must be exchanged.

Lemma 1. constraint bipartite vertex cover *has a problem kernel of size* $2k_1k_2$.

The search-tree part (leading to the $\mathcal{O}^*(2^{k_1+k_2})$ is also easy to explain: any edge must be covered (i.e., any failure must be repaired), and there are two ways to do it. This leads to the simplest algorithm variant A1.

Heuristic improvements. From a heuristic viewpoint, it is always good to branch at vertices of high degree (not just at edges) for vertex cover problems, since in the branch when that vertex is not taken into the cover, all its neighbors must go into the cover. A further very important technique is *early abort*. Namely, observe that for the minimum vertex cover C^* of $G = (V_1, V_2, E)$, we have $|C^*| \leq k_1 + k_2$ for any constraint bipartite cover C with $\sigma(C) \leq (k_1, k_2)$. Since those *overall minimum vertex cover*s can be computed in polynomial time using matching techniques (on bipartite graphs), we can stop computing the search-tree whenever the remaining current parameter budget (k_1, k_2) has dropped (in sum) below the overall minimum vertex cover cardinality. Notice that sometimes also the variant that *requires* a minimum vertex cover as constraint vertex cover is considered, e.g., in [4,18] from a parameterized complexity. However, it is not quite clear from a practical perspective why one should insist on overall cover minimality: to the contrary, repairable arrays should be repaired, irrespectively of whether they yield an overall minimum cover or not.

2.2 Triviality Last: Algorithm A2

There are two simple strategies to improve on quite simplistic search-tree algorithms: either (1) there are simple reduction rules that allow one to deduce that (in our case) branching at high-degree vertices is always possible or (2) one can simply avoid "bad branches" by deferring those branches to a later phase of the algorithm that can be performed in polynomial time. We have called the first strategy *triviality first* and the second one *triviality last* in [8]. Notice that both strategies can be used within a mathematical run time analysis for the search tree heuristic sketched in the previous subsection.

We will explain how to employ the triviality last principle based on the following observations.

Lemma 2. *Let $G = (V_1, V_2, E)$ be a connected undirected bipartite graph with maximum vertex degree 2 and let $\ell = |E|$ be the number of edges in G.*

1. *If G is a cycle, then for $\ell' := \ell/2$ we have the minimal signatures $(0, \ell')$, $(\ell', 0)$ as well as $(2, \ell' - 1), (3, \ell' - 2), \ldots, (\ell' - 1, 2)$ if $\ell > 4$.*
2. *Let G be a path.*
   (a) *If ℓ is odd, then for $\ell' := (\ell + 1)/2$ we have the minimal signatures $(0, \ell'), (1, \ell' - 1), \ldots, (\ell' - 1, 1), (\ell', 0)$.*

(b) *If ℓ is even, then for $\ell' := \ell/2 + 1$ we have the minimal signatures $(0, \ell' - 1), (2, \ell' - 2), \ldots, (\ell' - 1, 1), (\ell', 0)$ if $|V_1| > |V_2|$ and $(0, \ell'), (1, \ell' - 1), \ldots, (\ell' - 2, 2), (\ell' - 1, 0)$ if $|V_1| < |V_2|$.*

Lemma 3. *If the signature spectra of all components of a graph are known, then the signature spectrum of the overall graph can be computed in polynomial time.*

Algorithm 1. CBVC-TL: a still simple search tree algorithm for CBVC

Input(s): a bipartite graph $G = (V_1, V_2; E)$, positive integers k_1 and k_2
Output(s): YES iff there is a vertex cover $(C_1, C_2) \subseteq V_1 \times V_2$, $|C_1| \leq k_1$ and $|C_2| \leq k_2$

```
if k1 + k2 ≤ 0 and E ≠ ∅ then
  return NO
else if k1 + k2 ≥ 0 and E = ∅ then
  return YES
else if possible then
  Choose vertex x ∈ V1 ∪ V2 such that deg(x) ≥ 3.
  if x ∈ V1 then
    d = (1, 0)
  else
    d = (0, 1)
  if CBVC-TL(G − x, (k1, k2) − d) then
    return YES
  else
    return CBVC-TL(G − N(x), (k1, k2) − deg(x)((1, 1) − d))
else
  {vertex selection not possible ⇝ maximum degree is 2}
  resolve deterministically according to Lemma 4
```

Proof. The most straight-forward way to see this is via dynamic programming: If (k_1, k_2) is the parameter bound, then use a $k_1 \times k_2$ table that is originally filled by zeros, except the entry at $(0, 0)$ that contains a one. Let c loop from 1 to the number of components plus one. In that loop, for each entry (i, j) of that table that equals c and for each element (r, s) of the signature spectrum, we write $c + 1$ as entry into place $(i + r, j + s)$. Finally, the signature spectrum of the overall graph can be read off as those (i, j) whose table entry contains the number of components plus one. This procedure can be further sped up by noting that there could be at most $k_1 + k_2 + 1$ minimal signatures for any graph; so with some additional bookkeeping one can avoid looping through the whole table of size $k_1 \times k_2$. ∎

Lemma 4. CONSTRAINT BIPARTITE VERTEX COVER *can be solved in time $\mathcal{O}(k \log k)$ on forests of cycle and path components with budget k_1, k_2 (where $k = k_1 + k_2$), i.e., on graphs of maximum degree of two.*

We want to point out that Lemma 4 could be seen by a more efficient algorithm than indicated in the proof of Lemma 3. The strategy is the following one:

—First, solve paths of even length.
—Secondly, solve cycles.
—Finally, solve paths of odd length.

Within each of these three categories of components of a graph of maximum degree two, we basically solve small components first (except for cycles as discussed below).

The intuition behind this strategy is that solving paths of even length by an overall minimum cover is the most challenging task, while it is close to trivial for paths of odd length.

- (A) Namely, from Lemma 2, we can easily deduce that a cover is signature-minimal iff it is overall minimum in the case of paths of odd length. Therefore, we can defer the selection of the specific cover vertices to the very end, and this decision could be then taken in a greedy fashion.
- (B) The only difficulty that can show up with covering cycles is that both parameter budgets k_1 and k_2 might be smaller than the length of that cycle. So, when left with cycle components $C_1,\ldots,C_r$ (of length $\ell_1,\ldots,\ell_r$) and paths of odd length, we first test if there are $x_i \in \{0,1\}$ with $\sum_{i=1}^{r} x_i\ell_i \in \{k_1,k_2\}$. If $\sum_{i=1}^{r} x_i\ell_i = k_j$ for some $x_i \in \{0,1\}$, we solve those C_i with $x_i = 1$ by covering them with vertices from V_j. More generally, if $\sum_{i=1}^{r} x_i\ell_i \leq k_1$ and $\sum_{i=1}^{r}(1-x_i)\ell_i \leq k_2$ for some $x_i \in \{0,1\}$, we solve those C_i with $x_i = 1$ by covering them with vertices from V_1, and the remaining cycles by vertices from V_2. If such x_i cannot be found but the overall minimum for solving the cycle components is less than $k_1 + k_2$, our general greedy strategy (working from small length cycles onwards) will produce one cycle (and only one) that is not solved matching the overall minimum cover, by covering it both with vertices from V_1 and with vertices from V_2.
- (C) The greedy strategy used at the beginning on paths of even length might also lead to a point that some path cannot be solved matching the overall minimum. In that particular case (*), we will deliberately first empty the critical parameter budget. Similar to (A), one can see that this choice is arbitrary for that particular component: any other feasible minimal choice would have served alike. However, since we are working from smaller to larger components, we will never later see a path of even length that cannot be solved to optimum for the reason that we might have "stolen" the optimum solution through step (*). Moreover, the preference of solving small components first is justified by the fact that this way, the smallest distance from the overall minimum is guaranteed. A further justification for the choice of (*) is that this guarantees that (later on) all cycle components will be solved matching the overall minimum.

2.3 More Sophistication: Algorithm A3

The main part of [9] was struggling with improving the branching on vertices up to degree three. Here, we are going to display a much simpler branching

1. If possible: branch at a vertex of degree four or higher.
2. If the graph is polynomially solvable: do so and terminate.
3. if the graph is 3-regular: branch at an arbitrary vertex of degree three.
// In the following, let c be a component of the graph that is not polynomially solvable and that is not 3-regular.
4. Let A be the vertex in c with the largest number of attached tails.
4a. If A has three tails, then we branch at A. (Notice that c is no S_3.)
4b. If A has only two tails, then consider the neighbor D of A that is not on a tail. Let E be the first vertex of degree three on the path p starting with A, D (E must exist, since p is no tail) and branch at E, with possibly $E = D$.
4c. If A has only one tail p, then branch at A if p is not a microtail.
// The only tails in c are microtails attached to vertices A with no other tail.
4d. Let A be a vertex with microtail. Let $B \neq A$ be one of the two vertices of degree three that can be reached from A (with possible intermediate vertices of degree two), preferring the closer one, ties broken arbitrarily. Then, branch at B.
// This way, all microtails are deleted from c.
// Let A be a vertex of degree two, with the largest number of neighbors that also have degree two. Let B and C be the two vertices of degree three that can be reached from A in either direction, s.t. B is not farther away from A than C.
5. If C is not neighbor of A, then we branch at B.
6. // Now, the vertices B and C of degree three are neighbors of the degree-2-vertex A.
6a. If B and C have three common neighbors A, D, E, either take B, C or A, D, E together into the cover.
6b. If B and C have two common neighbors A, D of degree two, then branch at B.
6c. If B and C have two common neighbors A, D, with D of degree three, then branch at D.
7. // Now, the degree-3-vertices B and C have only one common neighbor, namely the degree-2-vertex A.
7a. If C has three neighbors of degree two, branch at B, and vice versa.
7b. If B or C has exactly one neighbor E of degree three, branch at E.
8. // Now, B and C possess only one neighbor of degree two, namely A.
// Let $N(B) = \{B_1, B_2, A\}$ and $N(C) = \{C_1, C_2, A\}$.
8a. If $(N(B_1) \cap N(B_2)) = \{B, B'\}$, then branch by either taking B and B' into the cover or taking B_1, B_2 into the cover. The case $|N(C_1) \cap N(C_2)| > 1$ is symmetric.
8b. If $N(B_1) = \{B, B', B''\}$ with $\deg(B'') = 2$, then branch at B'. (There are three possible symmetric cases to be considered.)
8c. Branch at some $X \in (N(B_1) \cup N(B_2) \cup N(C_1) \cup N(C_2)) \setminus \{B, C\}$.

Fig. 1. List of heuristic priorities

strategy (that was actually implemented) to obtain basically the same run time estimate improvements. We describe the adopted branching strategy in what follows. Here, S_3 denotes a star graph with four vertices, one center connected to the three other vertices (and these are all edges of the graph). Notice that we can easily adapt our polynomial-time algorithms described above to cope with S_3-components, as well. So, we term a graph that contains only components of maximum degree two or S_3-components *polynomially solvable*. We also need two further notions from [9]: a *tail* consists of a degree-3-vertex A, followed by a (possibly empty) sequence of degree-two-vertices, ended by a degree-1-vertex. If

case	branching vector	branching number
1.	$(1, 4)$	1.3803
2.	—	no branching
3.	$(1, 3)$	happens only once per search tree path
$4a. - 4c.$	$(2, 3)$	1.3248
$4d. - 5.$	$(3, 3, 4)$	1.3954
$6a.$	$(2, 3)$	1.3248
$6b.$	$(3, 3, 4)$	1.3954
$6c.; 7b.$	$(3, 4, 6, 6, 7)$	1.3954
$7a.$	$(3, 4, 6, 6, 8)$	1.3867
$8a.$	$(2, 4, 5)$	1.3803
$8b.$	$(4, 5, 7, 9, 7, 6, 7, 9)$	1.3905
$8c.$	$(4, 5, 7, 7, 8, 6, 6, 7)$	1.4154

Fig. 2. Branching vectors and numbers for different heuristic priorities

A is neighbor of a degree-1-vertex, we speak of a *micro-tail*. In Fig. 1, we give a list of priorities according to which branching should be done.

Notice that there are further variations of the algorithm that are easily at hand. For example, one can observe that step 4a. can be avoided, since then the component c is a tree for which a list of all minimal signatures can be obtained by dynamic programming. However, this does not affect our worst-case running time analysis that is sketched in the following.

Some Further Comments on the Run-Time Analysis

4d: Recall that there is a micro-tail A' attached to A. The worst case is when $B \in N(A)$, and when $C \in N(A)$ is also of degree three. If B is taken into the cover, then in the next recursive call of the procedure, in the worst case C would be selected for branching, since a (non micro-)tail is attached to C, giving it the highest heuristic priority. If C is put into the cover, then the edge $A'A$ must be covered by an additional vertex. Hence, we obtain the claimed branching vector.

5: In the worst case, $B \in N(A)$ and $C \in N(N(A))$. Since the graph is bipartite, $C \neq B$. If B goes into the cover, then we obtain a (non micro-)tail at C as in case 4d. The case when $B = C$, i.e., B and C are (at least) at distance four, is even better, yielding a branching vector of at least $(2, 3)$.

6: Here, we consider the remaining cases that the degree-2-vertex A is part of a 4-cycle. Notice that the analysis of small cycles was one of the cornerstones of the analysis in [9].

6a: If one out of A, D, E is not going into the cover, then B and C must go there. Conversely, if B or C does not go into the cover, then all of A, D, E are there.

6b: If B goes into the cover, we produce a situation with C having two microtails (case 4b).

6c: There are three subcases to be considered for the run-time analysis, depending on $j = |N(N(C) \setminus \{A, D\}) \cap N(N(B) \setminus \{A, D\})| \in \{0, 1, 2\}$. Notice that $N(B) \cap N(C) = \{A, D\}$, since otherwise case 6 would have applied. Details of the tedious but straight-forward analysis are omitted for reasons of space.

7: Observe that now $|(N(B) \cup N(C)) \setminus \{A\}| = 4$; otherwise, A is part of a 4-cycle or B and C do not have degree three; all these cases were treated above.
7a: If B is taken into the cover, we find a microtail at C. Since both neighbors of C are of degree two, we will branch at their neighbors C_1 and C_2 (not equal to C) at worst. Due to case 6b, we can assume that $N(C_1) \cap N(C_2) = \emptyset$. Taking the branches at C_1 and C_2 alone, we arrive at a branching vector of $(3, 5, 5, 7)$ in the case when B goes into the cover. Altogether, we get a branching vector of $(3, 4, 6, 6, 8)$ which yields a branching number of 1.3867.
7b: If E goes into the cover, we produce a chain of three consecutive degree-2 vertices. Assuming that we have no cycles of length four or six, such a chain can be resolved through a branching vector of $(3, 5, 5, 6)$. Altogether, we arrive at a branching vector of $(3, 4, 6, 6, 7)$, yielding a branching number of 1.3956.
8a: Notice that A, B form a tail after branching at B_1 and B_2. So, the analysis from [9, Table 15] applies.
8b: If B' is taken into the cover, this will be followed by branching at the hitherto unnamed neighbor $\hat{B}$ of B'' by priority 5. We consider the cases that the neighbors of B' or B' and the neighbors of $\hat{B}$ go into the cover. In both cases, we have the possibility to isolate small components (by observing tails) with branching at C and at B_2. Overall, this gives a branching vector of $(4, 5, 7, 9, 7, 6, 7, 9)$ and hence the claimed branching number.
8c: Consider $B' \in N(B_1)$ selected for branching. If B' goes into the cover, then B is neighbor of two vertices of degree two, so that B_2 will be selected for branching, following the analysis of case 7b. If $N(B')$ is put into the cover, A and B are two neighbored vertices of degree two; so, priority 5 applies. Combining the branching vectors yields the claim. Notice that this is by far the worst case branching. The analysis from [9] shows that this particular case can be improved, basically by consequently branching at all vertices from $(N(B_1) \cup N(B_2) \cup N(C_1) \cup N(C_2)) \setminus \{B, C\}$ in parallel. To actually and fully mimic the case distinctions from [9], cycles of length 6 should be treated in a separate way. However, as it turned out, this case (in fact, all subcases of case 8) showed up very rarely in the experiments, so that we could safely omit it. More precisely, less than 0.001% of all branches were due to these "isolated vertices of degree two." So, our implementation is deliberately omitting some of the details from [9] without sacrificing speed in practice. However, it would need only three special cases to be implemented to completely cover case 8c. according to the analysis from [9], namely those depicted in Tables 6, 22 and 23 in [9].

3 The Tests

Blough [2,3] discussed how to model failures in memory arrays. He suggested the so-called center-satellite model, based on earlier work of Meyer and Pradhan [16]. In that model, it is assumed that memory cells could spontaneously and independently fail with a certain probability p_1. However, once a cell failed, its neighbors also fail with a certain probability p_2, and this affects a whole region of radius r around a central element. So, we are dealing with compound

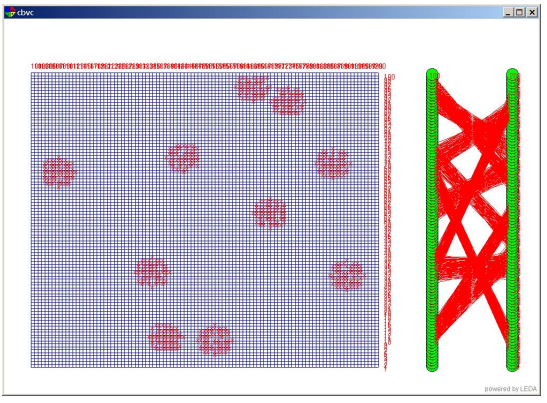


Fig. 3. The graphical interface of the implementation; to the right, the bipartite graph model is shown

probabilities. This leads to a two-stage model to simulate such kind of failures: in a first phase, with probability p_1, memory cells are assumed to be faulty; in a second phase, within a neighborhood of radius r, cells are assumed to be faulty with probability p_2. Fig. 3 shows a failure pattern obtained in this way.

Notice that we assume both underlying distributions to be uniform. This differs a bit from the original model of Meyer and Pradhan [16] who assumed a Poisson distribution for the satellites; however, since we are focussing on rather small radii in our experimental studies, this distinction is not essential. Also Blough and Pelc adopted this simpler approach. However, we deviate from the approach of Blough and Pelc insofar as we are considering cycles around the center with the usual meaning, while they considered cycles in the Manhattan metric, i.e., square-shaped failure regions.

All tests were run on a PC with Athlon XP 2000+ processor (2 GHz with 512 KB Cache) and with 256 MB main memory. We used Microsoft Windows XP Professional, Visual Studio 2003 and LEDA 3.0. All the running times were obtained by testing 100 independently generated instances within a given setup as described in the first columns of the tables, describing the dimensions of the array, the number of spare lines, the chosen probabilities and the chosen radius r (for the satellites). The run times are given in seconds per instance (on average). Fig. 4 tries to simulate the settings described by Blough and Pelc in [3]. We list the measurements separately for the case that a solution was found (or not).

At first glance, the corresponding figures seem to be surprising, since the running times for finding a solution are consistently larger than the ones for rejecting the instance. This phenomenon is partly explained when looking at the average number of branchings (# br) encountered. Obviously, the early abort method (based on maximum matching) allows to early reject most instances nearly without branching in polynomial time. We validated this hypothesis by testing our algorithms without that heuristic, which led to a tremendous slow-down, in particular in the case when no solution was to be found. We also separately list the

m=n	k_1=k_2	p_1	p_2	r	success (%)	without solution				with solution			
						Alg 2	# br	Alg 3	# br	Alg 2	# br	Alg 3	# br
1024	32	0.000007	0.8	5	29	0.1003	12	0.1225	12	0.4440	99	0.4416	98
1024	32	0.000004	0.8	9	11	0.1341	6	0.1337	6	0.2184	32	0.2212	32
1024	32	0.000002	0.8	15	100	----------	---	----------	---	0.0663	3	0.0660	3
1024	36	0.000003	0.8	15	8	0.0402	0	0.0407	0	0.0475	0	0.0488	0
1024	36	0.000005	0.7	9	6	0.0445	1	0.0457	1	0.3705	37	0.3255	37
2048	64	0.000003	0.7	7	3	0.0441	0	0.0425	0	5.2006	574	5.0236	573
2048	64	0.000002	0.5	9	10	0.0345	0	0.0322	0	12.6780	1875	10.6960	1875
4096	128	0.000001	0.7	7	30	0.0714	0	0.0728	0	11.2197	576	11.216	574

Fig. 4. The run times of Blough's setup

m=n	k_1=k_2	p_1	p_2	success (%)	without solution				with solution			
					Alg 2	# br.	Alg 3	# br	Alg 2	# br.	Alg 3	# br
4096	32	0.0000025	0.5	39	0.0213	0	0.0210	0	0.0562	13	0.0526	11
512	64	0.00037	0.5	59	0.0144	0	0.0126	0	0.1247	35	0.1168	30
1024	64	0.000085	0.5	47	0.0181	0	0.0209	0	0.1746	33	0.1838	29
2048	64	0.00002	0.5	78	0.0231	0	0.0254	0	0.1934	33	0.1823	27
4096	64	0.000005	0.5	48	0.0344	0	0.0348	0	0.2010	33	0.1888	24
1024	128	0.00018	0.5	40	0.0235	0	0.0233	0	0.3480	52	0.3020	45
2048	128	0.000042	0.5	52	0.03652	0	0.03502	0	0.7441	78	0.6911	64
4096	128	0.00001	0.5	66	0.0642	0	0.0530	0	0.7803	73	0.7530	57
6144	128	0.0000046	0.5	18	0.0774	0	0.0767	0	0.8389	76	0.8056	54
8192	128	0.0000027	0.4	53	0.0566	0	0.0594	0	0.4752	56	0.3816	36
2048	256	0.0001	0.5	90	0.0600	0	0.0500	0	2.3103	160	1.9757	132
4096	256	0.000021	0.5	80	0.0550	0	0.0550	0	2.6198	154	2.0930	126
6144	256	0.000009	0.5	78	0.0733	0	0.0770	0	2.3590	161	1.8155	121
8192	256	0.0000055	0.4	79	0.0800	0	0.0768	0	1.8086	119	1.4749	82
4096	512	0.000052	0.4	20	0.1101	0	0.1188	0	10.115	335	8.2220	267
6144	512	0.000022	0.4	20	0.1391	0	0.1452	0	9.6835	296	7.7065	224
8192	512	0.0000117	0.4	60	0.1277	0	0.1227	0	8.4538	274	6.6630	205
9216	512	0.0000092	0.4	20	0.1490	0	0.1440	0	8.0715	263	6.7795	198
10240	512	0.0000074	0.4	70	0.1436	0	0.1466	0	8.0216	255	6.0901	183

Fig. 5. The run times for larger numbers

running times for the (quite sophisticated) Alg. 3 in comparison with the much simpler Alg. 2. In practice, it does not necessarily pay off to invest much more time in implementing the more complicated algorithm.

In Fig. 5, we take a more radical approach: while we always restricted the radius of the defective region to one, we explored quite large parameter values. Even with values as high as $k_1 = k_2 = 512$, which gives an astronomic constant of about 2^{500} in our run-time worst case estimates, we still obtain (non-)solutions in only a few seconds. In this sense, our algorithm displays quite a robust behaviour.

It is also seen that the savings of branches of the more complicated Alg.3 are more visible in more complicated instances.

In fact, we also tested on usual random graph models and found similar observations as those reported for Blough's model, see [1]. In another set of experiments (again explicitly reported in [1]), we observed that good run times seem to depend on the fact that, in the experiments displayed so far, $k_1 = k_2$: whenever k_1 and k_2 are much different, the early abort method strikes only occasionally, and therefore the running times do considerably increase.

We can read our experimental results also as a validation of earlier probability-theoretic results indicating that it is unlikely to find hard instances for CBVC under various probability models, including the center-satellite model described above [3,17]. This also explains why the probabilities p_1 and p_2 listed in our figures look quite special: indeed they were searched for in order to display any non-trivial behaviour of our algorithms.

4 Conclusions: Lessons Learned and Future Work

We have shown that $\mathcal{FPT}$ methodology is quite useful at various stages when designing algorithms for $\mathcal{NP}$-hard problems:

- One could validate heuristics against known optimal solutions; this sort of application is useful even when the exact algorithm turns out to be too slow for actual applications within the industrial process at hand;
- At least in some circumstances, one could actually use those algorithms, even when dealing with a (pipelined, but still real-time) industrial application (as in chip manufacturing processes in our concrete application), at least with an additional time limit that might sometimes stop the search tree and output the current (sub-optimum) solution.

In particular, modern exact algorithms are very neat to implement since their overall structure tends to be quite simple: in our case, we could first implement Algorithm A1 as search tree backbone. A1 can be safely implemented in two weeks by one programmer. The integration of the polynomial phase, leading to A2, may take another week including validation. The details of the heuristic priority list may take more time than previously invested, but tests could and should always accompany this phase to see if the special cases considered with those priorities will actually occur. This consideration brought us to the decision to omit some of the special cases (that should have been implemented to fully match the analysis given in [9]), because those cases will not show up very often.

It remains as future work to compare our approach with existing published heuristics and also with some other exact approaches, a recent example being [14]. Even more interesting would be to test our algorithms on "real data", not only on simulated data. Since this particular problem is connected with many production secrets, these real data are not available.

Furthermore, there are alternative yield enhancement strategies employed in modern chip fabrication processes, as sketched in [7]. As detailed in [1], some

of these enhancements can actually be solved with (a slight variant of) the algorithms described in this paper. However, others seem to be more costly within the parameterized algorithm framework. So, the development of efficient parameterized algorithms for these variants is a further challenge for the future.

References

1. Bai, G.: Ein eingeschränktes Knotenüberdeckungsproblem in bipartiten Graphen. Diplomarbeit, FB IV, Informatik, Universität Trier, Germany (2007)
2. Blough, D.M.: On the reconfiguration of memory arrays containing clustered faults. In: Fault Tolerant Computing, pp. 444–451. IEEE Press, Los Alamitos (1991)
3. Blough, D.M., Pelc, A.: A clustered failure model for the memory array reconfiguration problem. IEEE Transactions on Computers 42(5), 518–528 (1993)
4. Chen, J., Kanj, I.A.: Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithmics. Journal of Computer and System Sciences 67, 833–847 (2003)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
6. Evans, R.C.: Testing repairable RAMs and mostly good memories. In: Proceedings of the IEEE Int'l Test Conf., pp. 49–55 (1981)
7. Fernau, H.: On parameterized enumeration. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2383, pp. 564–573. Springer, Heidelberg (2002)
8. Fernau, H.: Parameterized Algorithmics: A Graph-Theoretic Approach, Habilitationsschrift, Universität Tübingen, Germany (2005)
9. Fernau, H., Niedermeier, R.: An efficient exact algorithm for constraint bipartite vertex cover. Journal of Algorithms 38(2), 374–410 (2001)
10. Haddad, R.W., Dahbura, A.T., Sharma, A.B.: Increased throughput for the testing and repair of RAMs with redundancy. IEEE Transactions on Computers 40(2), 154–166 (1991)
11. Handa, K., Haruki, K.: A reconfiguration algorithm for memory arrays containing faulty spares. IEICE Trans. Fundamentals E83-A(6), 1123–1130 (2000)
12. Koren, I., Pradhan, D.K.: Modeling the effect of redundancy on yield and performance of VLSI systems. IEEE Transactions on Computers 36(3), 344–355 (1987)
13. Kuo, S.-Y., Fuchs, W.K.: Efficient spare allocation for reconfigurable arrays. IEEE Design and Test 4, 24–31 (1987)
14. Lin, H.-Y., Yeh, F.-M., Kuo, S.-Y.: An efficient algorithm for spare allocation problems. IEEE Transactions on Reliability 55(2), 369–378 (2006)
15. Lombardi, F., Huang, W.K.: Approaches to the repair of VLSI/WSI PRAMs by row/column deletion. In: International Symposium on Fault-Tolerant Computing (FTCS 1988), pp. 342–347. IEEE Press, Los Alamitos (1988)
16. Meyer, F.J., Pradhan, D.K.: Modeling defect spatial distribution. IEEE Transactions on Computers 38(4), 538–546 (1989)
17. Shi, W., Fuchs, W.K.: Probabilistic analysis and algorithms for reconfiguration of memory arrays. IEEE Transactions on Computer-Aided Design 11(9), 1153–1160 (1992)
18. Wang, J., Xu, X., Liu, Y.: An Exact Algorithm Based on Chain Implication for the Min-CVCB Problem. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) COCOA. LNCS, vol. 4616, pp. 343–353. Springer, Heidelberg (2007)

NP-Completeness of (k-SAT,r-UNk-SAT) and ($\mathbf{LSAT}_{\geq k}$,r-$\mathbf{UNLSAT}_{\geq k}$)*

Tianyan Deng[1] and Daoyun Xu[2]

[1] Department of Computer Science, Guizhou University(550025), Guiyang; Guangxi Teachers Education University(530001), Nanning, P.R.China
dengty@gxtc.edu.cn
[2] Department of Computer Science, Guizhou University(550025), Guiyang, P.R.China
dyxu@gzu.edu.cn

Abstract. k-CNF is the class of CNF formulas in which the length of each clause of every formula is k. The decision problem asks for an assignment of truth values to the variables that satisfies all the clauses of a given CNF formula. k-SAT problem is k-CNF decision problem. Cook [9] has shown that k-SAT is NP-complete for $k \geq 3$. $LCNF$ is the class of linear formulas and $LSAT$ is its decision problem. In [3] we present a general method to construct linear minimal unsatisfiable (MU) formulas. $NP = PCP(log, 1)$ is called PCP theorem, and it is equivalent to that there exists some $r > 1$ such that (3SAT, r-UN3SAT)(or denoted by $(1 - \frac{1}{r}) - GAP3SAT$) is NP-complete [1] [2]. In this paper,we show that for $k \geq 3$, (kSAT, r-UNkSAT) is NP-completre and (LSAT, r-UNLSAT) is NP-completre for some $r > 1$. Based on the application of linear MU formulas[3], we construct a reduction from (4SAT, r-UN4SAT) to ($LSAT_{\geq 4}$,r'-$UNLSAT_{\geq 4}$), and proved that ($LSAT_{\geq 4}$,$r - UNLSAT_{\geq 4}$) is NP-complete for some $r > 1$, so the approximation problem s-$Approx$-$LSAT_{\geq 4}$ is NP-hard for some $s > 1$.

Keywords: PCP theorem, linear CNF formula, LSAT, NP-completeness, reduction, minimal unsatisfiable(MU) formula.

1 Introduction

A clause C is a disjunction of literals, $C = (L_1 \vee \cdots \vee L_m)$, or denoted by a set $\{L_1, \cdots, L_m\}$. A formula F in conjunctive normal form (CNF) is a conjunction of clauses, $F = (C_1 \wedge \cdots \wedge C_n)$, or denoted by a set $\{C_1, \cdots, C_n\}$. $var(F)$ is the set of variables occurring in the formula F and $var(C)$ is the set of the variables in the clause C. We denote $\#cl(F)$ as the number of clauses of F and $\#var(F)$ (or $|var(F)|$) as the number of variables occurring in F. CNF(n, m) is the class of CNF formulas with n variables and m clauses. The *deficiency* of a formula F is defined as $\#cl(F) - \#var(F)$, denoted by $d(F)$. A formula F is minimal

* The work is supported by the National Natural Science Foundation of China (No. 60563008) and the Special Foundation for Improving Science Research Condition of Guizhou Province of China.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 79–88, 2008.

unsatisfiable (MU) if F is unsatisfiable and $F-\{C\}$ is satisfiable for any clause $C \in F$. It is well known that F is not minimal unsatisfiable if $d(F) \leq 0$ [4]. So, we denote MU(k) as the set of minimal unsatisfiable formulas with deficiency $k \geq 1$. Whether or not a formula belongs to MU(k) can be decided in polynomial time [5].

A CNF formula F is linear if any two distinct clauses in F contain at most one common variable. A CNF formula F is exact linear if any two distinct clauses in F contain exactly one common variable. We define k-CNF := {F ∈ CNF| (∀C ∈ F)(|C| = k)}, LCNF := {F ∈ CNF| F is linear}, $\text{LCNF}_{\geq \text{k}}$:= {F ∈ LCNF| (∀C ∈ F)(|C| ≥ k)}, and k-LCNF := {F ∈ LCNF| (∀C ∈ F)(|C| = k)}. The decision problems of satisfiability are denoted as k-SAT, LSAT, and k-LSAT for restricted instances to the corresponding to subclasses, respectively.

For $r > 1$, let r-UNkSAT=$\{F|F \in \text{k}-\text{CNF}, val(F) \leq 1-\frac{1}{r}\}$, r-UNLSAT= $\{F|F \in LCNF, val(F) \leq 1-\frac{1}{r}\}$, r-$\text{UNLSAT}_{\geq k}$ = $\{F|F \in \text{LCNF}_{\geq k}, val(F) \leq 1-\frac{1}{r}\}$, where $val(F)$ denoted the maximum proportion of clauses that can be satisfied to F. (A, B) is NP-hard if for any language $L \in NP$ there is a polynomial-time computable function f such that $x \in L$ then $f(x) \in A$ and otherwise $f(x) \in B$.

It is shown that LSAT is NP-completeness [6,7,8]. For the subclasses $\text{LCNF}_{\geq \text{k}}$, $\text{LSAT}_{\geq \text{k}}$ remains NP-completeness if there exists an unsatisfiable formula in $\text{LCNF}_{\geq \text{k}}$ [6,7,8].In [6,8], by the constructions of hypergraphs and latin squares, the unsatisfiable formulas in $\text{LCNF}_{\geq \text{k}}$ (k = 3, 4) are constructed, respectively. But, the method is too complex and has no generalization. In [8], it leaves the open question whether for each $k \geq 5$ there is an unsatisfiable formula in $\text{LCNF}_{\geq \text{k}}$.

Researchers are extremely interested in finding the best possible approximation algorithms for NP-hard optimization problems. Yet until the early 1990's most such questions were wide open. In particular, we did not know whether MAX3SAT has a polynomial-time ρ-approximation algorithm for every $\rho < 1$. It changed this situation when PCP Theorem appeared. PCP Theorem implies that for many NP optimization problems, computing near-optimal solutions is no easier than computing exact solutions.

In this paper, we show that there exists $r > 1$ such that (kSAT, r-UNkSAT) is NP-complete for $k \geq 3$. Based on the application of linear MU formulas[3], We construct a reduction from (4SAT, r-UN4SAT) to ($LSAT_{\geq 4}$,r'-$UNLSAT_{\geq 4}$), and proved that there exists a constant $r > 1$ such that ($LSAT_{\geq 4}$,$r-UNLSAT_{\geq 4}$) is NP-complete, so for some $s > 1$ the approximation problem s-$Approx$-$LSAT_{\geq 4}$ is NP-hard.

2 Preliminaries

We define $|F| = \sum_{1 \leq i \leq m} |C_i|$ as the size of F. In this paper, the formulas mean CNF formulas.

A formula $F = [C_1, \cdots, C_m]$ with n variables $x_1, \cdots, x_n$ in $CNF(n, m)$ can be represented as the following $n \times m$ matrix $(a_{i,j})$, called the representation

matrix of F, where $a_{ij} = +$ if $x_i \in C_j$, $a_{ij} = -$ if $\neg x_i \in C_j$, otherwise $a_{ij} = 0$ (or, blank).

A formula F is called *minimal unsatisfiable* if F is unsatisfiable, and for any clause $f \in F$, $F - \{f\}$ is satisfiable. We denote MU as the class of minimal unsatisfiable formulas, and MU(k) as the class of minimal unsatisfiable formulas with deficiency k. Let $C = (L_1 \vee \cdots \vee L_n)$ be a clause. We view a clause as a set of literals. The collection $C_1, \cdots, C_m$ of subsets of C (as a set) is a partition of C, where $C = \bigcup_{1 \leq i \leq m} C$ and $C_i \cap C_j = \phi$ for any $1 \leq i \neq j \leq m$, which corresponds to a formula $F_C = C_1 \wedge \cdots \wedge C_m$. We call F_C as a partition formula of C. Specially, the collection $\{L_1\}, \cdots, \{L_n\}$ of singleton subsets of C is called the simple partition of C, and the formula $[L_1, \cdots, L_n] = L_1 \wedge \cdots \wedge L_n$ is called the *simple partition formula* of C.

Let $F_1 = [f_1, \cdots, f_m]$ and $F_2 = [g_1, \cdots, g_m]$ be formulas. We denote $F_1 \vee_{cl} F_2 = [f_1 \vee g_1, \cdots, f_m \vee g_m]$. Similarly, let C be a clause and $F = [f_1, \cdots, f_m]$ a formula, denote $C \vee_{cl} F = [(C \vee f_1), \cdots, (C \vee_{cl} f_m)]$.

Let (A,B) denote the partial decision problem: input $x \in A \cup B$, decide $x \in A$ or $x \in B$. (If $x \notin A \cup B$, we ignore what the answer is so it is called partial decision problem). For a complexity class Γ, if there exists $C \in \Gamma$ such that $A \subseteq C$ and $B \subseteq \overline{C}$ then we say that the partial decision problem (A, B) belongs to Γ.

Definition 1. *Suppose $A \cap B = C \cap D = \phi$, if there exists a polynomial-time computable function f such that*

$x \in A \Rightarrow f(x) \in C$

$x \in B \Rightarrow f(x) \in D$ *we say (A,B) is polynomial-time reduction to (C,D) and denote by $(A, B) \leq_m^p (C, D)$.*

Theorem 1. *If $(A, B) \leq_m^p (C, D)$ and (A,B) is NP-hard, then (C,D) is NP-hard.*[1] [2]

Let $F = \{C_1, C_2, \cdots, C_m\} \in CNF$, we denote $pos(x, F)$ (resp. $neg(x, F)$) as the times of positive (resp. negative) occurrence of variable x in F , and write $occs(x, F) = pos(x, F) + pos(x, F)$. Sometimes, we denote F_{rest} as a subformula of F , which consists of part clauses of F . The following facts are clear:

(1) If $pos(x, F) > 0$ and $neg(x, F) = 0$ (or $pos(x, F) = 0$ and $neg(x, F) > 0$) for some $x \in var(F)$, then the formula F' , by deleting all clauses contained x, has the same satisfiability with φ .

(2) If $pos(x, F) = neg(x, F) = 1$, $pos(y, F) = neg(y, F) = 1$ and $F = [(x \vee y \vee C_1'), (\neg x \vee \neg y \vee C_1'), C_3, \cdots, C_m]$ (or $F = [(x \vee \neg y \vee C_1'), (\neg x \vee y \vee C_1'), C_3, \cdots, C_m]$) then the formula $F' = [(C_3, \cdots, C_m]$ has the same satisfiability with F.

From now on, for the sake of description, we assume that the formulas satisfy the following conditions: (for a formula F)

(1) For each $x \in var(F)$, $pos(x, F) > 0$ and $neg(x, F) > 0$, and

(2) For any $x, y \in var(F)(x \neq y)$, if $pos(x, F) = neg(x, F) = 1$ and $pos(y, F) = neg(y, F) = 1$ then the number of clauses contained x or y is at least three.

Definition 2. *For a CNF formula φ, define $val_\tau(\varphi)$ to be the fraction of clauses that can be satisfied by assignment τ, and $val(\varphi) = max_\tau\{val_\tau(\varphi)\}$, $sat^*(\varphi)$ to be the maximum number of clauses that can be satisfied by any assignment to φ's variables.*

In particular, if φ is satisfiable then $val(\varphi) = 1$ and $sat^*(\varphi) = \#cl(\varphi)$.

3 NP-Completeness of (k-SAT,*r*-UNk-SAT) and (LSAT,r-UNLSAT)

In this section , we introduce the class LCNF, called linear CNF formulas, and construct two reductions from 3-CNF to k-CNF (Theorem 3)and 3-CNF to LCNF(Theorem 4).

Definition 3. *(1) A formula $\varphi \in$ CNF is called linear if*
(a) φ contains no pair of complementary unit clauses, and
(b) For all $C_1, C_2 \in var(\varphi)$ with $C_1 \neq C_2$,$|var(C_1) \cap var(C_2)| \leq 1$.
Let LCNF denote the class of all linear formulas.
(2) A formula $\varphi \in CNF$ is called exact linear if φ is linear, and for all $C_1, C_2 \in var(\varphi)$ with $C_1 \neq C_2$, $|var(C_1) \cap var(C_2)| = 1$.

Let $k-$SAT $= \{F|F \in$ k $-$ CNF$, F$ is satisfiable$\}$,LSAT $= \{F|F \in$ LCNF$, F$ is satisfiable$\}$ and LSAT$_{\geq k} = \{F|F \in LCNF_{\geq k}, F$ is satisfiable$\}$, where LCNF$_{\geq k}$ denote the class of all linear formulas, in which formulas have only clauses of length at least $k(k \in N)$, and for $r > 1$, let r-UNk-SAT$=\{F|F \in$ k $-$ CNF$, val(F) \leq 1 - \frac{1}{r}\}$, r-UNLSAT$= \{F|F \in LCNF, val(F) \leq 1 - \frac{1}{r}\}$, r-UNLSAT$_{\geq k} = \{F|F \in$ LCNF$_{\geq k}, val(F) \leq 1 - \frac{1}{r}\}$.

It is shown that LSAT and LSAT$_{\geq k}(k = 3, 4)$ are NP-complete.[6,7,8]

PCP Theorem is NP=PCP(log,1)[1,2],and it is equivalent to the following theorem:

Theorem 2. *For some $r > 1$, $(3SAT, r\text{-}UN3SAT)$ is NP-complete.*[1,2]

The following result is not difficult.

Theorem 3. *For every $k \geq 3$, there is some constant $r > 1$ such that $(k - SAT, r\text{-}UNk - SAT)$ is NP-complete.*

Proof. We use induction to proof the result. By theorem 2,it holds for k=3.

Suppose that for a given constant $r > 1$, $(kSAT, r\text{-}UNkSAT)$ is NP-complete. We construct a reduction f from k-CNF to (k+1)-CNF.

Let $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m \in 3 - CNF$ or denote by $\varphi = \{C_1, C_2, \cdots, C_m\}$, where $|C_i| = 3$ for $1 \leq i \leq m$.

We construct a (k+1)-CNF formula $f(\varphi)$ as following:
$f(\varphi) = \{C_1 \vee y_1, C_1 \vee \neg y_1, C_2 \vee y_2, C_2 \vee \neg y_2, \cdots, C_m \vee y_m, C_m \vee \neg y_m\}$
where $var(\varphi) \cap \{y_1, y_2, \cdots, y_m\} = \phi$. It is clear that $f(\varphi) \in$(k+1)-CNF.

Completeness: If $\varphi \in kSAT$, then there is an assignment τ , such that $val_\tau(\varphi) = 1$, τ can extend to τ' , that is $\tau'(w) = \tau(w)$ if $w \in var(\varphi)$, 0 or 1 otherwise, then τ' satisfy the formula $f(\varphi)$. That is $f(\varphi) \in (k+1)SAT$.

Soundness: If $\varphi \in r - UNkSAT$, then for any assignment $\tau, val_\tau(\varphi) \leq (1 - \frac{1}{r})$, now for any assignment τ' for $f(\varphi)$, no matter what values be assigned to $y_1, y_2, \cdots, y_m$, we have half of the clauses in $f(\varphi)$ be satisfied, and at least $\frac{1}{r}m$ clauses be unsatisfied in rest clauses. So we have that

$$val(f(\varphi)) \leq (2m - \frac{m}{r})/(2m) = 1 - \frac{1}{2r}$$

Let $r' = 2r$, then $f(\varphi) \in r'$-UN(k+1)SAT.
That is for constant $r' = 2r$,((k+1)-SAT,r'-UN(k+1)SAT) is NP-complete.

□

Lemma 1. *For any finite set $X(|X| \geq 4)$ there is a constant $c > 0$ and there exists a strong connect directed graph G=(V,E) such that $X \subseteq V$ and satisfies the following properties:*
(a) $|V| \leq c|X|$,
(b) for any $x \in X$, its output degree and input degree is 1, for any $x \in V - X$, its degree (sum of output degree and input degree) is at most 3,and
(c) if $S \subseteq V$ and $|S \cap X| \leq |X|/2$,then
$|E \cap (S \times (V - S))| \geq |S \cap X|$ *and*
$|E \cap ((V - S) \times S)| \geq |S \cap X|$

About the proof of this lemma, one can refer to lemma 11.13 in [1].

Theorem 4. *There is some constant $r > 1$ such that (LSAT, r-UNLSAT) is NP-complete.*

Proof. By Theorem 2, we can assume a given $r > 1$ such that $(3SAT, r\text{-}UN3SAT)$ is NP-complete, we construct a polynomial-time reduction from $(3SAT, r\text{-}UN3SAT)$ to $(LSAT, r'\text{-}UNLSAT)$ where r' is some constant depending on r.

Let $\varphi \in 3CNF$, $\varphi = \{C_1, C_2, \cdots, C_m\}$, we transform φ into a linear formula φ^{lin} by invoking the following procedure 1.

Procedure 1: (Linear transformation for CNF formulas)
Input: a CNF formula φ with variables $x_1, x_2, \cdots, x_n$;
Output: a linear formula φ^{lin} ;
Begin
$\varphi^{lin} := \varphi; i := 1;$
while $(i \leq n) \wedge (occs(x_i, \varphi^{lin}) \geq 3)$ do
(let $\varphi^{lin} = [(x_i \vee f_1), \cdots, (x_i \vee f_{s_i}), (\neg x_i \vee g_1), \cdots, (\neg x_i g_{t_i}), \varphi^{lin}_{rest}]$,
where $(s_i + t_i = occ(x_i, \varphi^{lin}))$ and $s_i = pos(x_i, \varphi^{lin}), t_i = neg(x_i, \varphi^{lin})$)
case $occs(x_i, \varphi^{lin}) = 3$,
introducing new variables $y_{i,1}, y_{i,2}, y_{i,3}$;
$\varphi^{lin} \coloneqq [(y_{i,1} \vee f_1), \cdots, (y_{i,s_i} \vee f_{s_i}), (\neg y_{i,s_i+1} \vee g_1), \cdots, (\neg y_{i,s_i+t_i} \vee g_{t_i}), \varphi^{lin}_{rest}] +$
$[(\neg y_{i,1} \vee y_{i,2}), (\neg y_{i,2} \vee y_{i,3}), (\neg y_{i,3} \vee y_{i,1})]$;
case $occs(x_i, \varphi^{lin}) = k \geq 4$,

introducing new variables $y_{i,1}, y_{i,2}, \cdots, y_{i,k}$ to replace all occurrences x_i in the formula, and will append a series clauses of $u \to v(\neg u \vee v)$ such that all $y_{i,j}$ can take the same value and the number of satisfied clauses can be maximized. So let $X_i = \{y_{i,1}, y_{i,2}, \cdots, y_{i,k}\}$, by lemma 1,we construct directed graph $G_{x_i} = (V_{x_i}, E_{x_i})$ satisfies the properties (a)(b)(c),all vertexes in V_{x_i} as new boolean variables and all edges $u \to v$ as new clauses, that is

$\varphi^{lin} := \{(y_{i,1} \vee f_1), \cdots, (y_{i,s_i} \vee f_{s_i}), (\neg y_{i,s_i+1} \vee g_1), \cdots, (\neg y_{i,s_i+t_i} \vee g_{t_i}), \varphi^{lin}_{rest}\}+$

$\{\neg u \vee v | u, v \in V_{x_i}, u \to v \in E_{x_i}\};$

$i := i + 1;$

enddo;

output φ^{lin};

end.

The above procedure can be completed in polynomial-time of n.

We note that φ^{lin} has the following property: for any assignment τ, there exists an assignment τ' such that

(1) the number of clauses of φ^{lin} which τ' satisfied is not less than the number of clauses which τ satisfied, and

(2) τ' takes the same value to introduced new variables $y_{i,1}, y_{i,2}, \cdots, y_{i,s_i+t_i}$ for every x_i, that is $\tau'(y_{i,1}) = \tau'(y_{i,2}) = \cdots = \tau'(y_{i,s_i+t_i})$ for every i.

In fact, if τ can not satisfy (2) for some x_i, then we let variables$y_{i,1}, y_{i,2}, \cdots, y_{s_i+t_i}$ take the majority value and induce a new assignment τ' , that is

$$maj(y_{i,1}, y_{i,2}, \cdots, y_{s_i+t_i}) = \begin{cases} 1, \text{ if at least a half } \tau(y_{i,j}) = 1, \\ 0, \qquad\qquad \text{otherwise,} \end{cases}$$

and let $\tau'(y_{i,1}) = \tau'(y_{i,2}) = \cdots = \tau'(y_{i,s_i+t_i}) = maj(y_{i,1}, y_{i,2}, \cdots, y_{s_i+t_i})$, for fixed i, let $S_i = \{y_{i,j} \in V_{x_i} | \tau(y_{i,j}) \neq \tau'(y_{i,j})\}$, we note that the assignment from τ to τ' may at most $|X_i \cap S_i|$ satisfied clauses in φ change to be unsatisfied, but by lemma 1(c),in the new introduced clauses will have at least $|X_i \cap S_i|$ unsatisfied clauses change to be satisfied. So τ' satisfy the condition (1).

Completeness: If $\varphi \in 3SAT$, then there is an assignment τ , such that $val_\tau(\varphi) = 1$, τ induce to τ' , that is $\tau'(y_{i,1}) = \tau'(y_{i,2}) = \tau'(y_{i,3}) = \tau(x_i)$ for every x_i with $occs(x_i, \varphi) = s_i + t_i = 3$ and if $u \in V_{x_i}$ then $\tau'(u) = \tau(x_i)$ for every x_i with $occs(x_i, \varphi) = s_i + t_i \geq 4$, it is clear that τ' satisfy the formula φ^{lin}. That is $\varphi^{lin} \in LSAT$.

Soundness: If $\varphi \in r - 3UNSAT$, then for any assignment $\tau, val_\tau(\varphi) \leq (1 - \frac{1}{r})$,

For any assignment τ' for φ^{lin} , by above property, we could induce an assignment τ'' satisfy above condition (1) and (2) , please note, $\#cl(\varphi) = m$, φ^{lin} has at most $3mc$ variables, where c is the constant of lemma 1, because every variable occurs at most 3 times, so φ^{lin} has at most $9cm/2$ clauses (every clause has at least 2 variables) , that is $\#cl(\varphi^{lin}) \leq 9cm/2$, we have

$$\begin{aligned} val_{\tau'}(\varphi^{lin}) &\leq (\#cl(\varphi^{lin}) - \tfrac{1}{r}m)/\#cl(\varphi^{lin}) \\ &= 1 - \tfrac{1}{r}\tfrac{m}{\#cl(\varphi^{lin})} \\ &\leq (1 - \tfrac{1}{r}\tfrac{2}{9c} \\ &= 1 - \tfrac{2}{9rc} = 1 - \tfrac{1}{r'} \end{aligned}$$

where $r' = 9rc/2$, so $\varphi^{lin} \in r'$-UNLSAT. □

Let $s > 1$, s-Approx-LSAT is the problem of, given a $LCNF$ formula φ, finding an assignment satisfy at least $sat^*(\varphi)/s$ clauses, then we have

Corollary 1. *There exists a constant $s > 1$ such that s-Approx-LSAT is NP-hard.*

4 NP-Completeness of (LSAT$_{\geq 4}$, r-UNLSAT$_{\geq 4}$)

In [3], we have showed that for k=3 there exists some $r > 1$ such that (LSAT$_{\geq k}$, r-UNLSAT$_{\geq k}$) is NP-complete. Now we consider NP-completeness of ($LSAT_{\geq 4}$, r-$UNLSAT_{\geq 4}$) .

Let $B_6 = [(x_1, x_3), (\neg x_1, x_2), (\neg x_2, x_3), (\neg x_3, x_4), (\neg x_4, x_5), (\neg x_5, \neg x_3)]$, its representation is:

$$\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{pmatrix} + & - & & & \\ & + & - & & \\ + & & + & - & - \\ & & & + & - \\ & & & & + & - \end{pmatrix}$$

It is clear that B_6 belongs to $MU(1)$ and B_6 is a linear formula.

The standard MU formula S_6 with 6 variables $x_1, \cdots, x_6$ is defined as $S_6 = \wedge_{(\varepsilon_1,\cdots,\varepsilon_6)\in\{0,1\}^6}(x_1^{\varepsilon_1} \vee \cdots \vee x_6^{\varepsilon_6})$,where $x_i^0 = x_i$ and $x_i^1 = \neg x_i$ for $1 \leq i \leq 6$. Denote the clause $X_{\varepsilon_1,\cdots,\varepsilon_6} = x_1^{\varepsilon_1} \vee \cdots \vee x_6^{\varepsilon_6}$, $F_{\varepsilon_1,\cdots,\varepsilon_6} = [x_1^{\varepsilon_1}, \cdots, x_6^{\varepsilon_6}] = x_1^{\varepsilon_1} \wedge \cdots \wedge x_6^{\varepsilon_6}$,

$SL_3 := \wedge_{(\varepsilon_1,\cdots,\varepsilon_6)\in\{0,1\}^6}(F_{\varepsilon_1,\cdots,\varepsilon_6} \vee_{cl} B_6^{\varepsilon_1,\cdots,\varepsilon_6})$

where $B_6^{\varepsilon_1,\cdots,\varepsilon_6}$ is a copy of B_6 and it restricts $var(B_6^{\epsilon_1\cdots\ \epsilon_6}) \cap var(B_6^{\epsilon_1'\cdots\epsilon_6'}) = \phi$ for any distinct $(\epsilon_1, \cdots, \epsilon_6), (\epsilon_1', \cdots, \epsilon_6') \in \{0,1\}^6$, and $var(B_6^{\epsilon_1\cdots\ \epsilon_6}) \cap var(S_6) = \phi$ for any $(\epsilon_1, \cdots, \epsilon_6) \in \{0,1\}^6$.

SL_3 is a linear MU formula [3].

Please note that $\#cl(SL_3) = 6 \cdot 2^6$, and $|C| = 3$ for each $C \in SL_3$.

We define inductively a counting functions of clauses $cl(k)$ for $k \geq 3$: $cl(3) = 6 \cdot 2^6$ and $cl(k+1) = cl(k) \cdot 2^{cl(k)}$ for $k \geq 3$.

For the case of $k \geq 3$, suppose that the linear formula SL_k has been constructed such that SL_k is a linear MU formula, and the length of each clause in SL_k equals to k.

In [3], we define inductively the following linear MU formula:

$$SL_{k+1} := \bigwedge_{(\epsilon_1,\cdots,\ \epsilon_{cl(k)})\in\{0,1\}^{cl(k)}}(F_{\epsilon_1\cdots\ \epsilon_{cl(k)}} \vee_{cl} SL_k^{\epsilon_1\cdots\ \epsilon_{cl(k)}})$$

where, for $(\epsilon_1, \cdots, \epsilon_{cl(k)}) \in \{0,1\}^{cl(k)}$

(a) $F_{\epsilon_1\cdots\ \epsilon_{cl(k)}}$ is the simple partition formula of clause $X_{\epsilon_1\cdots\ \epsilon_{cl(k)}} \in S_{cl(k)}$.

(b) $SL_k^{\epsilon_1\cdots\ \epsilon_{cl(k)}}$ is a copy SL_k with new variables.

$S_{cl(k)}$ is minimal unsatisfiable, SL_k is both minimal unsatisfiable and linear. SL_{k+1} is a linear MU formula.

We will use the case SL_4 in the proof of theorem 5, please note that SL_4 is linear and MU formula, and it contains $cl(4)(= cl(3) \cdot 2^{cl(3)} = 6 \cdot 2^6 \cdot 2^{6 \cdot 2^6} = 6 \cdot 2^{6(1+2^6)})$ clauses, and each clause has length 4.

Based on the application of minimal unsatisfiable formulas SL_4, we have the following result.

Theorem 5. *There exists a constant $r > 1$ such that $(LSAT_{\geq 4}, r\text{-}UNLSAT_{\geq 4})$ is NP-complete.*

Proof. we construct a reduction from $(4SAT, r\text{-}UN4SAT)$ to $(LSAT_{\geq 4}$, $r\text{-}UNLSAT_{\geq 4})$ polynomially.

Let $F = [C_1, \cdots, C_m]$ be a 4CNF formula,$\#cl(F) = m, var(F) = \{x_1, \cdots, x_n\}$, $|F| = \sum_{1 \leq i \leq m} |C_i| = 4m$, W.l.o.g., we assume $occs(x, F) \geq 3$ for each $x \in var(F)$. We now transform F into F^* in 4-LCNF by the following two stages.

Stage 1: Call Procedure 1(Linear Transformation for CNF formulas) to transform F into a linear formula F^{lin} . Please note that for any clause $C \in F^{lin}$, $|C| =$ 4 or $|C| = 2$.

Stage2: Lengthen clauses of the length 2 in F^{lin}

We take a linear MU formula SL_4 which mentioned above[3], and assuming $SL_4 = [(l_1 \vee l_2 \vee l_3 \vee l_4), f_1, \cdots, f_s]$, where $s = cl(4) - 1$ and $|f_i| = 4$ for $1 \leq i \leq s$. Define $H = [(l_3 \vee l_4), f_1, \cdots, f_s]$. The following procedure 2 generates a linear formula F^* in 4-CNF.

Procedure 2: (Lengthen clauses in linear formula)
Input: a formula F^{lin};
Output: a linear formula F^* in 4-CNF;
Begin
$F^* := F^{lin}; i := 1;$
while $((\exists C \in F^{lin}) \wedge (|C| = 2))$ do
taking a copy $H^i = [(l_3^i \vee l_4^i), f_1^i, \cdots, f_s^i]$ of H with new variables, that is for any $i \neq j, var(H^i) \cap var(H^j) = \phi$;
$F^* := (F^* - C) + (C \vee l_3^i \vee l_4^i) + [f_1^i, \cdots, f_s^i]$;
enddo;
output F^* ;
end.
(For formulas F_1 and F_2, $F_1 + F_2$ means $F_1 \wedge F_2$)

The above stages can be completed in polynomial-time of $|F|$, and we have $|F^*| = |F| \cdot |H|$.

Please note that the formula F^* has also the property similarly to which Procedure 1 mentioned, that is : for any assignment τ,there exists an assignment τ' such that

(1) the number of clauses of F^* which τ' satisfied is not less than the number of clauses which τ satisfied, and

(2) τ' takes the same value to $y_{i,1}, y_{i,2}, \cdots, y_{i,s_i+t_i}$ for every i, that is $\tau'(y_{i,1}) = \tau'(y_{i,2}) = \cdots = \tau'(y_{i,s_i+t_i})$ for every i.

Completeness: If $F \in 4SAT$, then there is an assignment τ satisfy F, by the proof of Theorem 4 there is an assignment τ' satisfy F^{lin} . As $SL4$ is MU formula, for every copy H^i of H,$[f_1^i, \cdots, f_s^i] = H^i - \{(l_3^i \vee l_4^i)\}$ is satisfiable, so there is a assignment τ^i satisfy $[f_1^i, \cdots, f_s^i]$. we can extend τ' to τ'' by combining of τ' and all τ^i, such that τ'' satisfy F^* . So $F^* \in LSAT_{\geq 4}$.

Soundness: If $x \in r - UN4SAT$,$val(F) \leq 1 - \frac{1}{r}$, then $val(F^{lin}) \leq 1 - \frac{1}{5r}$,let $m' = \#cl(F^{lin})$ and $m'' = \#cl(F^*)$, and please note $m' = m + 4m = 5m$,and $m'' = m + 4m + 4m \times (cl(4) - 1) = m + 4m \times cl(4)$

$$\begin{aligned} val(F^*) &\leq = [m" - \tfrac{1}{r}m]/m'' \\ &= 1 - \tfrac{1}{r} \cdot \tfrac{m}{m''} \\ &= 1 - \tfrac{1}{r(1+4\times cl(4))} = 1 - \tfrac{1}{r'}, \end{aligned}$$

where $r' = r(1 + 4 \times cl(4))$, so $F^* \in r'\text{-}UNLSAT_{\geq 4}$. □

Corollary 2. *There exists a constant $r > 1$ such that r-Approx-$LSAT_{\geq 4}$ is NP-hard, where r-Approx-$LSAT_{\geq 4}$ is the problem of given a $LCNF_{\geq 4}$ formula F output an assignment which satisfy at least $sat^*(F)/r$ clauses.*

5 Conclusions and Future Works

Based on the application of minimal unsatisfiable formulas SL_4 [3], we show a polynomial-time reduction from (4SAT, r-UN4SAT) to $(LSAT, r'\text{-}UNLSAT)$ and proved that $(LSAT, r'\text{-}UNLSAT)$ is NP-complete for some $r > 1$, and hence the approximation *s-Approx-LSAT* is NP-hard. We also construct a reduction from (4SAT, r'-UN4SAT) to $(LSAT_{\geq 4}, r\text{-}UNLSAT_{\geq 4})$, and proved that there exists a constant $r > 1$ such that $(LSAT_{\geq 4}, r\text{-}UNLSAT_{\geq 4})$ is also NP-complete. The future works is to investigate deeply structures and characterizations of linear formulas, and for $k \geq 5$ the NP-completeness of $(LST_{\geq k}, r\text{-}UNLSAT_{\geq k})$, and the NP-completeness of the gap decision problem of classes (r, s)-CNF's and its approximation problem.

References

1. Du, D., Ko, K.-I., Wang, J.: Introduction to Computational Complexity (Chinese). Higher Education Press, P.R.China (2004)
2. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Princeton University, Princeton (2007)
3. Zhang, Q., Xu, D.: The Existence of Unsatisfiable Formulas in k-LCNF for k ≥ 3. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 616–623. Springer, Heidelberg (2007)
4. Davydov, G., Davydova, I., Kleine Büning, H.: An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. Annals of Mathematics and Artificial Intelligence 23, 229–245 (1998)

5. Fleischner, H., Kullmann, O., Szeider, S.: Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. Theoretical Computer Science 289(1), 503–516 (2002)
6. Porschen, S., Speckenmeyer, E.: Linear CNF formulas and satisfiability, Tech. Report zaik2006-520, University Köln (2006)
7. Porschen, S., Speckenmeyer, E., Randerath, B.: On Linear CNF Formulas. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 212–225. Springer, Heidelberg (2006)
8. Porschen, S., Speckenmeyer, E.: NP-completeness of SAT for restricted linear formulas classes. In: Proceedings of Guangzhou Symposium on Satisfiability in Logic-Based Modeling, vol. 1, pp. 108–121, pp. 111–123 (1997)
9. Cook, S.C.: The complexity of theorem-proving procedures. In: Proc. 3rd ACM STOC, pp. 151–158 (1971)

Absorbing Random Walks and the NAE2SAT Problem

K. Subramani*

LDCSEE,
West Virginia University,
Morgantown, WV
ksmani@csee.wvu.edu

Abstract. In this paper, we propose a simple, randomized algorithm for the NAE2SAT problem; the analysis of the algorithm uses the theory of symmetric, absorbing random walks. NAESAT (Not-All-Equal SAT) is the variant of the Satisfiability problem (SAT), in which we are interested in an assignment that satisfies all the clauses, but falsifies at least one literal in each clause. We show that the NAE2SAT problem admits an extremely simple literal-flipping algorithm, in precisely the same way that 2SAT does. On a satisfiable instance involving n variables, our algorithm finds a satisfying assignment using at most $\frac{9}{4}n^2$ verification calls with probability at least $\frac{5}{6}$. The randomized algorithm takes $O(1)$ extra space, in the presence of a verifier and provides an interesting insight into checking whether a graph is bipartite. It must be noted that the bounds we derive are much sharper than the ones in [1].

1 Introduction

This paper details a randomized algorithm for the problem of determining whether an instance of NAE2SAT is satisfiable. To recapitulate, NAESAT is the version of clausal satisfiability (SAT), in which we seek an assignment that satisfies all the clauses, while falsifying at least one literal in each clause. NAESAT is known to be NP-complete, even when there are at most three literals per clause (NAE3SAT) [2]. The NAE2SAT problem is the variant of NAESAT in which there are exactly two literals per clause and is known to be solvable in polynomial time. For instance, one could use each clause as a partitioning constraint and decide an instance in linear time. From a complexity-theoretic perspective, [3] established that NAE2SAT is in the complexity class SL; more recently, [4] proved that SL=L, from which it follows that NAE2SAT is in L. Our algorithm is based on the literal-flipping algorithm proposed in [1] for the 2SAT problem and is extremely space-efficient. In particular, on a satisfiable NAE2SAT instance having n variables and m clauses, the expected number of literal-flips to find the satisfying assignment is $\frac{n^2}{4}$. Further, if a satisfying assignment is not found within $\frac{9}{4}n^2$ literal-flips, then the probability that the instance is not satisfiable is at least $\frac{5}{6}$.

The principal contributions of this paper are as follows:

(a) The design and analysis of a randomized, literal-flipping algorithm for the NAE2SAT problem.

* This research was supported in part by a research grant from the Air-Force Office of Scientific Research under contract FA9550-06-1-0050.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 89–100, 2008.

(b) Some new observations on the convergence times of absorbing random walks.
(c) Establishing the complexity of NAE2SATPOS, which is the variant of NAE2SAT, in which all literals are positive.

2 Preliminaries

Let $\phi = C_1 \wedge C_2 \ldots \wedge C_m$ denote a 2CNF formula on the literal set $L = \{x_1, \bar{x_1}, x_2, \bar{x_2} \ldots x_n, \bar{x_n}\}$.

Definition 1. *The Not-All-Equal satisfiability problem for 2CNF formulas (NAE2SAT), is concerned with checking whether there exists a satisfying assignment to ϕ, such that at least one literal in each clause is set to* **false**. *If such an assignment exists, then ϕ is said to be nae-satisfiable.*

For instance, the 2CNF formula $\phi = (x_1, x_2) \wedge (x_2, x_3)$ is nae-satisfiable ($x_1 = \mathbf{true}\ x_2 = \mathbf{false}\ x_3 = \mathbf{true}$), while the 2CNF formula $\phi = (\bar{x_1}, x_2) \wedge (x_1, x_2))$ is not nae-satisfiable.

Without loss of generality, we assume that each clause has *exactly* two literals, since ϕ cannot be nae-satisfiable, if it has a clause with exactly one literal.

In order to understand our approach for the NAE2SAT problem, we discuss a graph theoretic approach. Given a 2CNF formula $\phi(x)$, we can construct its nae-graph $G_\phi(x)$ as follows:

(a) For each literal x_i, create a vertex labeled x_i. Note that corresponding to boolean variable x_i, there are two literals, viz., x_i, and its complement literal, $\bar{x_i}$.
(b) Corresponding to each clause $(x_i,\ y_j)$ (say), add the undirected edges $\bar{y_j} - x_i$ and $\bar{x_i} - y_j$.

The graph that is created is similar but not identical to the implication graph technique for 2SAT discussed in [5]. In particular, note that in our case, the graph edges are undirected, whereas in [5], the edges represent implications and are therefore directed. The implication graph G models the fact that each 2-literal clause $(x_i,\ \bar{y_j})$ can be thought of as a pair of implications $\bar{x_i} \rightarrow y_j$ and $\bar{y_j} \rightarrow x_i$ connected conjunctively. In [5], it is shown that the $\phi(x)$ is unsatisfiable if and only there exists a path from a vertex x_i to its complement vertex $\bar{x_i}$ and vice versa. We now provide a test for the nae-satisfiability of $\phi(x)$.

Theorem 1. *Let $\phi(x)$ denote a CNF formula and $G_{\phi(x)}$ denote the corresponding nae-graph built as per the above rules. Then $\phi(x)$ is nae-unsatisfiable if and only there exists a path from a vertex x_i to its complement vertex $\bar{x_i}$ in $G_{\phi(x)}$.*

Proof: Assume that there exists a path from a vertex x_i to its complement vertex $\bar{x_i}$ denoted by $x_i \rightarrow x_1 \rightarrow \ldots \rightarrow x_k \rightarrow \bar{x_1}$. As per the construction of the implication graph, the following clauses are part of the clause set $\phi(x)$: $(\bar{x_i}, x_1)(\bar{x_1}, x_2) \ldots (\bar{x_k}, \bar{x_i})$. Now consider an assignment in which x_i is set to **true**. It follows that x_1 must be set to **true**. Arguing similarly, $x_2, x_3, \ldots, x_k$ must all be set to **true**. However, the last clause $(\bar{x_k}, \bar{x_i})$ has both literals set to **false** and is therefore not nae-satisfied. Likewise,

consider an assignment in which x_i is set to **false**. This forces x_1 to be **false**; arguing similarly, $x_2, x_3, \ldots, x_k$ must all be set to **false**. However, this forces both literals in the last clause to be **true**, thereby falsifying it from a nae-sat perspective.

The converse can be argued similarly, albeit with some effort involving case-based analysis. □

The above theorem immediately establishes that NAE2SAT can be solved in linear time using a variant of the connected components approach discussed in [5]. All that we need to ensure that is that no literal is reachable from its complement.

2.1 Assignment Verification for NAE2SAT

Consider an instance F of NAE2SAT, i.e., a 2CNF formula on the x variables only. Let $\vec{\mathbf{a}}$ represent a $\{\mathbf{true}, \mathbf{false}\}$ assignment to the variables of F. It is straightforward to verify whether $\vec{\mathbf{a}}$ nae-satisfies F, by substituting the value of literals in each clause, as specified by $\vec{\mathbf{a}}$ and checking that all clauses evaluate to **true**, with at least one literal set to **false** in each clause. We now present an implication graph interpretation of a nae-satisfying assignment. Let G represent the implication graph of F and consider the labeling of the vertices of G, as per $\vec{\mathbf{a}}$.

Theorem 2. *An assignment $\vec{\mathbf{a}}$ nae-satisfies a 2CNF formula $\phi(x)$, if and only if there is no arc, with tail vertex assigned* **true** *and head vertex assigned to* **false***, i.e., a (***true**→**false***) arc* **or** *a (***false**→**true***) arc, in the implication graph $G_\phi(x)$ of $\phi(x)$.*

Proof: Let $(x_i,\ x_j)$ denote an arbitrary clause of $\phi(x)$; this clause contributes the two arcs $l_1 : \bar{x_i} \rightarrow x_j$ and $l_2 : \bar{x_j} \rightarrow x_i$ to the implication graph G. If this clause is satisfied by the assignment $\vec{\mathbf{a}}$, either x_i is set to **true** and x_j is set to **false** or x_j is set to **true** and x_i is set to **false**. If x_i is set to **true** and x_j is set to **false**, the tail of arc l_1 is set to **false** and so is its head, while the head of arc l_2 is set to **true** and so is its head. We thus see that there is no (**true**→**false**) arc or (**false**→**true**) arc. The case when x_j is assigned to **true** and x_i is assigned to **false** can be argued symmetrically.

Now consider the case in which both x_i and x_j are set to **false** in $\vec{\mathbf{a}}$. Both the arcs l_1 and l_2 are (**true**→**false**) arcs. Similarly, the case in which x_i and x_j are both set to **true**, results in l_1 and l_2 both becoming (**false**→**true**) arcs. □

From Theorem 2, it follows that given an assignment to a NAE2SAT instance $\phi(x)$, the verification process consists of scanning through the implication graph $G_\phi(x)$ and ensuring that there does not exist a (**true**→**false**) or (**false**→**true**) arc. If the instance $\phi(x)$ has m clauses and n variables, then as per the implication graph construction, $G_\phi(x)$ has $2 \cdot m$ arcs. Hence, the verification process can be accomplished in $O(m)$ time.

2.2 Complexity Classes

Definition 2. *A language $\mathcal{L}$ is said to be in* RL*, if there exists a randomized algorithm which recognizes it in logarithmic space.*

Definition 3. *The undirected $s-t$ connectivity problem (USTCON) is defined as follows: Given an undirected graph $G = \langle V, E \rangle$ and two vertices, $s, t \in V$, is t reachable from s?*

Definition 4. *A language $\mathcal{L}$ is said to be in* SL*, if it can be log-space reduced to USTCON.*

3 Markov Chains and Random Walks

This section contains preliminaries on Markov Chains and Random Walks that may not be familiar to non-experts in Probability Theory. I have included this material so that the paper as a whole is accessible to researchers in both Computer Science and Statistics. If the reviewers feel that this section does not add value to the paper, it will be removed.

There are two types of theorems that are described here; the first type are known results for which appropriate references are provided and the second type are results which are proved here for the first time, to the best of our knowledge. The latter theorems are marked as **O** for original.

Let $\mathcal{R}$ denote a stochastic process which can assume one of the following values: $S = \{0, 1, 2, \ldots n\}$. If $\mathcal{R}$ assumes the value i at a particular time, then it is said to be in state i at that time. Further, suppose that when it is in state i, the probability that it will move to state j, at the next instant is a fixed constant p_{ij}, independent of the history of the process. $\mathcal{R}$ is said to be a *Finite Markov Chain* over the state space S.

Definition 5. *A Finite Markov Chain over the state space $S = \{0, 1, \ldots, n\}$ is said to be a* Random Walk*, if for some fixed constants $p \in (0, 1)$, $p_0 \in [0, 1]$, $p_n \in [0, 1]$,*

$$\begin{aligned} p_{i\,i+1} = p = 1 - p_{i\,i-1},\ i = 1, 2, \ldots, n-1, \\ p_{0\,1} = p_0 \\ p_{n\,n-1} = p_n \end{aligned}$$

Definition 6. *A random walk $\mathcal{R}$ is said to be absorbing, if $p_{0\,1} = p_{n,n-1} = 0$.*

States 0 and n are said to be absorbing barriers of the random walk.

Definition 7. *An absorbing random walk is said to be symmetric, if $p = \frac{1}{2}$.*

The rest of this paper will be concerned with absorbing, symmetric random walks (ASR walks) only. An ASR walk has the following game-theoretic interpretation: Imagine a drunkard on a straight road, with his house on one end and a bar on the other. Assume that the drunkard is at some point i, in between the house and the bar. Point i is the initial position of the game. In each state the drunkard is currently in, he takes one step towards the bar with probability one-half or one step towards his house with probability one-half. The game is over when the drunkard reaches either his home or the bar.

Definition 8. *The absorption time of state i, $0 \leq i \leq n$, in an ASR walk $\mathcal{R}$, is the expected number of steps for $\mathcal{R}$ to reach an absorbing state, given that it is currently in state i.*

The absorption time of state i, is denoted by $t(i)$; we thus have,

$$t(i) = \mathbf{E}[\text{ASR walk } \mathcal{R} \text{ to reach 0 or n} \,|\, \mathcal{R} \text{ is currently in state i}]$$

Definition 9. *The absorption time of an ASR walk $\mathcal{R}$ is defined as the maximum absorption time over all the states in $\mathcal{R}$.*

The absorption time of $\mathcal{R}$ is denoted by $\mathcal{R}_t$; we thus have,

$$\mathcal{R}_t = \max_{0 \le i \le n} t(i)$$

The literature has shown that every state in an absorbing, symmetric random walk is recurrent, i.e., the probability that a state is ever reached is 1 [6]. From the perspective of algorithmic efficiency, mere recurrence is not sufficient; it is important that the absorption time of a state is s a small number, preferably a polynomial function of the total number of states.

We now proceed to compute the absorption time of various states in an ASR walk $\mathcal{R}$; in order to carry out this computation, we need the following technical lemma that helps us to compute the expectation of a random variable by conditioning it on a different random variable. This lemma has been proved in [7].

Lemma 1. *Let X and Y denote two random variables; let $\mathbf{E}[X \mid Y]$ denote that function of the random variable Y, whose value at $Y = y$ is $\mathbf{E}[X \mid Y = y]$. Then,*

$$\mathbf{E}[X] = \mathbf{E}[\mathbf{E}[X \mid Y]]. \tag{1}$$

In other words,

$$\mathbf{E}[X] = \sum_{y} \mathbf{E}[X \mid Y = y] \cdot \mathbf{Pr}[Y = y]. \tag{2}$$

We use $t(i)$ to denote the absorption time of state i and $\mathcal{R}_t$ to denote the absorption time of the ASR walk $\mathcal{R}$. Observe that if the ASR walk is currently in state i, where i is a non-absorbing state, then at the next step, it will be in state $(i+1)$ with probability one-half and it will be in state $(i-1)$, with probability one-half. In the former case, the absorption time for state i is $(1 + t(i+1))$, while in the latter case, it is $(1 + t(i-1))$. Noting that $t(0) = t(n) = 0$, we apply Lemma (1) to derive the following set of equations for computing the absorption times of states in the ASR walk.

$$\begin{aligned} t(0) &= 0 \\ t(i) &= \frac{1}{2} \cdot (t(i-1) + 1) + \frac{1}{2} \cdot (t(i+1) + 1), \ 0 < i < n \\ t(n) &= 0 \end{aligned} \tag{3}$$

Note that System (3) contains $(n+1)$ equations and $(n+1)$ unknowns and can be represented as: $\mathbf{A} \cdot \vec{\mathbf{x}} = \vec{\mathbf{b}}$, where, $\vec{\mathbf{x}}$ is an $(n+1)$-vector representing the expected absorption

times of the states $([t(0),\ t(1),\ \ldots, t(n)]^T)$, $\vec{\mathbf{b}}$ is an $(n+1)$-vector $[0,\ 1,\ 1,\ \ldots, 1,\ 0]^T$, and $\mathbf{A}$ is the $(n+1) \times (n+1)$ matrix represented by:

$$\begin{bmatrix} 1 & 0 & \ldots & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2} & \ldots \\ \vdots & \vdots & \vdots & \vdots \\ \ldots & -\frac{1}{2} & 1 & -\frac{1}{2} \\ 0 & 0 & \ldots & 1 \end{bmatrix}$$

Row i $(0 < i < n)$, of $\mathbf{A}$ has 1 in the diagonal entry and a negative one-half in the entry preceding and succeeding it. All other entries in this row are zero. Row 0 and Row n are unit vectors with the unit entry occupying the diagonal entry. It is not hard to see that $\mathbf{A}$ is non-singular and therefore, System (3) has a unique solution.

We need the following lemma to aid us in solving System (3).

Lemma 2

$$t(i) = t(n-i),\ \forall i = 0, 1,\ \ldots, n$$

Proof: By symmetry of the random walk. □

The technique to solve System (3) (which is a non-trivial contribution) appears in the journal version of the paper. We show that in the worst case, the absorption time of a state (i.e., the expected number of steps for the random walk to reach an absorbing state, from a state) in an ASR walk is $\frac{n^2}{4}$, i.e., $\max_{0 \leq i \leq n} t(i) \leq \frac{n^2}{4}$.

Theorem 3. *Let* $\mathbf{Var}[t(i)]$ *denote the variance of the absorption time of state* i*, in the ASR walk* $\mathcal{R}$*. Then,*

$$\max_{0 \leq i \leq n} \mathbf{Var}[t(i)] \leq \frac{2}{3} n^4.$$

Proof: Will provide in revised version. □

The following theorem is known as Chebyshev's inequality and is proved in [7] among other places.

Theorem 4. *Let* X *denote a random variable, with mean* $\mathbf{E}[X]$ *and variance* σ^2*. Then, for any* $k > 0$*, we have,*

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq k] \leq \frac{\sigma^2}{k^2}$$

Theorem 5. *(***O***) Given an ASR walk* $\mathcal{R}$*, which is initially in an arbitrary state* i*,* $0 \leq i \leq n$*, the probability that* $\mathcal{R}$ *does not reach an absorbing state in* $\frac{9}{4}n^2$ *steps is at most* $\frac{1}{6}$*.*

Proof: Let X denote the worst-case number of steps taken by the ASR walk $\mathcal{R}$ to reach an absorbing state. We are interested in the quantity $\mathbf{Pr}[X \geq \frac{9}{4}n^2]$.

Note that,

$$\begin{aligned}
\mathbf{Pr}[X \geq \frac{9}{4}n^2] &= \mathbf{Pr}[X - \frac{n^2}{4} \geq 2 \cdot n^2] \\
&\leq \mathbf{Pr}[|X - \frac{n^2}{4}| \geq 2 \cdot n^2] \\
&= \mathbf{Pr}[X - \mathbf{E}[X]| \geq 2 \cdot n^2] \\
&\leq \frac{\frac{2}{3}n^4}{(2n^2)^2}, \text{ using Chebyshev's inequality} \\
&= \frac{1}{6}
\end{aligned}$$

□

3.1 The Convergence Numbers

Let $G(n, k)$ denote the expected number of steps taken by an ASR walk to reach an absorbing state, assuming that it is currently in state k, $0 \leq k \leq n$, i.e., $G(n, k)$ is the absorption time of state k.

Table (1) represents the computed values of $G(n, k)$ for small values of n. For each value of n, the corresponding row stores the absorption time for all $n+1$ initial positions of the random walk.

Table 1. Convergence Numbers

Number of variables	Starting Point of Walk
$n = 0$	0
$n = 1$	0 0
$n = 2$	0 1 0
$n = 3$	0 2 2 0
$n = 4$	0 3 4 3 0
$n = 5$	0 4 6 6 4 0
$n = 6$	0 5 8 9 8 5 0
$n = 7$	0 6 10 12 12 10 6 0

Each of the following theorems (which do not appear in the literature, to the best of our knowledge) can be proved using induction.

Theorem 6. *For all* $n \geq 2$, $G(n, 1) = G(n-1, 1) + 1$.

Theorem 7. *For all* $n \geq 3$, $G(n, 2) = G(n-2, 2) + 2$.

Theorem 8. *For all* $n \geq k$, $G(n, k) = G(n-k, k) + k$.

Theorem 9. *For all* n, $G(n, k) = G(n, n-k)$.

4 Algorithm and Analysis

Algorithm 4.1 is our strategy to solve the NAE2SAT problem.

Algorithm 4.1. Randomized algorithm for the NAE2SAT problem

Function NAE2SAT-SOLVE($G_\phi(x)$)
1: {The 2CNF formula $\phi(x)$ is input through its implication graph $G_\phi(x)$.}
2: {We assume that $\phi(x)$ has n variables and m clauses, so $G_\phi(x)$ has $2 \cdot n$ vertices and $2 \cdot m$ arcs.}
3: Let T be an arbitrary truth assignment to the variables of $\phi(x)$.
4: Update $G_\phi(x)$ with T.
5: {We say that an arc in $G_{\phi(x)}$ is broken, if under the current assignment it is a **true**→**false** arc or a **false**→**true** arc.}
6: $count = 0$;
7: **if** (there does not exist a broken arc in $G_\phi(x)$) **then**
8: **return** ("$\phi(x)$ is nae-satisfiable")
9: **end if**
10: **while** (there exists at least one broken arc in $G_\phi(x)$) **and** ($count \leq \frac{9}{4}n^2$) **do**
11: Select any broken arc in $G_\phi(x)$, say $x_1 \rightarrow x_2$.
12: Flip a fair coin to pick one of x_1 and x_2.
13: **if** (x_1 is selected) **then**
14: Flip x_1; i.e., complement its assignment.
15: **else**
16: Flip x_2.
17: **end if**
18: Adjust T and $G_\phi(x)$ to reflect the changed assignments.
19: **if** (T now satisfies $\phi(x)$. i.e., there is no broken arc in $G_\phi(x)$) **then**
20: **return** ("$\phi(x)$ is nae-satisfiable.")
21: **else**
22: $count = count + 1$.
23: **end if**
24: **end while**
25: **return** ("$\phi(x)$ is probably nae-unsatisfiable.")

4.1 Analysis

Observe that if Algorithm 4.1 claims that $\phi(x)$ is nae-satisfiable, then $\phi(x)$ definitely has a nae-satisfying assignment, i.e, the assignment T, which causes the algorithm to terminate. On the other hand, if Algorithm 4.1 claims that $\phi(x)$ is not nae-satisfiable, then it is possible that $\phi(x)$ is still nae-satisfiable; we now show that the probability of this occurrence is less than $\frac{1}{6}$.

Lemma 3. *Let $\phi(x)$ denote a 2CNF formula; if assignment $\mathbf{x}$ nae-satisfies $\phi(x)$, then so does assignment $\mathbf{x^c}$, where $\mathbf{x^c}$ is derived from $\mathbf{x}$, by complementing the assignment to each variable in $\mathbf{x}$. The tuple $(\mathbf{x}, \mathbf{x^c})$ is called a complementary pair.*

Proof: Since, $\mathbf{x}$ nae-satisfies $\phi(x)$, it sets one literal to **true** and one literal to **false** in each clause. Under $\mathbf{x^c}$, the literals which are set to **true** become **false** and vice versa. It follows that each clause is still nae-satisfied and therefore, so is $\phi(x)$. □

Assume that $\phi(x)$ is nae-satisfiable and let us focus on a particular nae-satisfying complementary pair of assignments $\hat{T}$ and $\hat{T}^c$. Let T denote the current assignment to the variables of $\phi(x)$. If T is a nae-satisfying assignment, we are done. If it is not, then there is a clause, say (x_i, x_j) that is not nae-satisfied by T. There are two cases to consider:

(i) Both x_i and x_j are set to **false** in T - In $\hat{T}$, at least one of these two literals is set to **true**; likewise, in $\hat{T}^c$, at least one of these literals is set to **false**. Hence choosing one of them uniformly and at random, moves T closer to $\hat{T}$, with probability at least one-half and closer to $\hat{T}^c$ with probability one-half. In other words, after the literal flip, with probability one-half, T agrees with $\hat{T}$ in one more variable and with probability one-half, T agrees with $\hat{T}^c$ in one more variable.

(ii) Both x_i and x_j are set to **true** in T - In $\hat{T}$, at least one of the two literals is set to **false** and in $\hat{T}^c$, at least one of the two literals is set to **true**. Hence choosing one of the two literals uniformly and at random and flipping it, moves T closer (by one variable) to $\hat{T}$ with probability at least one-half and closer to $\hat{T}^c$ (by one variable) with probability at least one-half.

Let $r(i)$ denote the expected number of literal-flips for Algorithm 4.1 to take the current assignment T to the nae-satisfying assignment $\hat{T}$ or its complement $\hat{T}^c$, assuming that T differs from $\hat{T}$ on exactly i variables. By our previous arguments, it is clear that T differs $\hat{T}^c$ in exactly $(n-i)$ variables. Note that $r(0) = 0$, since if the current assignment differs from $\hat{T}$ on 0 variables, then it is a nae-satisfying assignment. Likewise, $r(n) = 0$, since if the current assignment differs from $\hat{T}$ on all n variables, then it must differ from $\hat{T}^c$ on exactly 0 variables, i.e., T must coincide with the nae-satisfying assignment $\hat{T}^c$.

Applying Lemma (1) to the discussion above, for each i, $0 < i < n$, we must have

$$\begin{aligned} r(i) &= \frac{1}{2}\left(r(i-1)+1\right) + \frac{1}{2}(r(i+1)+1) \\ &= \frac{1}{2}r(i-1) + \frac{1}{2}r(i+1) + 1 \end{aligned} \tag{4}$$

But System (4) in conjunction with the boundary conditions is precisely the defining system of an ASR walk $\mathcal{R}$. We therefore conclude that:

Theorem 10

$$\max_{0\le i\le n} r(i) \le \frac{n^2}{4}.$$

$$\max_{0\le i\le n} \mathbf{Var}[r(i)] = \frac{2}{3}n^4.$$

Corollary 1. *Algorithm 4.1 is a Monte-Carlo algorithm for the NAE2SAT problem; on a nae-satisfiable instance, the probability that it does not find a nae-satisfying assignment is at most* $\frac{1}{6}$.

5 Graph Bipartiteness

In this section, we apply the techniques of Algorithm 4.1 to derive a Monte Carlo algorithm for the problems of checking whether an undirected graph is 2-colorable. Without loss of generality, we assume that the vertices of the graph need to be colored from the set {**red, blue**}. A particular coloring **c** for the vertices of G, is inconsistent if there exists at least one edge e such that both its endpoints are colored **blue** or **red**. If no such edge exists, **c** is said to be a consistent (valid) coloring.

Algorithm 5.1. Randomized algorithm for the Undirected Graph 2-coloring problem.

Function GRAPH-2-COLOR(G)
1: {We assume that G has n variables and m edges.}
2: Let **c** be an arbitrary color assignment of **red** and **blue** to the vertices of G.
3: $count = 0$;
4: **if** (**c** is a valid coloring) **then**
5: **return** ("G is 2-colorable.")
6: **end if**
7: **while** ((**c** is an inconsistent coloring)**and** ($count \leq \frac{9}{4} \cdot n^2$)) **do**
8: Pick an edge $e = (x_i, x_j) \in E$ such that $c[x_i] = c[x_j] =$ **red** or $c[x_i] = c[x_j] =$ **blue**
9: Flip a fair coin to pick one of x_i and x_j.
10: **if** (x_i is selected) **then**
11: Change its color to **blue**, if it was **red** and to **red**, if it was **blue**.
12: **else**
13: Change the color of x_j to **blue**, if it was **red** and to **red**, if it was **blue**.
14: **end if**
15: Update **c** accordingly.
16: **if** (the current color assignment is valid) **then**
17: "G is 2-colorable")
18: **else**
19: $count = count + 1$.
20: **end if**
21: **end while**
22: **return** ("G is probably not 2-colorable.)

Algorithm 5.1 is a Monte Carlo algorithm for checking whether a graph is bipartite.

5.1 Analysis

The graph bipartiteness problem shares an important property with the NAE2SAT problem, in that if a particular coloring **c** is a valid 2-coloring of a graph, then so is its complement coloring $\bar{\mathbf{c}}$. Clearly if Algorithm 5.1 returns "yes", then the input instance G is bipartite. However, if the algorithm claims that G is not 2-colorable, then it could be incorrect. The error bound analysis is identical to the one for NAE2SAT, since at each step, Algorithm 5.1 moves one step closer to a valid coloring or one step closer to the complement of that valid coloring. Algorithm 5.1 can therefore also be modeled as

a one dimensional random walk with one reflecting barrier and one absorbing barrier. Accordingly, we get,

Theorem 11. *Algorithm 5.1 is a Monte Carlo algorithm for the problem of checking whether an undirected graph is 2-colorable. If the input graph is not 2-colorable, the algorithm always returns the correct answer; if the input graph is 2-colorable, the probability that the algorithm returns the incorrect answer is at most* $\frac{1}{6}$.

An interesting offshoot of the above work is the following theorem.

Theorem 12. *Let NAE2SATPOS denote the class of NA2SAT problems in which every literal is positive. NAE2SATPOS is* L-complete.

Proof: First observe that NAE2SATPOS is trivially in L, since it is a special case of NAE2SAT. Likewise, it has already been established that Undirected Graph 2-coloring (UG2COL) is in SL and therefore in L [3,8,4]. Now, consider the following $\mathtt{AC}_0$ reduction from UG2COL to NAE2SATPOS.

Let $G = \langle V, E\rangle$ denote an instance of the UG2COL problem, where, $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_{ij}$: there is an undirected edge between vertices v_i and $v_j\}$.

We construct the following instance of NAESATPOS:

(a) Corresponding to vertex v_i, the variable x_i is created.
(b) Corresponding to edge $e_{ij} = (v_i, v_j)$, the clause (x_i, x_j) is created.
(c) The conjunction of all the clauses gives us the 2CNF formula $\phi(x)$.

It is not hard to see that the graph G has a 2-coloring if and only if $\phi(x)$ is NAE-satisfiable. □

This result can be seen as an addition to the collection of results in [9].

6 Conclusion

In this paper, we designed and analyzed a randomized, literal-flipping algorithm for the NAE2SAT problem. As mentioned before, the existence of a polynomial time randomized algorithm for this problem is not surprising, since NAE2SAT belongs to the complexity class $\mathtt{SL} \subseteq \mathtt{P} \subseteq \mathtt{RP}$. The interesting aspect of our work is the simplicity of the randomized algorithm and its analysis. We extended the algorithm to check for bipartiteness in undirected graphs.

References

1. Papadimitriou, C.H.: On selecting a satisfying truth assignment. In: IEEE (ed.) Proceedings: 32nd annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, October 1–4, pp. 163–169. IEEE Computer Society Press, USA (1991)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman Company, San Francisco (1979)
3. Reif, J.H.: Symmetric complementation. J. ACM 31(2), 401–421 (1984)

4. Reingold, O.: Undirected st-connectivity in log-space. In: STOC, pp. 376–385 (2005)
5. Aspvall, B., Plass, M.F., Tarjan, R.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters 8(3), 121–123 (1979)
6. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
7. Ross, S.M.: Probability Models, 7th edn. Academic Press, Inc., London (2000)
8. Àlvarez, C., Greenlaw, R.: A compendium of problems complete for symmetric logarithmic space. Electronic Colloquium on Computational Complexity (ECCC) 3(39) (1996)
9. Johannsen, J.: Satisfiability problems complete for deterministic logarithmic space. In: STACS, pp. 317–325 (2004)

Versioning Tree Structures by Path-Merging

Khaireel A. Mohamed, Tobias Langner, and Thomas Ottmann

Albert-Ludwigs-Universität Freiburg, D-79110 Freiburg, Germany
{khaireel,langneto,ottmann}@informatik.uni-freiburg.de

Abstract. We propose path-merging as a refinement of techniques used to make linked data structures partially persistent. Path-merging supports bursts of operations between any two adjacent versions in contrast to only one operation in the original variant. The superiority of the method is shown both theoretically and experimentally. Details of the technique are explained for the case of binary search trees. Path-merging is particularly useful for the implementation of scan-line algorithms where many update operations on the sweep status structure have to be performed at the same event points. Examples are algorithms for planar point location, for answering intersection queries for sets of horizontal line segments, and for detecting conflicts in sets of 1-dim IP packet filters.

Subject Classifications: E.1 [**Data**]: Data Structures – *trees*; E.2 [**Data**]: Data Storage Representations – *linked representations*; F.2.2 [**Analysis of Algorithms and Problem Complexity**] Nonnumerical Algorithms and Problems – *Geometrical problems and computations.*

Keywords: Partial persistence, path-merging, path-copying, node-copying.

1 Introduction

A data structure supporting access to multiple versions is called a *persistent* data structure, and to date, there are various problems in computer science where such structures are often sought after. This is mainly due to their elegance of maintaining a historical list of the ever-changing primary structure through efficient *update operations*, and then providing convenient ways to get to the archived data by means of well-designed *access operations*.

There are mainly two different degrees of persistence; partial and full. A *partially persistent* structure allows only read access to previous versions, while a *fully persistent* structure allows write access to earlier versions, on top of the read access. In this article, we shall concentrate on the former degree of persistence and discuss the two well-known, classical methods of making linked data structures persistent, namely the 'path-copying' method and the 'node-copying' method. Our perusal of the two methods shall be applied primarily onto binary search tree (BST) structures.

As their names suggest, the path-copying method reproduces an entire path in the BST to effect a single update operation, while the node-copying method

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 101–112, 2008.

copies only single nodes (one node in amortized average) per update operation. An update operation creates a new persistent version in the BST. But ever so often, in many domain specific applications, we find that we do not actually require every single one of these versions. Suffice to keep only those versions that we find essential and remove all others as they play no significant part in the broader view of the application it ministers. However, we cannot simply delete those non-essential intermediate versions, as in a partially persistent BST, nodes in one version may share subtrees belonging to other versions.

Hence, we introduce our *path-merging* technique to show how we can comprehensively collate and properly link the non-essential intermediate versions to keep only those versions we think are essential. The correctness of our technique leads to the cost savings in both time and space when compared to the original methods it overlays.

2 Implications of Access Operations in Partially Persistent Structures

Following a series of successful *update* operations on a partially persistent linked data structure τ, we can acquire and assemble a previously persisted version with an *access* operation. For the purpose of our deliberations, we shall exploit τ as a partially persistent binary search tree (BST), where an access operation refers to a search for an item or items in τ, at some past version v_i given a query object q. The accessed set forms a path in τ that starts at the root at v_i, and is extended one node at a time, ensuing an *access heuristic* until the desired items matching the query q are found. Particularly, we shall discuss the access heuristic of Sarnak and Tarjan's *path-copying* method [1] and Driscoll *et al.*'s *node-copying* method [2], which will lead to the discourse of our *path-merging* method later on in this article.

Let us first observe the importance of access operations in persistent data structures when they are used to support surrogate applications.

2.1 Planar Point Location

The key to efficiently solve the planar point location problem is to build a systematically organized data structure to represent a planar subdivision S of n edges. In order to report the face f of S that contains a given query point q, de Berg *et al.* [3] showed how to decompose S into a trapezoidal map $T(S)$, which uses $O(n)$ space and answers the query in $O(\log n)$ time.

An alternative method introduced by Sarnak and Tarjan [1] is to use a partially persistent RB-BST as an improvement over Cole's [4] persistent representation of sorted sets. This also answers the same query in $O(\log n)$ time. The space consumption, however, depends on the method of persistence; the path-copying method takes up $O(n \log n)$ space, while the node-copying method requires $O(n)$ storage.

Vertical lines are drawn through each vertex in S which split the plane into $O(n)$ slabs as shown in Fig. 1(a). Each slab contains the edges of S, which are

associated to the faces just above them, ordered from bottom to top. An RB-BST τ is first built for the left-most slab to hold the sorted edges. Afterwhich, a left to right sweep is carried out on all the slabs, stopping at each vertical line and persisting τ by creating one new version for every update operation. The x-coordinate value of the vertical line is also augmented to the top-most version pointer during the update operation, and these are indexed in another balanced BST on a higher level. This vertical partitioning of the subdivision gives us exactly $2n$ update operations; where at every vertical line, one edge is deleted at version v_i and one other edge is inserted at version v_{i+1}.

Thus, to locate the face f in which q lies, we first perform an $O(\log n)$ time search on the upper level BST to locate the correct version of τ corresponding to the x-coordinate of q, and then access τ at version v_{q_x} to perform a second $O(\log n)$ time search using the y-coordinate of q to determine the correct f.

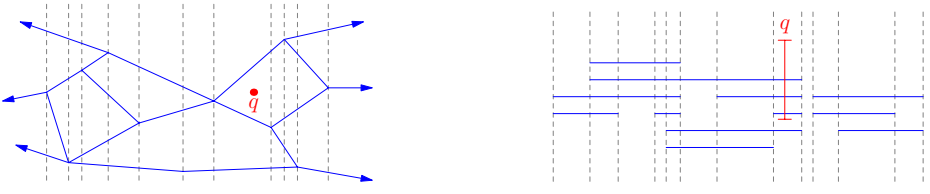


Fig. 1. (a) Planar point location problem. (b) Stabbing range query q on a set of n horizontal line-segments.

2.2 Range Query of Horizontal Line-Segments

We can apply a similar technique as we did before in handling the point location problem with a persistent data structure when we are presented with a set S of n horizontal line-segments. Given a vertical query range q, we are to report all the line-segments in S that intersect q. Again, we draw vertical lines at both endpoints of each line-segment, splitting the plane into at most $2n - 1$ slabs as depicted in Fig. 1(b). Each slab contains the line-segments in S sorted in ascending y order. We then perform an identical sweepline approach to build a partially persistent RB-BST τ on the slabs, such that at every encounter of the vertical line l_i, all line-segments whose left-endpoints match l_i are inserted into τ and all other line-segments whose right-endpoints match l_i are deleted from τ. Every single one of these update operations creates a new version of τ, and unlike the slabs in the point location problem, we tend to face an arbitrarily many insertions and deletions per vertical line as we transit between adjacent slabs during the sweep.

Using the resultant partially persistent structure, we can report the solution in time $O(\log n + k)$, where k is the number of line-segments in S intersecting the vertical range q. Like before, we first execute a binary search to access the correct version v_{q_x} of τ where q lies, and then perform a range query on τ at v_{q_x} to return the active line-segments that intersect q.

2.3 Essential and Non-essential Versions

Clearly, as evidenced by the two examples above, it is not necessary to persist every single version, every time we perform an update operation. It is sufficient to store only those versions that are the collective results of multiple update operations after completely handling an event point. In other words, the versions that we persist must be substantially *essential*, such that all other versions leading to an essential version will have no effect on the overall correctness of the application that τ serves. Thus, we should be able to omit the *non-essential* versions in the partially persistent τ without breaking the temporal flow of the essential versions within.

3 Merging Non-essential Versions in Partially Persistent BSTs

Our problem involving the partially persistent data structures laid out in the previous section is what Driscoll *et al.* and Sarnak and Tarjan [2,1] termed the "persistent sorted set" problem. Here, we maintain a set of elements that changes over time, where each element has a distinct key that is comparable to all other keys in the elements in the same set, such that these keys can be totally ordered. The BST τ is a structure that represents such a set, where it contains one element per node arranged in a symmetric ordering.

We begin by characterizing the different types of nodes that can exist during the intercession of two adjacent essential versions of τ. As we collate the series of non-essential versions effected on τ by the corresponding series of intermediate update operations, we need to distinguish the set of *ephemeral* nodes from the set of *persistent* nodes. They tend to appear simultaneously in τ amidst this transition period, but always in a some formal ordering.

An 'ephemeral node' is a node created during an intermediate update operation and is ephemerally modifiable until it becomes a persistent node, or until it is deleted. On the other hand, a 'persistent node' is a *versioned* node belonging to an existing persistent version v_i of τ, and that any modification on it is strictly not allowed.

Let v_{m-1} be the latest essential version of a partially persistent BST τ under our path-merging technique. Let v_m be the next essential version of τ to be spawned. Let all intermediate versions contributing to the non-essential versions of τ between v_{m-1} and v_m be v_m^*. Then τ at version v_m^* is a 'semi-ephemeral' BST containing both ephemeral nodes and persistent nodes. Note that v_m^* may change over time.

3.1 Path-Merging Via Path-Copying

Sarnak and Tarjan [1] compiled from several sources and presented the idea of the path-copying method to make a linked data structure persistent. During an update operation on τ, we copy only those nodes that are effected by the said operation and percolate the copying procedure to any node with a direct pointer

to the copied nodes. Consequently, if τ is a BST, then entire paths from the effected nodes to the roots are copied, creating a set of search trees for all the partially persistent versions of τ, having different roots per version but sharing common subtrees.

In our *path-merging* technique, only the first update operation that immediately follows the successful persistence of the latest essential version v_{m-1}, adheres to the original path-copying method. This effectively gives us a new semi-ephemeral BST rooted at v_m^*. The newly copied path forms a set of linked ephemeral nodes in v_m^*. All other nodes linked from the subtrees of the ephemeral nodes in v_m^* belong to other partially persistent versions of τ, and they make up the set of persistent nodes in v_m^*. All subsequent update operations contributing to the non-essential versions of τ shall begin with a search at the root at v_m^*. Note that each update will change v_m^*.

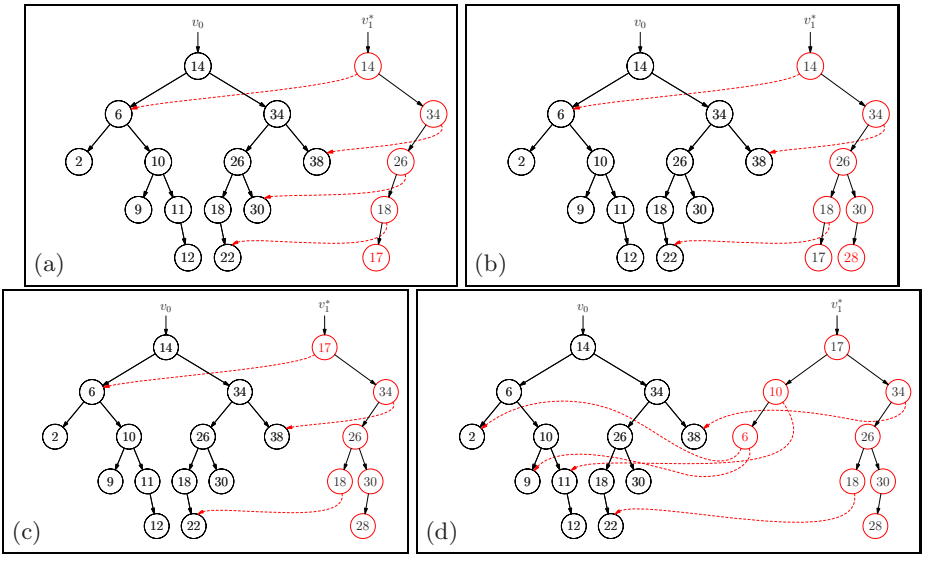


Fig. 2. Path-merging via path-copying. Version v_0: Insert {14, 6, 34, 38, 26, 10, 2, 18, 30, 22, 11, 12, 9}. Version v_1: {Insert {17}, Insert{28}, Delete {14}, Rotate-Left {6}}, showed in sequence from (a) to (d), respectively. The BST rooted at v_1^* during the intercession is a semi-ephemeral structure. Red nodes are ephemeral nodes, and black nodes are persistent nodes.

Let x be a node in v_m^* in which an update operation will be effected upon, and let $c(x)$ denote an ephemeral copy of the node x. Then we only need to copy the persistent nodes in v_m^* if our search for x breaks away from the traversal of the path of ephemeral nodes. We note here that a newly created copy of a persistent node is an ephemeral node.

The next two rules complete the handling of the path-merging technique once we have identified the node x:

1. If x is an ephemeral node, then we treat the update operation on x as a normal ephemeral instruction, overriding the effects of a previous operation made on x.
2. If x is a persistent node, then we perform the update operation on $c(x)$.

Fig. 2 shows an example of path-merging four successive intermediate update operations between v_0 and v_1, with each sub-figure showing an intermediate update operation.

After a series of i intermediate update operations, the "net" set of ephemeral nodes in v_m^* is the result of merging i non-essential versions of the original path-copying method of persistence. What is left to be executed in persisting this set of merged paths for the next essential version of τ is to set the status of all the ephemeral nodes in version v_m to 'persistent'. Since the set of ephemeral nodes in v_m is a connected subtree at the root, we can carry out this change of status in time proportional to the number of ephemeral nodes in v_m using a simple depth-first traversal.

3.2 Path-Merging Via Node-Copying

The node-copying method was conceived to eliminate the shortcomings of the naïve fat node method. Where there can be arbitrarily many outgoing pointers in a fat node in a persistent structure, a node following the node-copying method is allowed to have only a fixed number of such pointers.

In fact, Driscoll *et al.* [2] showed that the improved node for the persistent BST τ needs to store, apart from its key element, only three obligatory pointers: one left pointer, one right pointer, and one other *modification* pointer, each with a version stamp. When such a node becomes full, we create a new copy of the node containing only the newest value of each pointer field. Also, as was with the case of the path-copying method, every time we copy an existing node, its predecessor must be informed of the change. In this case, the parent of the copied node must update its modification pointer to point to the newly copied node and accorded the newest version stamp. But if the parent is also full, then the parent too, must be copied. This copy-percolation process may end up with the root itself being copied.

Accessing Persisted Versions. Unlike the path-copying method where every update operation always produces a new root, the node copying method tends to be more conservative in its expansion of the main tree structure. Every update operation leaves a distinct version stamp in the pointers to the nodes *inside* τ that are effected by the operation. Thus, traversing a persistent BST τ made by the node-copying method must abide by the following access heuristic (which is similar to the access heuristic of the fat node method):

1. Find the correct root for a given version v_i.
2. Traverse the nodes by choosing only pointers with the maximum version stamp that is less than or equal to v_i.

Intermediate Update Operations. As before, let x be a node in v_m^* in which an update operation will be effected upon, and let $c(x)$ denote an ephemeral copy of the node x.

We impose a slight variant on the original node-copying procedures in our path-merging technique when an update operation contributes to a non-essential version. Navigating and manipulating this conservative structure of τ at version v_m^*, influenced by the node-copying method, requires a different kind of attention to be paid when managing the internal nodes. This is not as forthright as compared to the more discernible arrangement of ephemeral and persistent nodes in τ created by the path-copying method. That is, whenever we access τ at v_m^* to search for the node x, we may end up retrieving a path from the root to x that contains both ephemeral and persistent nodes, in random sequence! Furthermore, the nodes in τ have an additional modification pointer field that exhibits special properties when we apply our path-merging technique via the node-copying method.

Hence, here, we extend the notions of our earlier terminologies so as to apply them to the context of the node-copying method, in order to handle the path-merging procedure efficiently.

At one end of the access-spectrum, we have the persistent nodes. A 'persistent' node is a versioned and *full* node in τ belonging to an existing essential version, and is strictly unmodifiable. By *full*, we mean that this persistent node has its key and all three of its pointer fields, particularly the modification pointer, assigned to objects (even to a null object). On the opposite end of the spectrum, lies the ephemeral nodes. An 'ephemeral' node in τ is always created during an intermediate update operation at version v_m^*, either as a new entity or as an ephemeral copy of an existing persistent node. All the contents in this ephemeral node are ephemerally modifiable, and remain so until the node is deleted, or until the node is versioned at v_m. What is now left to be considered are those nodes that are in the middle of the spectrum, and they fit neither of the two descriptions above. We shall call them *semi-persistent* nodes.

An ephemeral node becomes a 'semi-persistent' node, if and only if its modification pointer is empty at the time of spawning a new essential version of τ. In other words, only its key and its left and right pointers can be versioned. The modification pointer, left untouched, is ephemerally modifiable by any future update operation, and remains so as long as the node is in transition. Furthermore, the semi-persistent node becomes a persistent node if at the next essential version v_m the modification pointer is no longer empty.

Given an intermediate update operation, we first invoke the access heuristic on τ at version v_m^* (or at version v_{m-1} if v_m^* does not yet exist) and traverse τ until we arrive at the node x in which to effect the operation. We then execute the node-copying method on x implicitly, while explicitly adhering to an additional set of rules when administering any nodal changes.

Rule 1. If x is an ephemeral node, then we treat the update operation on x as a normal ephemeral instruction, overriding the effects of a previous operation made on x.

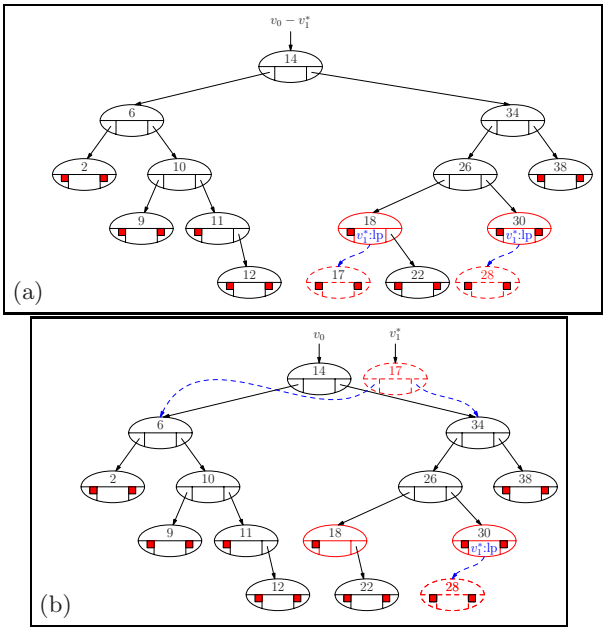


Fig. 3. For example, deleting {14} from (a) results in (b), where the dotted red nodes are ephemeral nodes and all others are semi-persistent nodes. Note that {17} was deleted in (a) without consequence following Rule 1 below, and a copy of the root was made in (b) following Rule 3(a).

Rule 2. If x is a persistent node, then we perform the update operation on $c(x)$.
Rule 3. If x is a semi-persistent node, then we react according to the update operation as follows:

(a) If the update operation changes the key in x, then we perform the change of key in $c(x)$.
(b) If the update operation changes the modification pointer of x *without* causing a *node-contention*, then we simply execute the change. We give a further explanation of what a node-contention is in the next sub-section.
(c) If the update operation changes the modification pointer of x *and* causes a node-contention, then we make a copy of x and resolve the conflict between x and the update operation, in the new $c(x)$.

In addition to the rules above, the pointers in every intermediate update operation effecting or effected by x shall carry the version stamp v_m^*. Fig. 4 depicts an example of path-merging the same four successive intermediate update operations as performed in the previous section.

After a series of i intermediate update operations, the "net" set of ephemeral nodes is again the result of merging i non-essential versions of the original node-copying method. The partially persistent BST τ in Fig. 4 underwent exactly the same sequence of intermediate update operations as was in the case of the tree

produced in Fig. 2. The stark difference between both these trees is that the ephemeral nodes in Fig. 4 are sporadically dispersed, rather than being ordered as a proper subtree as we saw in Fig. 2.

Hence, it may require an $O(n)$ effort to locate the ephemeral nodes in τ at v_m^* in order to change their access statuses when spawning the next essential version v_m. One way to counter this problem is not to find them at all in the first place. That is, instead of stamping v_m^* to pointers effecting x when executing intermediate update operations, we simply stamp the identity of the next essential version v_m. Since each node knows which essential version it belongs to, past or future, stamping the version v_m during the merging of non-essential versions holds for our path-merging technique via the node-copying method.

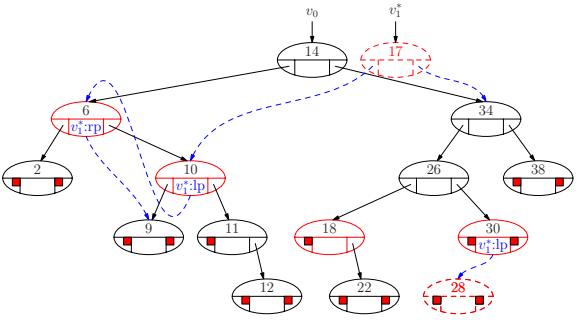


Fig. 4. Path-merging via node-copying. Version v_0: Insert {14, 6, 34, 38, 26, 10, 2, 18, 30, 22, 11, 12, 9}. Version v_1: {Insert {17}, Insert{28}, Delete {14}, Rotate-Left {6}}. Dotted red nodes are ephemeral nodes. Red nodes are 'marked' semi-persistent nodes.

Node-Contentions in Semi-persistent Nodes. A 'node-contention' can occur only in a semi-persistent node during its transition between two essential versions. The contention is caused between a current update operation and the non-empty modification pointer in the node. More specifically, it happens when the modification pointer is already pointing to an object meant to override the node's right (left) pointer, while the current update operation contains an instruction to override the node's left (right) pointer.

When such a case happens, and if we are to replace the modification pointer in favour of the instruction, we then end up with an incorrect routing path in τ. And since we cannot modify the node's original left and right pointers, we resolve this contention by making an ephemeral copy of this semi-persistent node, and directly assign its new left and right pointers from the instruction and from the reference from the original modification pointer. Afterwhich, we delete the modification pointer in the original node to complete the reassignment.

For example, suppose we need to handle one more intermediate update operation in v_1^* in Fig. 4 – to Delete {2}. A search for the node x in v_1^* to effect the delete operation returns the parent of the node {2}, so that x = node {6} and where x is a semi-persistent node. Now, in order to delete {2}, we need the *left* pointer of x to be null. Since x is a semi-persistent node, we can only change

its modification pointer. However, its modification pointer is already assigned to point *right* to node {9}, and that overriding this pointer will be erroneous to v_1^*. Thus, we make a copy of x and resolve the contention by assigning the latest left and right pointers to the new ephemeral $c(x) = \text{copy}(\text{node } \{6\})$, and then remove the modification pointer in the original x. The result is shown in Fig. 5.

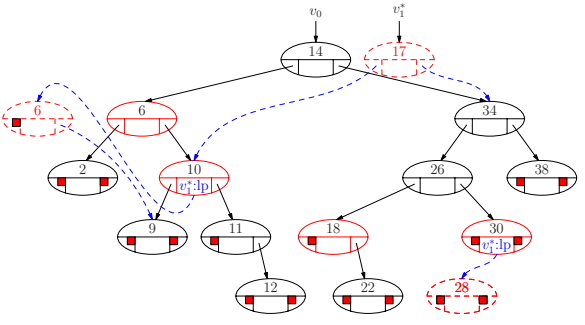


Fig. 5. Resolving a node-contention: Deleting {2} from v_1^* in Fig. 4, where the parent node $\{6\} = x$ was a semi-persistent node, which would have triggered a node-contention if 2 was deleted without checking

4 Analysis of the Path-Merging Technique

The time required for a single update operation in the path-merging technique is the same as the time taken to execute a single update operation by the original underlying methods of path-copying and node-copying [2,1]. However, we note the stern reduction in the overall time by a large factor in the path-merging technique, since no ephemeral nodes are copied more than once.

In terms of space consumption, the path-merging techniques surpass both its predecessors', as it supports bursts of operations between any two essential versions. That is, only the "net" set of ephemeral nodes, which comes from the resultant set of newly created nodes after merging the non-essential versions, contributes to the net increase in space after i intermediate operations. This net increase can even be zero, particularly for the path-merging via node-copying technique, in the case that exactly the same set of keys is inserted into and then deleted from τ several times during the transition period between essential versions. This, compared to the original node-copying method which will end up spawning entire paths after $O(h)$ insertions and deletions of a single key, resulting in $h + (h-1) + \ldots + 1 = O(h^2)$ additional space, where h is the height of the BST τ.

Using the examples in Section 2, we can expect to see a significant reduction in storage space when using the path-merging technique; particularly for the problem of the 'Range Query of Horizontal Line-Segments', where we can anticipate handling arbitrarily many insertions and deletions at an event point. Furthermore, our benchmarked results in Fig. 6 proves the space efficiency between the original path-copying method versus our path-merging technique. For

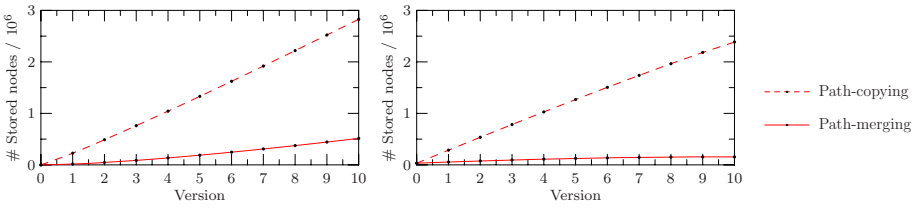


Fig. 6. Space complexity comparison between 'path-merging via path-copying' and the original 'path-copying' methods. Graphs show the execution of 2^{14} operations between v_i and v_{i+1}.

the complete details of the benchmarking process, the reader is invited to check out the work by Langner [5].

5 Conclusion

The pertinence of the path-merging technique reignites the relevance of the applicative components of the classical path-copying and node-copying methods in partially persistent data structures. The technique's strengths lie in their subtle, yet effective ways of merging non-essential versions of the original underlying methods of persistence to derive efficient time and space bounds, that are primed for handling applications where it makes sense to store only the substantially essential versions.

We conclude with a real-world example to prove the usefulness of the path-merging technique. We invite the reader to review our completed works of detecting conflicts in internet router tables [6,7,8], where the 1-dim IP packet filters resemble that of the horizontal line-segments on the plane, similar to the problem discussed in Section 2.2. In the summarized context of the IP-Lookup problem, q is taken to be an incoming packet filter and becomes a stabbing query for the set S of n filters. We need to return the most-specific filter that q stabs. The advantage in this case is two-fold: We were able to solve the conflict detection problem in optimal time of $O(n \log n)$ – *while* building the partially persistent structure; and then utilize the benefits of path-merging's space saving output to store the entire conflict-free set S, which is immediately ready for packet classification.

Now, to appreciate the solution to this problem better, usually, we would require two separate structures to handle the two independent problems of conflict detection and packet classification. But by executing path-merging as described above, we are able to unite them and take advantage of path-merging's adeptness to kill two birds with one stone.

Acknowledgement

This research is funded by the Deutschen Forschungsgemeinschaft (DFG) as part of the research project ,,Algorithmen und Datenstrukturen für ausgewählte diskrete Probleme (DFG-Projekt Ot64/8-3)".

References

1. Sarnak, N., Tarjan, R.E.: Planar point location using persistent search trees. Communications of the ACM 29(7), 669–679 (1986)
2. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. In: STOC 1986: Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, pp. 109–121. ACM Press, New York (1986)
3. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)
4. Cole, R.: Searching and storing similar lists. Journal of Algorithms 7(2), 202–220 (1986)
5. Langner, T.: Using partial persistence to support bursts of operations in IP-lookup. Bachelor Thesis, Albert-Ludwigs-Universität Freiburg (March 2007)
6. Maindorfer, C., Mohamed, K.A., Ottmann, T., Datta, A.: A new output-sensitive algorithm to detect and resolve conflicts in internet router tables. In: INFOCOM 2007: Proceedings of the 26th IEEE International Conference on Computer Communications, May 2007, pp. 2431–2435. IEEE Press, Los Alamitos (2007)
7. Mohamed, K.A., Kupich, C.: An $O(n \log n)$ output-sensitive algorithm to detect and resolve conflicts for 1D range filters in router tables. Technical Report 226, Institut für Informatik, Albert-Ludwigs-Universität Freiburg (August 2006)
8. Kupich, C., Mohamed, K.A.: Conflict detection in internet router tables. Technical Report 225, Institut für Informatik, Albert-Ludwigs-Universität Freiburg (August 2006)

A Linear In-situ Algorithm for the Power of Cyclic Permutation*

Jinyun Xue[1], Bo Yang[1,2,3], and Zhengkang Zuo[1,2]

[1] Provincial Key Lab. of High-Performance Computing , Jiangxi Normal University, Nanchang 330027, China
Jinyun@jxnu.edu.cn
[2] Institute of Software, Chinese Academy of Sciences, Beijing 100080, China
[3] Information School, Jiangxi University of Finance & Economics, Nanchang 330013, China

Abstract. We present and develop a linear in-situ algorithm for the power of a cyclic permutation P^r ($-n< r < n$). Several related algorithms become the special cases of this algorithm. At first, we used an abstract structure, named twin ring, to represent cyclic permutation and derive a simple algorithm to compute P^r on twin ring. Then, the algorithm and the abstract structure twin ring were implemented based on PAR platform that consists of a set of program generating tools. The correctness of the final algorithmic program is based on the data coupling invariant and the assumption of PAR Platform correct. The abstract structure twin ring and the program generating tools take a key role in deriving the simple algorithm for computing P^r. The techniques demonstrated in this paper can be used in developing intricate algorithms.

Keywords: Linear In-situ Algorithm, PAR platform, abstract structure, power of a cyclic permutation, twin ring.

1 Introduction

Cyclic permutation is a specific permutation and a general permutation can be partitioned into some cyclic permutations which have no shared element. It has very important applications in the area of cryptography and wireless radio communication etc[15-16]. There are variant algorithmic problems related with cyclic permutation, say products of permutations, inversion of a cyclic permutation and generating random cyclic permutation, etc[1-9]. Knuth presented two algorithms for production of two permutations and two algorithms for computing inversion of cyclic permutation [4]. One better algorithm for computing the inversion of a cyclic permutation was given by Huang B,C.[3]. Sattolo presented an algorithm to generate random cyclic permutations [6]. All this kind of algorithms did not give clear explanation that shows the process of developing the algorithms and convincing proof of the algorithms. We want to search for an unified algorithm to deal with some algorithms of cyclic permutation and a

* Supported by the National Natural Science Foundation of China under Grant No.60573080, 60773054 and the National Grand Fundamental Research 973 Program of China under Grant No. 2003CCA02800 and the Natural Science Foundation of Jiangxi Province no.0211027.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 113–123, 2008.

suitable representation of cyclic permutation that make the algorithms more simple, easy to understand and to implement. Section 2 describes the preliminaries in developing algorithm; Section 3 gives the approach for deriving a simple algorithm to compute Pr with P represented by twin ring; Section 4 describes the implementation of abstract structure twin ring and the abstract algorithm based on the PAR platform; The comparisons with related works are given in section 5; Section 6 includes concluding remarks.

2 Preliminaries

2.1 Notations about Permutations

If P is a permutation of a finite set B={0, 1, 2, …, n-1}, then P can be viewed as a one-to-one function on the set B. The following is a permutation P on the set{0, 1, 2, 3, 4} :

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 0 & 2 \end{pmatrix} \tag{2.1}$$

Then P.0=1, P.1=3, P.2=4, P.3=0, P.4=2, where "." is used to represent the function application. If we use an array A to represent this permutation, then A[0]=1, A[1]=3, A[2]=4, A[3]=0, A[4]=2. Therefore, the array A represents permutation P if and only if P.x = A[x] for all values of x. This case is marked as A=P in this article.

If P and Q are two permutations on the same set, P*Q can be defined as product of P and Q as follow: (P*Q).x=P.(Q.x).

Define P^0 denotes the identity permutation with P.x=x, $x \in B$;

P^{-1} is the inversion of P with $P^{-1}.(P.x)=x$

for $r \geq 0$, $P^{r+1}=P^r*P$, $P^{-r-1}=P^{-r}*P^{-1}$

Obviously, $P*P^r=P^r*P$ and $P^{-1}*P^{-r}=P^{-r}*P^{-1}$ hold.

The product of permutations, the power of permutations, and the inversion of permutations are all simple concepts in abstract algebra and combinatorial mathematics. However, it is not easy to realize these calculations in computers.

Let array A contain the permutation P on the set B, for every $x \in B$, A[x] = P.x. The A can be viewed as the function on the set B. An algorithm S is needed to transform P in A to P^r. where r is arbitrary integer. The algorithm only uses O(1) extra storage units. The specification of the algorithm S is shown as follows.

$$\{A=P \wedge (-n< r < n)\}S\{A=P^r \wedge (-n< r < n)\} \tag{2.2}$$

If $|r| \geq n$, Obviously, $P^r=P^{n-r}$ holds. Thus, we always assume $|r| < n$.

Any permutation can be viewed as a combination of its cyclic components which have no shared element. For example, equation (1) can be represented as the combination of

$$P=\begin{pmatrix} 1 & 2 & 4 \\ 2 & 4 & 1 \end{pmatrix}\begin{pmatrix} 3 & 5 \\ 5 & 3 \end{pmatrix} \tag{2.3}$$

The power of permutation is equal to combination of powers of these cyclic permutations. Therefore, we restrict our attention to permutations P that are cyclic. And P is assumed to be a permutation on set B.

It is extremely difficult to derive algorithm S which is satisfied with specification (2.2) based on array A directly. Thus, we search for another new representation of P so that the law about computing P^r can be revealed simply. The answer will be given in section 3.

2.2 A Brief Description of PAR[10 - 14]

PAR means PAR method and PAR platform, which consists of the methodology for development of algorithms and programs, specification and algorithm describing language Radl, abstract programming language Apla, a set of rules for specification transformation and a set of automatic transformation tools of algorithms and programs. PAR provides powerful generic structures that support convenient generic programming and the methodology that supports formal derivation of executable algorithmic programs from their formal specification. PAR can be used to develop correct application program that access to database and to develop software components with high reliability.

2.2.1 Algorithm Design Language Radl

Radl was designed for the description of algorithm specifications, transformation rules for deriving algorithms and algorithms itself. We presented a set of abstract notations for expressing pre-defined data type, say array, set, list, binary tree, graph and table, etc. Radl provides a user-defined mechanism for abstract data type. The motivation of developing these mathematics-oriented notations is aimed at making specification transformation, algorithm derivation and program proof like operating traditional mathematical formula.

2.2.2 Apla Language and Its ADT Mechanism

Apla is an object-based abstract programming language with convenient generics. The purpose of developing Apla is to implement functional abstract and data abstract in program development perfectly so that any Apla program is simple enough and is ease for understanding, formal derivation or proof. It is also ease to transform into some OOP language programs, say C++, Java, Delphi and VB, etc.. Apla and Radl have same standard procedures and functions. The pre-defined data types and user-defined ADT mechanism are also same. The relational algebra is embedded into Apla and Radl that make easy to access relational database. We borrow some control structure from Dijkstra's Guarded Command Language, but restrict the nondeterminism.

The following is the ADT mechanism:

```
define ADT  <ADT name>(<list of generic parameters> );
  type <ADT name> = private;
  ……..
enddef;

implement ADT <ADT name>(<list of generic parameters> );
 type <ADT name> = concrete data type;
…….
endimp;
```

Generic programming is parameterized programming. In Apla and Radl, PAR supports data value, data type and subroutine as parameter of procedure, function and abstract data type. Generic programming makes programming simpler and increases obviously the reusability, security and reliability of programs.

2.2.3 PAR Platform

Radl algorithms and Apla programs are simple enough and ease for formal derivation and proof. But, they can not be executed in a computer. Therefore, we developed the PAR platform that consists of 5 automatic transformation tools of algorithms or programs. One of them would be able to transform a Radl algorithm into Apla program. Others may transform Apla programs to the programs of target language that is linked to some database, says VB, Delphi, C++ and Java, etc. Sometimes, we call the tool as program generator. Based on the PAR platform, the efficiency of developing algorithmic program and reliability of the programs are increased obviously.

3 Computing P^r with P Represented by Twin Ring T

A cyclic permutation can be represented by a sequence (consisting of elements of the domain of P) in which the successor of any element x in the sequence is the value P.x (the successor of the last element of the sequence is first) . For example, the two cyclic permutations in (1) are

$\begin{pmatrix} 0 & 1 & 3 \\ 1 & 3 & 0 \end{pmatrix}$ and $\begin{pmatrix} 2 & 4 \\ 4 & 2 \end{pmatrix}$. They can be represented as (0 1 3) and (2 4). Obviously, such representations are not exclusive; the above cyclic permutations can also be represented as (1 3 0), (3 0 1), (4 2) and so on. Feijien et el. Called the sequence as ring[1]. In a given ring (XxY), the follower of every element x is P.x.

Let P be a cyclic permutation. Based on the definition of inversion of a permutation and the properties of a cyclic permutation, we have following lemma:

Lemma 1. IF P is a cyclic permutation, then P^{-1} is a cyclic permutation too and $P^{-1}.x = P\text{-}x$ where $x \in B$, P.x denotes the successor of x and P-x denotes the antecedent of x.

Consider that sequence K is a ring representation of a cyclic permutation P. If every element in K is rotated one position to the left, then another ring representation of the p is gotten, that is denoted as sequence H. In[9], we represented a cyclic permutation P as T, a two-line scheme:

$$P = T = \begin{pmatrix} K \\ H \end{pmatrix} \text{ or } P=T=(H:K). \tag{3.1}$$

Based on this representation, we defined the function rotL.k;k to represent the value of the permutation that was produced by rotating sequence k one position to the left, and had P^3=rotL^3.K:K., Thus we got a simple algorithm to compute P^3.

After further research on the two-line scheme of a cyclic permutation, we found more interesting properties about it. We define the function rotR.K;K to represent the value of the permutation that is produced by rotating sequence K one position to the right, and had a more interesting result: P^{-1}=rotR.K:K., Thus we got a simple algorithm

to compute P^{-1} that is the inversion of a cyclic permutation. This inspire us to give the definition of function $rotR^r.K:K$ and computation of P^{-r}. For describing the properties of the two-line scheme of a cyclic permutation precisely and simply, we renamed it as the *twin ring* where the top line is a ring K that represents the domain. The bottom line is a ring H that gives its rang. We always use linear notation P=(H:K) to represent twin ring T.

Following examples can help us to understand function rotL.s and rotR.s.

Let s=(0,1,2,3,4), then rotL.s=(1,2,3,4,0), rotR.s=(4,0,1,2,3).

$rotL^{i+1}.s$ is defined as $rotL.(rotL^i.s)$; $rotR^{i+1}.s$ is defined as $rotR.(\ rotR^{i+1}.s)$, where $i\geq 0$, $rotL^0.s=rotR^0.s=s$.

With the twin ring representation of a cyclic permutation P, following interesting properties can be verified:

$P^2=rotL^2.K:K$ $P^3=rotL^3.K:K$ $P^{-1}=rotR.K:K$ $P^{-2}=rotR^2.K:K$

These properties can be induced into a theorem as follows.

Theorem 1: Let H:K be a twin ring representation of a cyclic permutation p, then for $r\geq 0$, $P^r=rotL^r.K:K$ $P^{-r}=rotR^r.K:K$

Proof: At first, we try to prove $P^r=rotL^r.K:K$

If r=0, $rotL^r.K:K= rotL^0.K:K= K:K= P^r$

Hence the equation $P^r=rotL^r.K:K$ is correct for r=0.

Now if r>0, suppose $P^{r-1}=rotL^{r-1}.K:K$ has been proved, then we have

P^r= {the definition of the power of a permutation}$P*P^{r-1}$

={ Induction postulate }

$P*(\ rotL^{r-1}.K:K)$

={ the definition of multiplication of permutation and the property of a cyclic permutation }

$rotL.(\ rotL^{r-1}.K:K)$

={ The definition of power of Function rotL.s }

$rotL^r.K:K$

Therefore, equation $P^r=rotL^r.K:K$ is correct for any $r\geq 0$.

Similarly, equation $P^{-r}=rotR^r.K:K$ can be proved.

Obviously, we have

Lemma 2: For $0\leqslant r < n$, $P^r = rotL^r.K:K = rotR^{n-r}.K:K = P^{r-n}$.

According to Lemma 2 and for pursuing high efficiency of the algorithm, we compute P^{r-n} for getting the value of P^r with $r\geq n/2$; vice versa. Therefore, let $0\leqslant r\leqslant n/2$ in P^r and P^{-r} .

Based on theorem 1 and for given cyclic permutation and its twin ring represent T=(H:K), for computing T^r, if r>0, we only need to rotate ring H r-1 position to the left; if $r\leqslant 0$, for computing P^r, we only need to rotate ring H $|r|+1$ positions to the right.

Let function rotL(T, r) denote $rotL^{r-1}.H$ and function rotR(T, r) denote $rotR^{|r-1|}.H$, we can get a simple program to computing T^r. The program is described using Apla that is an abstract programming language supported by PAR platform. Apla provides a language mechanism to define abstract data type TwinRing. We will give the implementation of twin ring type TwinRing in next section.

```
program PowerPermutation;
const n=10;
define ADT TwinRing(sometype elem);
  type TwinRing = private;
  function CreateTwinRing(T:TwinRing):TwinRing;
  function rotL(T:TwinRing; r:integer):TwinRing;
  function rotR(T:TwinRing; r:integer):TwinRing;
  procedure TwinRingOutput(T:TwinRing);
enddef;

implement ADT TwinRing(sometype elem);
  type TwinRing = array[0..n-1,elem];
  ……..
endimp.

ADT intTwinRing : new TwinRing(integer);
var T:intTwinRing;
   r:integer;
begin
  {Q:( -n/2≤r≤n/2)∧T=P:intTwinRing}
  {R:( -n/2≤r≤n/2)∧T = P^r}
  writeln("create-twin-ring(must-input-cyclic-permutatio
  n-for-0~",n-1,") : ");
  T:= CreateTwinRing(T); //: Create a cyclic permutation
  in T ://
  readln(r);
  if r> 0→ T:=rotL(T, r);
  [] r≤ 0→T:=rotR(T,-1*r);
  fi;
  TwinRingOutput(T);//: output T^r inT.://
end.
```

We need that the program users only see the twin ring T which is logical view of permutation P. Under this case, users need not to know how to realize this twin ring data type. Therefore, we can define the twin ring a new ADT(abstract data type). And

the new ADT(abstract data type) can be realized by the concrete data type which can be array or other concrete data types. Program users need not to know the concrete datatype, they only need to know how to use the ADT(abstract data type).

In next section, we will define a new ADT which is called twin ring, and it is realized by the concrete data type 'array'. Therefore, the twin ring T does not take any additional memory space, it is only virtual rather practical. The algorithm based on the new ADT is in-situ algorithm for computing P^r.

4 Implementation of Twin Ring Based on PAR Platform

The ADT TwinRing consists of four operations, where function CreateTwinRing (T:TwinRing) and procedure TwinRingOutput(T:TwinRing) provides input and output functions. Their implementations are quite simple. We just pay main attention on the implementation of function rotL(T:TwinRing; r:integer) and function rotR(T:TwinRing; r:integer). According to program specification (2.2) and for developing the in-situ algorithmic program, an array variable A should be used to store a cyclic permutation P and P^{r}. The twin ring type TwinRing should be implemented using an array. Obviously three types of data permutation P, twin ring T and array A satisfies following coupling invariant of data:

$$P.x = T.x = A.x = \text{the successor of x in the ring} \tag{4.1}$$

where x is any element in set B={0, 1, 2, …, n-1}.

Here, twin ring variable T is a *though variable* that can be used to develop and describe algorithm. Based on the thought variable, the derivation of the algorithm becomes quite easy and the algorithm described using twin ring is easy to understand. Therefore, for getting an efficient implementation, we derive two efficient algorithms for implementation of function rotL and rotR based on twin ring firstly, then derive corresponding programs based on array and coupling invariant.

4.1 Derive a Algorithmic Program to Implement rotL(T:TwinRing; r:Integer)

The function rotL(T:TwinRing; r:integer):TwinRing computes T^r, where ($-n/2 \leqslant r \leqslant n/2$) and T is a cyclic permutation of set B= {0, 1, 2, …, n-1}. Let T=(H:K) and K=x1, x2, …, xr, W, where W is a sequence of elements in set B and $W \neq []$. Therefore, we have

T=(H:K)=(x2, x3, …, xr, W, x1):(x1, x2, …, xr, W)

Based on theorem 1, the specification of function rotL can be written as follows:

$$\begin{aligned} &\{Q:\ T=(H:K)=(\ x2, x3, \ldots, xr, W, x1):(\ x1, x2, \ldots, xr, W)\ \wedge(\ -n/2 \leqslant r \leqslant n/2)\} \\ &\quad rotL(T:TwinRing;\ r:integer):TwinRing \\ &\{R:\ T=(H:K)=(\ W, x1, x2, x3, \ldots, xr,):(\ x1, x2, \ldots, xr, W)\ \wedge(\ -n/2 \leqslant r \leqslant n/2)\} \end{aligned} \tag{4.2}$$

There are two methods to satisfy postcondition R. One is following theorem 1, rotate x2, x3, …, xr to left one by one; another method is better than this method in time and space. Following is the Algorithm:

Algorithm 4.1 for computing rotL: For given twin ring T=(H:K), take head element from sequence Wx1, move the sequence x2, x3, …, xr one position to right, put the

head element before x2; repeat the process of taking and putting element in sequence WX1, until the sequence Wx1 is been removed to the position before x2 and Postcondition R is satisfied.

Following is the implementation of the algorithm 4.1.The twin ring variable T:TwinRing is denoted by array variable T: array[0..n-1, elem]. The subscript of array variable T correspond ring K and the value of T corresponds ring H. Let subscripts of array that will be changed be stored in list variable y. The program satisfies the coupling invariant (4.1).

```
function rotL(T:TwinRing; r:integer):TwinRing;
var x,j,temp:integer;
begin
  x,y:=T[0],y↑[T[0]];
  foreach(j:1≤j<r:y:=y↑[T[y[j-1]]];);
    do y[y.t]≠x →
      temp:=T[y[y.t]];
      foreach(j:0≤j<r-1:T[y[y.t-j]]:=T[y[y.t-j-1]];);
      T[y[y.h]]:=temp;
      y:=y[y.h+1..y.t]↑[T[y[y.h]]];
    od;
 end;
```

Let r=3, the function rotL(T, 3) compute rotL T^3 that cube a cyclic permutation.

4.2 Derive a Algorithmic Program to Implement rotR(T:TwinRing; r:Integer)

The function rotR(T:TwinRing; r:integer):TwinRing computes T^r, where $r<0 \wedge 0 \leqslant |r| \leqslant n/2)$ and T is a cyclic permutation of set B= {0, 1, 2, …, n-1}. Let i= |r |, T=(H:K) and K=x ,W, x1 x2, …, xi where W is a sequence of elements in set B and W ≠ [ ]. Therefore, we have

T=(H:K)=(W x1, x2, x3, …, xi, x):(x, W, x1, x2, …, xi)

Based on theorem 1, the specification of function rotR can be written as follows:

$$\begin{array}{l}\{Q:\ T=(H:K)=(\ W,\ x1,\ x2,\ x3,\ \ldots,\ xi,\ x):(\ x,\ W,\ x1,\ x2,\ \ldots,\ xi)\ \wedge\ r<0 \\ \wedge 0\leq|r|\leq n/2\} \\ \quad \text{rotR(T:TwinRing; r:integer):TwinRing} \\ \{R:\ T=(H:K)=(\ x1,\ x2,\ x3,\ \ldots,\ xi,\ x,\ W):(\ x,\ W,\ x1,\ x2,\ \ldots,\ xi\ )\wedge r<0 \\ \wedge 0\leq|r|\leq n/2\ \}\end{array} \tag{4.3}$$

There is also one efficient algorithm to satisfy specification (4.2). Following is the algorithm:

Algorithm 4.2 for rotR: For given twin ring T=(H:K), choose any element x from T, fixed i+1 position starting at x in T, take tail element from sequence xW , move the elements in the fixed i+1 positions and the tail element to left cyclically; repeat the process until each element in sequence xW has been processed and Postcondition R is satisfied.

Following is the implementation of the algorithm 4.2 that is same as algorithm 4.1.

```
function rotR(T:TwinRing; r:integer):TwinRing;
var x,j,b,temp:integer;
begin
  x,y:=T[0],y↑[T[0]];
  foreach(j:1≤j<r+1:y:=y↑[T[y[j-1]]];);
  do T[y[y.t]]≠x → b,temp:=T[y[y.t]],T[y[y.h]];
     foreach(j:0≤j<r:T[y[y.h+j]]:=T[y[y.h+j+1]];);
     T[y[y.t]],T[b]:=T[b],temp;
  od;
end;
```

Let r=1 the function rotR(T , 1) computing T^{-1} that is the inversion of permutation.

Using C++ program generating system tool in PAR platform, the Apla program permutationPower can be transformed to executable C++ program.

5 Compare with Related Works

Cyclic permutation has very important applications in the area of cryptography and wireless radio communication. There are many algorithmic problems and related research about the cyclic permutation.

Knuth presented two in-situ algorithms for computing the inverse of permutation in his famous book [4]. One of them is the algorithm I. Another algorithm is the algorithm J which is due to Boothroyd. It is less obvious that the algorithm J really works. Both of algorithm I and Algorithm J are based on finding cycle and the antecedent of each element in the cycle.

A modified Knuth's in-situ algorithm, called algorithm A, for inversion of permutation was presented by B.C, Huang in [3]. The algorithm is also based on finding the cycles of the permutation and the antecedent of each element in the cycle. The main difference between the two algorithms is that algorithm A is a little shorter than algorithm I. The common shortcomings of above three algorithms are no explanation about the correctness of algorithms and no process of designing algorithm.

In [1], Feijen and Gries try to present the explanation and proof of the algorithm A given by B.C. Huang. They used a sequence, called ring, to denote cyclic permutation and get abstract algorithm based on the ring, then a coordinate transformation was used to transform the abstract algorithm to concrete program based on array manually.

Xue and Gries presented a two-line ring representation for a cyclic permutation in [9]. They developed a linear abstract algorithm for cubing a cyclic permutation based on the two-line ring representation. A coordinate transformation was also used to transform the abstract algorithm to concrete program based on array manually. The two-line ring representation of a cyclic permutation was used to produce a convincing proof for Sattolo's algorithm for generating a random permutation [2,6]. Two successful examples show us that the two-line ring representation of a cyclic permutation is superior to the one-line representation.

In this paper, we rename the two-line ring representation of a cyclic permutation as twin ring and reveal more properties about twin ring. We try to use a unified approach to treat the power of a cyclic permutation and present a linear in-suit algorithm to computing P^r. Based on the abstract structure twin ring, we presented and proved theorem 1 for computing P^r and got a very simple algorithm. then defined the abstract data type TwinRing to implementation the twin ring using the language mechanism provided in PAR platform. The theorem 1 and the ADT TwinRing made the algorithm and program for computing P^r quite simple. The correctness of the final algorithmic program is based on the data coupling invariant and assumption of PAR Platform correct. The success in this example gives us more evidence that the twin ring is more suitable for deriving the permutation algorithms than the single ring.

6 Concluding Remarks

Several novel points are deserved summary as follows:

1. We developed a linear in-suit algorithm and executable program for computing the power of a cyclic permutation. Algorithm for computing inversion of permutation and linear in-situ algorithm for cubing a cyclic permutation can be viewed as the special cases of this algorithmic program.

2. The application of the idea of thought variable provides us an efficient approach to develop intricate algorithms. Abstract structure twin ring is thought variable that is different from general abstract data type. It is a dummy variable. Only partial properties are implemented. The twin ring consists of two lists, but we did not implement it using two lists. We just use a general array to denote it. The success in this example gives us more evidence that the twin ring is more suitable for deriving the permutation algorithms than the single ring.

3. PAR platform is very useful in developing intricate algorithms. The platform provides a convenient language mechanism to define and implement ADT, say twin ring type TwinRing. The platform provides predefined composition data type that make programming more simple and reliable. In this paper, the predefined data type *list* can be used like general standard data type.

References

1. Feijen, et al.: In-situ Inversion of a Cyclic Permutation. J. IPL. 24, 11–14 (1987)
2. Gries, D., Xue, J.: Generating a Random Cyclic Permutation. J. BIT 28 (1988)
3. Huang, B.C.: An Algorithm for Inverting a Permutation. J. IPL 10 (1981)
4. Knuth, D.E.: The Art of Computer Programming, vol. 1. Addison-Wesley, Reading (1973)
5. Prodinger, H.: On the Analysis of An Algorithm to Generate a Random Cyclic Permutation. J. Ars Comb. 65 (2002)
6. Sattolo (Sandra).: An algorithm to generate a random cyclic permutation. J. Information Processing Letters. 22, 315–317 (1986)
7. Semple, C., Steel, M.: Cyclic permutations and evolutionary trees. J. Advances in Applied Mathematics 32, 669–680 (2004)
8. Wilson, M.C.: Overview of Sattolo's Algorithm. J. Algorithms Seminar 2002 – 2004, Chyzak, F.(ed.) INRIA, pp.105–108 (2005)
9. Xue, J., Gries, D.: Developing a Linear Algorithm for Cubing a Cyclic Permutation. Sci. Comput. Program. 11(2), 161–165 (1988)
10. Xue, J.: A Unified Approach for Developing Efficient Algorithmic Programs. J. Journal of Computer Science and Technology 12, 103–118 (1997)
11. Xue, J.: Formal Derivation of Graph Algorithmic Programs Using Partition and Recur. J. Journal of Computer Sciences and Technology 13, 553–561 (1998)
12. Xue, J.: A Practicable Approach for Formal Development of Algorithmic Programs. In: Proceedings of the International Symposium on Future software Technology (1999)
13. Xue, J., et al.: Methods of program design. Higher education press (2001)
14. Xue, J.: PAR Method and Its Supporting Platform. In: Proceedings of The First International Workshop on Asian Working Conference on Verified Software (2006)
15. The application in cryptography, http://planetmath.org/encyclopedia/CyclicPermutation.htm http://www.cs.utsa.edu/~wagner/laws/AESkeys.html
16. The application in a wireless radio communication, http://www.patentstorm.us/patents/6999760-claims.html

Multi-bidding Strategy in Sponsored Keyword Auction*

Tian-Ming Bu[1,**], Xiaotie Deng[2,***], and Qi Qi[2]

[1] Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, P.R. China
tmbu@sei.ecnu.edu.cn
[2] Department of Computer Science
City University of Hong Kong
Hong Kong SAR
csdeng@cityu.edu.hk, qi.qi@student.cityu.edu.hk

Abstract. The generalized second price auction has recently become a much studied model for sponsored keyword auctions for Internet advertisement. Though it is known not to be incentive compatible, properties of its pure Nash equilibria have been well characterized under the single bidding strategy of each bidder.

In this paper, we study the properties of pure Nash equilibria of the generalized second price auction when each bidder is allowed to submit more than one bid. This multi-bidding strategy is noted to have been adopted by companies for keyword advertisements on search engines. In consideration of the pure Nash equilibria, we completely characterize conditions on the number of selling slots for a pure Nash equilibrium to exist, assuming all the advertisers are allowed to use multi-bidding strategies or only one advertiser will use a multi-bidding strategy.

Our findings reveal interesting properties of limitation and potentials of the market place of online advertisement.

1 Introduction

Sponsored keyword auction is a brand new type of market models adopted by major search engine companies such as Google and Yahoo. It has become a principal source of revenue for those companies. As its name indicates, such kind of auctions mostly sells the advertising positions for web links displayed along with the search results when a user places a related keyword or a few related keywords to find information on those search engines.

Different advertising positions have different *click-through-rates*, the ratio of the number of clicks on the advertising to the number of appearances of the

* We would like to thank the anonymous referees for constructive suggestions.
** Research is supported by grants of NSF of China (No. 60496321 and No. 90718013) and a grant of 863 program (No. 2007AA01Z189).
*** The work was supported by a grant from CityU (7001989).

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 124–134, 2008.

advertising web links. Some advertising position draws more attentions from users and generates more clicks than others. For this reason, it is named the *position auction* by Varian [1], which is equivalent to the *generalized second price auction* (GSP for short), a term used by Edelman, Ostrovsky and Schwarz [2]. It is the primary protocol for sponsored link auctions to sell the advertising positions.

Under GSP, each advertiser bid for a price per each click and each winning advertiser is allocated exclusively for a position to place its web-link. The positions are sorted in their click-through-rates (commonly assumed to be the same). For K positions to sell, the K highest bidders win them in the corresponding decreasing order of their bidding prices. A winner pays a price per click, which is equal to the bidding per-click-price of the next highest bidder, i.e., the highest bidding price that is lower than its own bidding price.

If there is only one advertising position to sell, GSP is equal to the popular second price auction/Vickrey auction [3] which is a special case of the more general VCG [3,4,5] mechanisms. Therefore, no bidder can gain any advantage by not bidding its own private value for each click. Any protocol with the property that every agent has a dominant optimal strategy to reveal its own private value is called a *truthful* one. It is also often called an *incentive compatible* protocol.

The GSP allocation method seems simple, intuitive and, arguably, fair. However, if there are more than one position to sell, it will no longer be incentive compatible [2,6]. This observation has inspired further studies of GSP, mostly its pure Nash equilibria in a single bidding strategy.

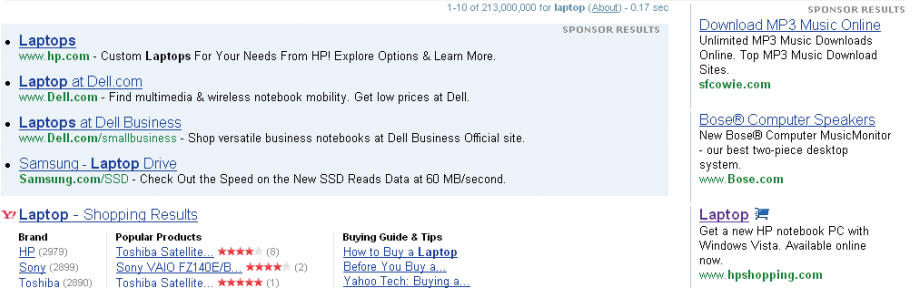


Fig. 1. The sponsor results when keyword "laptop" is typed into the search engine of Yahoo. Both "Dell" and "Hp" adopt the multi-bidding strategy.

In the practice of sponsored keyword auctions, however, bidders may submit multiple bids. For example, when the keyword "laptop" is typed into Yahoo, the sponsor results shown along with the search results page will be similar to the sponsor results shown in Figure 1. In Figure 1, Dell gets two adjacent advertising positions by multiply bids. Although one of the link points to Dell's homepage and the other link points to Dell's sub-homepage, it opens up a possibility for Dell to manipulate these bids to decrease the company's total advertising costs. Similarly, HP also gets two advertising positions in the figure. So this case shows

that i) multi-bidding strategy exists in current online advertisement market, ii) usually only the big companies have the competence and the need for the multi-bidding strategy.

We are particularly interested in the GSP auctions with multi-bidding strategy. We aim at studying the pure Nash equilibrium behavior of the bidders in such market conditions.

1.1 Related Work

The considered model for sponsored search auctions were formalized in [2,1]. In [2], Edelman, Ostrovsky and Schwarz name it *generalized second price auction*, while Varian names it *position auction* in [1]. They all discovered that the auction model is not incentive compatible. Then [2,1,7,8] refined the concept of Nash equilibrium of the position auction and study the related properties. Regarding the auction as a static one-shot complete information game, in [2], Edelman, Ostrovsky and Schwarz introduced the concept of *locally envy-free equilibrium*. In [1], Varian proposed *symmetric Nash equilibrium* due to mathematic considerations. In [7], Zhou and Lukose argued for a certain type of *pure strategy Nash equilibrium*, as a result of some *anti-social behavior*, called *vindictive bidding strategy*.

In [8], Bu, Deng and Qi presented the concept of *forward looking Nash equilibrium* as a result of the auction's dynamics and the bidders' strategic manipulations, based on an important property called the forward looking attribute. Furthermore, in [9] they analyzed the convergence of this dynamic system. Coincidentally, convergence is also studied based on the same bidding strategy by Cary, Das, Edelman, Giotis, Heimerl, Karlin, Mathieu and Schwarz in [10] where it is called the *greedy bidding strategy*.

The concept of *false-name bid* (multiple bids by the same agent under different false names) was firstly introduced by [11] in 1999. In [11], Sakurai, Yokoo and Matsubara observed that *generalized Vickrey auction* mechanism, the generalized version of Vickrey auction, is not robust enough against false-name bid behavior in combinatorial auctions. Then they showed that the concavity of a surplus function over bidders is the sufficient condition where the VCG mechanism is false-name-proof in [12].

Note that the multiple biddings we discuss here are related but not exactly the same as the false name biddings. The agents are submitting their bids under their own true identities where false name biddings are the bids under assumed different identities by the same agent.

1.2 Our Contributions

We study the next technical issue when multiple biddings are allowed. Note that this is different from false name bids in that multiple bidding bidders reveal their true identities but false name bidders do not.

We completely characterize conditions on the number of selling slots for a pure Nash equilibrium to exist, if the advertisers are allowed to use multiple-bidding

strategies. We find that there always exists a pure Nash equilibrium when the maximum allowed number of submitted bids is not less than the number of slots. Otherwise, a pure Nash equilibrium need not exist. Even there is only one advertiser using the multi-bidding strategy, the property of non-existence of pure Nash equilibria still be discovered when the number of multiple bids is greater than 2.

As commented above, when the number of positions to bid for is one, the GSP auction is the same as Vickrey auction which is known to be incentive compatible. When the number of positions is at least two, the GSP auction is no longer incentive compatible even if everyone is allowed for at most one bid. Furthermore, we prove that no other auction protocols selling two or more slots can have the properties of incentive compatibility, social efficiency and individual rationality if multiple biddings are allowed.

Furthermore, we study the most general case when a single bidder may have several advertisements to bid with different private values for each of them. We develop a complete characterization of the existence conditions of equilibrium in this model.

Therefore, in general, we have the existence of Nash equilibrium and the impossibility result in the multiple-bidding market.

The paper is organized as follows. In Section 2 we present the standard GSP model and the extended version with multi-bidding strategy. Section 3 completely characterize conditions on the number of selling slots for a pure Nash equilibrium to exist, if the advertisers are allowed to use multi-bidding strategies. Section 4 discusses the existence of pure Nash equilibria when only one bidder is allowed to use multi-bidding strategy. We present the impossible result in section 5. In Section 6, we discuss the issue of biddings of agents each with multiple advertisement needs of different values.

2 Model and Notation

We follow the GSP auction model presented in [2,1]. For some keyword, there are $\mathcal{N} = \{1, 2, \ldots, N\}$ advertisers who bid $\mathcal{K} = \{1, 2, \ldots, K\}$ advertisement slots ($K < N$). If the indexes of slots satisfy $k_1 < k_2$, then slot k_1's expected *click-through-rate* (CTR for short) c_{k_1} is larger than c_{k_2}. Namely, $c_1 > c_2 > \cdots > c_K > 0$. Moreover, each bidder $i \in \mathcal{N}$ has a privately known information, $v^{(i)}$, which represents the maximum price he is willing to pay for per-click of his advertisement.

According to each bidder i's submitted bid $b^{(i)} \geq 0$, the auctioneer decides how to distribute the advertisement slots among the bidders and how much they should pay for per-click. In particular, the auctioneer firstly sorts the bidders in decreasing order according to their submitted bids. Then the slot with smaller index will be allocated to the bidder with higher bidding value. The last $N - K$ bidders would lose and get nothing. Finally, each winner would be charged for per-click the next bid to his in the descending bid queue. The losers would pay nothing. In the case of ties, we assume that the auctioneer would break

ties according to a prior notice he declares. For example, ties could be broken randomly. Another method of breaking ties is to allocate the higher slot to the bidder with a prior time stamp.

Let b_k denote k^{th} highest bid in the descending bid queue and v_k the true value of the k^{th} bidder in the descending queue. So if bidder i got slot k, i's payment would be $b_{k+1} \cdot c_k$. Otherwise, his payment would be zero. Hence, for any bidder $i \in \mathcal{N}$, if i were on slot $k \in \mathcal{K}$, his utility (payoff) could be represented as

$$u_k^i = (v^{(i)} - b_{k+1}) \cdot c_k \ .$$

2.1 Multi-bidding Model

We consider the extended GSP model associated with the multiple bidding strategy. In other words, each bidder is allowed to submit several bids instead of only one bid. We refer to the extended GSP model as M-GSP if each bidder is allowed to submit at most M bids.

In M-GSP, every bidder i submits at most M non-negative bidding prices to the auctioneer, despite having a unique $v^{(i)}$. We denote bidder i's j^{th} bidding price by $b^{(i,j)}$. Without loss of generality, if the number of submitted bids of i is less than M, the extra dummy bids will be added to make sure that bidder i submits exactly M bids. So for any bidder i, his bidding vector could be written as $\mathbf{b}^{(i)} = \{b^{(i,1)}, \ldots, b^{(i,M)}\}$.

Similarly, if bidder i's j^{th} bidding price is the k^{th} highest among all the bids of bidders, i would be on slot k and the utility of bidding $b^{(i,j)}$ could be represented as

$$u_k^{i,j} = (v^{(i)} - b_{k+1}) \cdot c_k \ .$$

As a result, the total utility of bidder i submitting $\mathbf{b}^{(i)}$ is

$$u^{(i)} = \sum_{j=1}^{M} u^{(i,j)} \ .$$

Additionally, the following lemma states that each bidder would never overbid his true value in the sponsored keyword auctions. Let $\mathbf{b}^{(-i)} = (\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(i-1)}, \mathbf{b}^{(i+1)}, \ldots, \mathbf{b}^{(N)})$. $u^{(i)}(\mathbf{b})$ represents the total utility of bidder i given all the bidders' bidding vector $\mathbf{b}$.

Lemma 2.1. *In M-GSP, $\forall i \in \mathcal{N}$, for any fixed $\mathbf{b}^{(-i)}$, if*

$$\bar{\mathbf{b}}^{(i)} \in \arg \max_{\{\mathbf{b}^{(i)} | b^{(i,j)} \le v^{(i)}, \forall j \in \{1,\ldots,M\}\}} u^{(i)}(\mathbf{b}^{(-i)}, \mathbf{b}^{(i)}) \ ,$$

then

$$\bar{\mathbf{b}}^{(i)} \in \arg \max_{\mathbf{b}^{(i)}} u^{(i)}(\mathbf{b}^{(-i)}, \mathbf{b}^{(i)}) \ .$$

As a result, our model adopts the similar assumption in [13].

Assumption 2.2. (Non-overbidding strategy) In M-GSP, $\forall i \in \mathcal{N}$, $\forall j \in \{1, \ldots, M\}$, $b^{(i,j)} \le v^{(i)}$.

At last, we give the formal definition of pure Nash equilibrium in M-GSP.

Definition 2.3. ***(Pure Nash equilibrium)*** *In M-GSP, the pure Nash Equilibrium is a set of biddings $\widehat{\mathbf{b}} = (\widehat{\mathbf{b}}^{(1)}, \widehat{\mathbf{b}}^{(2)}, \ldots, \widehat{\mathbf{b}}^{(N)})$, in which $\forall i$, $\widehat{\mathbf{b}}^{(i)} \in \arg\max_{\mathbf{b}^{(i)}} u^{(i)}(\widehat{\mathbf{b}}^{(-i)}, \mathbf{b}^{(i)})$.*

In other words, no bidder can benefit from changing any of his or her bids unilaterally. It should be note that since the bidder could decrease one of his bids from some value to 0 or increase one of his bids from 0 to some value, no bidder could benefit even from adding or removing any bids unilaterally in any pure Nash equilibrium.

3 The Existence of Nash Equilibrium

In this section, we focus on the existence of pure Nash equilibrium in GSP auction with multiple bidding strategy.

3.1 Preliminaries

Firstly, The following lemma gives some necessary conditions for the existence of Nash equilibrium, which is helpful to verify the (non)existence of Nash equilibrium later.

Lemma 3.1. ***(Necessary conditions)*** *If there exists a pure Nash equilibrium $\widehat{\mathbf{b}}$ in M-GSP, then the following propositions must be true.*

1. *If $v^{(i)} \neq v^{(j)}$ for any $i, j \in \mathcal{N}$, then bidder i gets at least one slot except slot K (the last slot) in $\widehat{\mathbf{b}} \Rightarrow$ bidder i gets exactly M slots in $\widehat{\mathbf{b}}$;*
2. *$\forall i \in \mathcal{N}$, bidder i gets slot $k, k+1, \ldots, k+l$ $(l < m)$ in $\widehat{\mathbf{b}} \Rightarrow b_{k+1} = b_{k+2} = \cdots = b_{k+l+1} + \varepsilon$ for arbitrarily small $\varepsilon > 0$;*
3. ***(Winner monotone)*** *[14] $\forall i, j \in \mathcal{N}$, $v^{(i)} < v^{(j)}$ and bidder i gets at least one slot in $\widehat{\mathbf{b}} \Rightarrow$ bidder j must also gets at least one slot in $\widehat{\mathbf{b}}$;*
4. *If the owner of slot K is bidder i, and $\bar{v} = \max\{v^{(j)} | j \neq i$ and j gets less than M slots in $\widehat{\mathbf{b}}\}$, then $b_K \geq \bar{v}$.*

3.2 Simple Setting

We first consider a simple setting where $K = 3$, $M = 2$. I.e, there are totally 3 slots and each bidder can submit 2 bids. Let the three slots be slot 1, 2, and 3 with CTR $c_1 > c_2 > c_3$. Assume N bidders compete for these three slots and $v^{(1)} > v^{(2)} > \cdots > v^{(N)}$. According to Condition 1 and 3 of Lemma 3.1, in the equilibrium, the winners must be bidder 1 and bidder 2. Either bidder 1 gets slot 1, 2 and bidder 2 gets slot 3 or bidder 2 gets slot 1, 2 and bidder 1 gets slot 3. By Condition 2 and 4 of Lemma 3.1, in equilibrium, bidder 3 bids $b_4 \leq v^{(3)}$, and $b_2 = b_3 \geq v^{(3)}$. Now assume $b_2 = b_3 = x$.

Case I: Bidder 1 gets slot 1, 2 and bidder 2 gets slot 3.

This allocation is an equilibrium if and only if for some fixed $c_1 > c_2 > c_3$, $v^{(1)} > v^{(2)} > v^{(3)}$, all the following inequalities are satisfied.

$$\begin{aligned}(v^{(1)} - x)c_2 &\geq (v^{(1)} - b_4)c_3\\ (v^{(1)} - x)(c_1 + c_2) &\geq (v^{(1)} - b_4)(c_2 + c_3)\\ (v^{(2)} - b_4)c_3 &\geq (v^{(2)} - x)(c_2 + c_3)\\ b_4 &\leq v^{(3)}\\ v^{(2)} \geq x &\geq v^{(3)}\end{aligned} \tag{3.1}$$

Case II: Bidder 2 gets slot 1, 2 and bidder 1 gets slot 3.

Similarly, it is an equilibrium if and only if all the following inequalities are satisfied.

$$\begin{aligned}(v^{(2)} - x)c_2 &\geq (v^{(2)} - b_4)c_3\\ (v^{(2)} - x)(c_1 + c_2) &\geq (v^{(2)} - b_4)(c_2 + c_3)\\ (v^{(1)} - b_4)c_3 &\geq (v^{(1)} - x)(c_2 + c_3)\\ (v^{(1)} - b_4)c_3 &\geq (v^{(1)} - v^{(2)})(c_1 + c_2)\\ b_4 &\leq v^{(3)}\\ v^{(2)} \geq x &\geq v^{(3)}\end{aligned} \tag{3.2}$$

Thus, an equilibrium exists in this setting if and only if one of the above inequality set is satisfied.

Let $A = \{x | x$ is a feasible solution to inequality set (3.1)$\}$, $B = \{x | x$ is a feasible solution to inequality set (3.2)$\}$. We observe that $B \subset A$. Thus the case $K = 3$, $M = 2$ has an equilibrium if and only if the inequality set (3.1) has a solution. By solving the inequality set (3.1), we obtain the following proposition.

Proposition 3.2. *For $K = 3$, 2-GSP has equilibria if and only if one of the following inequality satisfies,*

1. $\frac{v^{(1)}-v^{(2)}}{v^{(1)}-v^{(3)}} \geq \frac{c_3^2}{c_2^2}, c_1c_3 \geq c_2^2$;
2. $\frac{c_1-c_3}{c_1+c_2}v^{(1)} + \frac{c_2+c_3}{c_1+c_2}v^{(3)} \geq \frac{c_2}{c_2+c_3}v^{(2)} + \frac{c_3}{c_2+c_3}v^{(3)}, c_1c_3 \leq c_2^2$,

where $c_1 > c_2 > c_3$ are CTRs of the three slots and $v^{(1)} > v^{(2)} > v^{(3)}$ are the three highest private values among all the bidders.

When some bidders share a same value, it could be verified similarly that the above proposition is still true.

3.3 Existence of Pure Nash Equilibria

When the number of submitted bids of each bidder is unlimited, the following theorem shows that there always exist a pure Nash equilibrium.

Theorem 3.3. *M-GSP always has pure Nash equilibria when* $M \geq K$.

Theorem 3.4. *(Revenue in M-GSP) In M-GSP, the auctioneer's revenue is* $R = v^{(2)} \sum_{j=1}^{K} c_j$ *in any equilibrium when* $M \geq K$.

Obviously, the revenue under this situation is trivially equal to the revenue under VCG mechanism.

3.4 Non-existence of Nash Equilibria

Now, we explore the case where the maximum number of allowed submitted bids of each bidder is less than the number of advertising positions. We first fix $M = 2$ and study the relationship between the number of slots and the (non)existence of Nash equilibrium. The important lemmas obtained are as follows.

Lemma 3.5. ***(The (non)Existence of Nash Equilibrium when*** $M = 2$***)*** *2-GSP always has Nash equilibria for any* $K \leq 2$*; 2-GSP doesn't always have a Nash equilibrium for any* $K \geq 3$.

After studying the case of $M = 2$, we try to generalize our observation to any M and we get some interesting results as follows.

Algorithm 1. Counter Example Generator (K, M)

if $(K \leq M)$ **then**
  exit
end if
if $(K/M == 2)$ **then**
  Let $a = 0$
else
  Let $a = \lfloor K/M \rfloor - 1$
end if
Let the click through rates of K slots be $\mathbf{c} = (c_1, c_2, \cdots, c_K)$
for $i = 1 : aM + M - 2$ **do**
  Let $c_i = 200 + aM + M - 2 - i$
end for
Let $c_{aM+M-1} = 20$, $c_{aM+M} = 11$ and $c_{aM+M+1} = 10$
for $i = aM + M + 2 : K$ **do**
  Let $c_i = 10^{-i}$
end for
Let the true values of $a + 3$ bidders be $\mathbf{v} = (v^{(1)}, v^{(2)}, \cdots, v^{(a+3)})$
for $i = 1 : a$ **do**
  Let $v^{(i)} = 6 + a - i$
end for
Let $v^{(a+1)} = 5, v^{(a+2)} = 4, v^{(a+3)} = 1$
Output $\mathbf{c}$, $\mathbf{v}$

Algorithm 1 is a counter example generator. For any input $K > M$, the algorithm will output K slots with click through rates $c_1 > c_2 > \cdots > c_K$ and $\lfloor K/M \rfloor + 2$ bidders with true values $v^{(1)} > v^{(2)} > \cdots > v^{(\lfloor K/M \rfloor + 2)}$. And in fact,

there doesn't exist a Nash equilibrium for multiple-bidding position auctions with input $\mathbf{c}, \mathbf{v}$ generated by the above algorithm. So we called the algorithm *Counter Example Generator*. The idea of the algorithm comes from the proof of lemma 3.5. In order to prove there may not exist a Nash equilibrium for the case: $K = 4$ in lemma 3.5, we add one more slot with very small click through rate based on the counter example for $K = 3$. And later, to prove the nonexistence of Nash equilibrium for $K > 4$, we add some more slots with very large click through rates. Similarly, here we add $K - 3$ more slots with very large or very small click through rates based on $c_i = 20, c_{i+1} = 11, c_{i+2} = 10$ for some i and add $\lfloor K/M \rfloor - 1$ more bidders with high true values to $v^{(\lfloor K/M \rfloor)} = 5, v^{(\lfloor K/M \rfloor + 1)} = 4, v^{(\lfloor K/M \rfloor + 2)} = 1$. It's easy to verify the correctness of the algorithm. And from the above algorithm, we obtain the following lemma.

Lemma 3.6. *M-GSP doesn't always have a Nash equilibrium when $M < K$.*

From the above two lemmas, we obtain the theorem of (non)existence of Nash equilibrium as follows.

Theorem 3.7. ***(The (non)Existence of Nash Equilibrium)*** *M-GSP always has Nash equilibria when $M \geq K$; It doesn't always have a Nash equilibrium when $M < K$.*

4 Only One Bidder Multi-bidding

In the above sections, we study the M-GSP model in which every bidder can submit at most M bids in the auction. We prove that when the number of slots $K > M$, there doesn't always exist a Nash equilibrium in that model. Now consider the case that only one bidder can submit at most M bids and each of the other bidders can only submit one bid, i.e., only one bidder multi-bidding. Does there always exist a Nash equilibrium in this situation? In this section, we try to find the solution to this question. We use $M^{(i)}$-GSP to represent the extended GSP model in which only bidder i is allowed submitting at most M bids, and each of the other bidders can only submit one bid. In the following part, without loss of generality, we always assume $v^{(1)} \geq v^{(2)} \geq \cdots \geq v^{(N)}$.

Lemma 4.1. ***(The (non)Existence of Nash Equilibrium when $M = 2$)***

1. *$\forall\ i > K$, $2^{(i)}$-GSP always has Nash equilibria;*
2. *$\forall\ i \leq K$, $2^{(i)}$-GSP always has Nash equilibria for any $K \leq 2$; $2^{(i)}$-GSP doesn't always have a Nash equilibrium for any $K \geq 3$.*

Next, we relax the constraint $M = 2$ and study the existence of Nash equilibrium in $M^{(i)}$-GSP model for any given M.

Theorem 4.2. ***(The (non)Existence of Nash Equilibrium in $M^{(i)}$-GSP)***

1. *$\forall\ i > K$, $M^{(i)}$-GSP always has Nash equilibria;*
2. *$\forall\ i \leq K$, $M^{(i)}$-GSP always has Nash equilibria for any $K \leq 2$; $M^{(i)}$-GSP doesn't always have a Nash equilibrium for any $K \geq 3$.*

5 An Impossibility Result

As we mentioned, the sponsored keyword auction is not incentive compatible. However, if we replace the allocation method and pricing method in the position auction by other methods, does there exist a mechanism satisfying not only truthful but also multiple-bidding proof? The following theorem answers the question.

Theorem 5.1. *There exists no mechanism (with allocation method and pricing method) which is truthful, multiple-bidding proof, social efficiency and individual rationality.*

6 Multiple Biddings of Players with Multiple Private Values

Up to this point, we have regarded that each bidder's value per-click is the same for all the submitted bids. It may more adequately called the single value multi-bidding model. In general, there could be a multi-value multi-bidding model. In the multi-value multi-bidding model, a bidder submits more than one bid and his value per-click is not unique. For example, a computer company sells both business laptops and home laptops. The company wants to buy two slots for the keyword 'laptop', one for his business laptops and the other for his home laptops. The value per click for the advertise of the business laptops may be different from that of the home laptops. The company cares about his total profit and in the auction he can always cooperate with himself. And indeed it is often the case as in the examples of the multiple advertisement displays of Dell and HP in Figure 1.

Proposition 6.1. *(Individual Efficiency Property) At any Nash equilibrium, an agent's winning biddings are ordered in their privates values for the corresponding advertisements. That is, the higher is the private value, the higher is the bid.*

In the single-value multi-bidding GSP auction, there may not always exist a Nash equilibrium. Unfortunately, there may not exist a Nash equilibrium in the multi-value multi-bidding GSP auction neither. This can be shown by the following counter example.

	Business Laptop	Home Laptop
Merchant A	10	8
Merchant B	9	6

For the same keyword both Merchant A and B bid for two slots, one for business laptop and the other for home laptop. Their values per-click are shown in the above table. However, there are totally three slots with $c_1 = 12, c_2 = 11, c_3 = 10$. In this setting, there doesn't exist a Nash equilibrium.

References

1. Varian, H.R.: Position auctions. International Journal of Industrial Organization 25(6), 1163–1178 (2007)
2. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. American Economic Review 97(1), 242–259 (2007)
3. Vickrey, W.: Counterspeculation, auctions, and competitive sealed tenders. Journal of Finance XVI(1), 8–37 (1961)
4. Clarke, E.H.: Multipart pricing of public goods. Public Choice 11(1), 11–33 (1971)
5. Groves, T.: Incentives in teams. Econometrica 41(4), 617–631 (1973)
6. Aggarwal, G., Goel, A., Motwani, R.: Truthful auctions for pricing search keywords. In: Proceedings of the 7th ACM Conference on Electronic Commerce (EC 2006), Ann Arbor, Michigan, USA, June 11–15, pp. 1–7 (2006)
7. Zhou, Y., Lukose, R.: Vindictive bidding in keyword auctions. In: Second Workshop on Sponsored Search Auctions, in conjunction with the ACM Conference on Electronic Commerce (EC 2006), Ann Arbor, Michigan, June 11 (2006)
8. Bu, T.M., Deng, X., Qi, Q.: Forward looking nash equilibrium for keyword auction. Information Processing Letters 105(2), 41–46 (2008)
9. Bu, T.M., Deng, X., Qi, Q.: Dynamics of strategic manipulation in ad-words auction. In: Third Workshop on Sponsored Search Auctions, in conjunction with WWW 2007, Banff, Canada, May 8–12 (2007)
10. Cary, M., Das, A., Edelman, B., Giotis, I., Heimerl, K., Karlin, A.R., Mathieu, C., Schwarz, M.: Greedy bidding strategies for keyword auctions. In: Proceedings of the 8th ACM Conference on Electronic Commerce (EC 2007), San Diego, California, USA, June 11–15, pp. 262–271 (2007)
11. Sakurai, Y., Yokoo, M., Matsubara, S.: A limitation of the generalized vickrey auction in electronic commerce: Robustness against false-name bids. In: Proceedings of the 6th National Conference on Artificial Intelligence (AAAI 1999); Proceedings of the 11th Conference on Innovative Applications of Artificial Intelligence, Menlo Park, Cal, July 18–22, pp. 86–92 (1999)
12. Yokoo, M., Sakurai, Y., Matsubara, S.: The effect of false-name declarations in mechanism design: Towards collective decision making on the internet. In: Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000), April 2000, pp. 146–153 (2000)
13. Lambert, N.S., Shoham, Y.: Asymptotically optimal repeated auctions for sponsored search. In: Proceedings of the 9th International Conference on Electronic Commerce: The Wireless World of Electronic Commerce, Minneapolis, USA, August 19–22, pp. 55–64 (2007)
14. Deng, X., Iwama, K., Qi, Q., Sun, A.W., Tasaka, T.: Properties of Symmetric Incentive Compatible Auctions. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 264–273. Springer, Heidelberg (2007)

A CSP-Based Approach for Solving Parity Game*

Min Jiang[1], Changle Zhou[1], Guoqing Wu[2], and Fan Zhang[2]

[1] Department of Cognitive Science, Xiamen University, Fujian, China,
minjiang@xmu.edu.cn
[2] Computer School of Wuhan University, Hubei, China

Abstract. No matter from the theoretical or practical perspective, solving parity game plays a very important role. On one side, this problem possesses some amazing properties of computational complexity, and people are still searching for a polynomial time algorithm. On the other side, solving it and modal mu-calculus are almost the same in nature, so any efficient algorithm concerning this topic can be applied to model checking problem of modal mu-calculus. Considering the importance of modal mu-calculus in the automatic verification field, a series of model checkers will benefit from it. The main purpose of our study is to use constraints satisfaction problem (CSP), a deeply-studied and widely-accepted method, to settle parity game. The significance lies in that we can design efficient model checker through introducing various CSP algorithms, hence open a door to explore this problem of practical importance from a different viewpoint. In the paper, we propose a CSP-based algorithm and the related experimental results are presented.

1 Introduction

Game theory has aroused more interest coming from computer science and AI community in the recent years. The following reasons may explain the growing enthusiasm. At first, after a series of exciting breakthroughs [1,2,3], people has a deeper understanding of the computational complexity on some foundational problems in the field. Secondly, from the practical point of view, game theory plays a major role in handling the matter in the real world. None disbelieves that this world is filled with uncertainty and conflict. Game theory, as probability helps us to investigate the uncertainty, is a powerful mathematical tool to understand the opposite and selfish world. Computational game theory, a brand-new field, is taking shape under the influence of a lot of relevant disciplines.

In computational game theory, there are two kinds of problems attracting special attention: equilibrium, especially Nash equilibrium [4], and winning strategy.

* Research supported by China Postdoctoral Science Foundation funded project under contract No. 20070420749; the National High-Tech Research and Development Plan of China under contract No.2006AA01Z129; the National High-Tech Research and Development Plan of China under contract No. 2007AA01Z185.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 135–146, 2008.

Very roughly speaking, equilibrium depicts, under some conditions, a stable state among $n(n \geq 2)$ players. Winning strategy study can be regarded as the qualitative analysis of some problems we are interested in, in other words, it is to investigate whether a model based on game theory has some desired properties. In this paper, we focus our attention on a special instance of the latter – parity game.

Simply and informally, a parity game is a game played by two players, player 0 and player 1, on a directed graph (V, E). V is finite and composed of two kinds of vertices, i.e., $V = V_0 \bigcup V_1$ and $V_0 \bigcap V_1 = \varnothing$, while a priority function $p : V \rightarrow \mathcal{N}$ assigns a priority to every vertex. Those players move a token from a vertex to a vertex in turns according to the following rules to form a path: if the token is at a V_0-type vertex, player 0 decides which is the next vertex , and if the token is at a V_1-type node, player 1 can make his decision. In the event they can play this game all the time. It is not difficult to find that, some nodes in the path will occur infinitely often. We say player 0 wins the game if the smallest priority of those nodes is even, otherwise player 1 succeed. What we care about is that, assume they play rationally enough, who will win this game.

The motivations of studying parity game come from two fields: pure theory and practical application. At first, solving a parity game is a foundational and interesting problem in theoretical computer science, especially in computational complexity, since it is one of the few problems which are in $\mathbf{NP} \bigcap \mathbf{coNP}$ [7] and even in $\mathbf{UP} \bigcap \mathbf{coUP}$ [11]. Although the community devotes substantial effort to settling it and many people believe that it's in $\mathbf{P}$, finding a polynomial time algorithm is still a major open problem at present. Solving parity problem, just as David S. Johnson stated in his recent paper [6], is " the only candidates I know for (nontrival) membership in $\mathbf{NP} \bigcap \mathbf{coNP}$ - $\mathbf{P}$ ". Secondly, we have already known that if problem X is in $\mathbf{NP} \bigcap \mathbf{coNP}$, then finding the answer of its instance and proving the answer is correct are in $\mathbf{TFNP}$ (total function in $\mathbf{NP}$), so if a problem is in $\mathbf{NP} \bigcap \mathbf{coNP}$ - $\mathbf{P}$, the above question is a candidate for $\mathbf{TFNP} - \mathbf{FP}$.

Moreover, parity game plays an interesting connector in algorithmic game theory. We have known that parity game is polynomial time reducible to simple stochastic game [5], and simple stochastic game can be reduced in polynomial time to finding a Brouwer fixpoint [10] and computing a $\mathbf{P}$-matrix linear complementary problem [8]. The approaches for conquering both latter problems also can be used to compute Nash equilibrium in bimatrix games [2], hence the problem is also in $\mathbf{PPAD}$ (Polynomial Parity Argument, Directed version) [13].

We also want to point out that by using strategy improvement [14], another pretty good approach in practice, one can transform the problems of solving those games into a search problem, hence the value functions used in those algorithms witness membership of search problem in the class of polynomial local search problem ($\mathbf{PLS}$) [9].

If considering this problem from the practical perspective, we will also find its significance. As is well known, model checking [21] has gained tremendous

success in automatic verification field, and modal mu-calculus[1] has received substantial academic interest and enjoyed widely industrial acceptance. The alternation depth of modal mu-calculus formula is a standard measure of conceptual complexity. We can use this measure to induce a hierarchy which is semantically strict, and this notion of syntactic complexity of a formula is reflected in its semantic complexity. With this measure, lots of common formalisms adopted by successful model checkers can be translated into low levels of modal mu-calculus alternation hierarchy, for instance, Propositional dynamic logic (PDL) and Computational Tree Logic (CTL) can be depicted in the first level of this hierarchy, while their extensions ΔPDL and CTL* do not exceed the second level.

Parity game has a close connection with modal mu-calculus, so solving parity game and modal mu-calculus are almost the same in nature. It also tells us that if we can find out a method to solve parity game effectively, even if it lacks strict theoretical fundament, we still can apply it to various types of model checkers. Obviously, it makes our study sense from a practical perspective.

Some parts need explanations. Comparing with [17], we use a concept named game parity progress measure introduced by the literature, which is an extension of progress measure proposed in [18]. But the major differences are that, firstly, their method constructed a strict lattice based on game parity progress measure, and provided a monotonic function. By this way, they transformed solving a parity game into computing a fixpoint on the lattice. Our approach is to handle this problem by using the methods based on constraints satisfaction problem. Both methods use the concept, game parity progress measure from an opposite direction in some sense; secondly, the original paper uses induction method to prove some important theorems, but we give a constructional interpretation on it to accelerate the solving speed.

The rest of the paper is organized as follows: Section 2 introduces some previous works; in section 3 we describe some concepts about parity game and constraints satisfaction problem (**CSP**); in section 4, we discuss how to translate the problem of solving parity game into a solution of CSP; then one CSP-based algorithm and experimental results are presented in Section 5 and Section 6; and in section 7, we conclude this paper and future work is discussed.

2 Previous Works

As we claimed, solving parity game shows its significance in different fields; so many different methods have been presented to settle this amazing but still open problem. Here, we just list three kinds of algorithms with their great influence on the community: deterministic, randomized and strategy improvement algorithms.

Let $n = |V|$ and $m = |E|$ be the number of vertices and edges of parity game graph and let d be the largest priority assigned to vertices by a function $p : V \rightarrow \{0, \ldots, d\text{-}1\}$.

[1] Roughly speaking, modal mu-calculus is regard as an extension of standard modal logic, which adds some monadic variables bounded by fixpoint of definable operators.

In [17], the authors give a deterministic algorithm based on progress measure. Progress measure [18] are decorations of graphs. We can obtain some global, often infinitely desired properties by guaranteeing local consistency of graph. When $d = O(\sqrt{n})$, the time complexity of this algorithm is $O(dm \cdot (2n/d)^{d/2})$. In a very recent paper [16], the same author presented another deterministic algorithm for solving this problem. If the out-degree of all vertices is bounded, the complexity of the algorithm is $n^{O(\sqrt{n/\log n})}$; if we drop this constraint, the complexity is $n^{O(\sqrt{n})}$, and at the same time, this new algorithm just needs polynomial space.

Another randomized algorithm has been given by Björklund et.al [15], which also has an expected running time bound of $n^{O(\sqrt{n/\log n})}$. Compared with the above deterministic algorithm, the advantage of this randomized algorithm is that it can be used to settle some other games, for example, computing mean payoff games, discounted games and simple stochastic games. But the deterministic algorithm does not seem to be good for other games in an obvious way.

Strategy improvement, though lacking a rigorous theoretical analysis, is an effective method in practice. Hoffman and Karp first proposed it for solving stochastic game in 1966 [14], then Puri [19] and vöge[20] et.al gave two algorithms based on this concept to conquer parity game. Both algorithms can do one iteration in polynomial time, but it is a pity that the number of iterations may be huge. The major problem of Puri' approach is that their algorithm is not a discrete algorithm, but involves linear programming and high precision arithmetic. Vöge's algorithm avoids those weaknesses.

3 Preliminaries

3.1 Parity Game

A parity game is composed of an arena, a priority function and a starting node.

An arena is a triple $\mathcal{A} = (V_0, V_1, E)$. V_0 a is a set of 0-type vertices and V_1 denotes a set of 1-type vertices, and both sets are finite. We define $V_0 \bigcup V_1 = V$ and $V_0 \bigcap V_1 = \varnothing$. $E \subseteq V \times V$, sometimes, it also means the set of moves. The set of successors of $v \in V$ is defined by $vE = \{v' \in V | (v, v') \in E\}$. We define $[d] = \{0, 1, \ldots, d-1\}$ and $d \in \mathcal{N}$. A priority function $p : V \to [d]$, which assigns a priority to every vertex . Starting node $v_0 \in V$. So a parity game is a tuple $\mathcal{G} = (\mathcal{A}, p, v_0)$. We assume that $\mathcal{G}$ is total, in other words, every vertex on the graph has at least one out-going edge, i.e., for every $v \in V$, there is a $v' \in V$ with $(v, v') \in E$.

A play of a parity game $\mathcal{G}$ is a path $\pi = v_0 v_1, \ldots$, which initiates in v_0. We can imagine it is formed as the following way: at the beginning of the game, a token is placed on v_0. If v_0 is a V_0-type vertex, player 0 chooses a vertex $v' \in vE$ as successor. If v_0 is a V_1-type vertex player 1 makes his decision, and the construction of the play continues with v'. According to our definition of parity game, obviously, all the paths are infinite since every vertex has at least one successor to choose.

Given a path π, we define a function $Inf : \pi \rightarrow 2^V$, which denotes a set of vertices which occur infinitely often[2]. Let another function $Min : 2^V \rightarrow [d]$, which finds the smallest priority among a set of vertices. So $Min(Inf(\pi))$ means the smallest priority of those vertices occurring infinitely often in the play π. if $Min(Inf(\pi))$ is even, we say player 0 wins this play, otherwise, π is a winning play for player 1.

A *strategy* for player σ, $\sigma \in \{0, 1\}$, is a function $\theta : V^*V_\sigma \rightarrow V$, which tells the player σ how to make his decision according to the current situation in a play. we call a strategy *memoryless* (or *positional*) strategy if $\theta : V_\sigma \rightarrow V$ such that $\theta(v) \in vE$ for all $v \in V_\sigma$. It means a memoryless strategy chooses the next vertex just according to the last nodes visited. A play $\pi = v_0v_1 \ldots$ is called *conforming* a player σ's strategy θ, if for all $i \in \mathcal{N}$, $v_i \in V_\sigma$, $v_{i+1} = \theta(v_i)$. We say a strategy is a *winning strategy* for player σ, i.e., if every play conforms to θ, player σ will wins this game,

Given a parity game $\mathcal{G}$, we define the problem of solving the parity game is to determine the winner of it. The following theorem guarantees every parity game has a winner.

Theorem 1 (Memoryless Determinacy [26,27]). *For every parity game, there is a unique partition (W_0, W_1) of the set of vertices of its game graph, such that there is a memoryless winning strategy for player 0 from W_0, and a memoryless winning strategy for player 1 from W_1.*

Remark 1. At first, the theorem is not trivial in an infinite game, and it plays a fundamental role in solving parity game. According to our definition, it tells us that either player 0 has a memoryless winning strategy in a parity game or player 1 has it. Secondly, those literatures define solving parity game as finding the partition (W_0, W_1). Obviously, our problem is polynomial time equivalent to their problem since we can carry them out in polynomial time to get the partition. In the rest of the paper, strategy means memoryless strategy.

3.2 Constraint Satisfaction Problem (CSP)

Many computational problems from different application fields can be considered as constraint satisfaction problems. For example, the problems of scheduling a collection of tasks [22], laying out a chip [23], even understanding the visual image [24], can all be done in this way. ACM(Association for Computing Machinery) identified it as one of the strategic directions in computer research. Many facts tell us CSP is a powerful tool to settle some hard problems, especially, combinational optimization problem.

Let us recall some primary concepts about CSP briefly. A CSP is a 3-tuple $CSP =< X, D, C >$, where $X = \{x_1, ..., x_n\}$ is a set of variables, and D_i is a non-empty domain of x_i; $C = \{c_1, \ldots, c_m\}$, and every c_i involves some subsets of the variables and specifies the allowable combinations of the values for those subsets.

[2] we just have finite vertices, and the path is infinite, so there are some vertices occurring infinitely often.

CSP is a problem which was deeply studied, and there are a huge amount of literature and algorithms. Although it is hard to solve theoretically, i.e., the famous **3-SAT** problem is a special case, sometimes we still can find a solution effectively in practice. Another predominance of CSP is that the simple yet powerful description ability makes one possess the capability to handle sophisticated problem even if he is not very familiar with it.

4 Translate Parity Game to CSP

4.1 Character of Winning Strategy

For depicting the character of winning strategy, we need the following definitions and theorems.

Definition 1 (Annotation). *We call* $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{d-1})$ *an* annotation, *which is a d-tuple of non-negative integers. We can compare an annotation to another one according to lexicographic ordering. If we subscript with a number* $i \in \mathcal{N}$ *under the comparison symbols* $\leqq, \geqq, <, \ldots$, *it means the lexicographic ordering applied to their first i components, for example,* $(1,2,3) >_0 (0,2,3)$, $(4,3,4) =_1 (4,3,3)$.

Definition 2 (Parity Progress Measure [17]). *Let* $\mathcal{G}$ *be a parity game, and d is the max priority among all the vertices. We call a function* $\rho : V \to \mathcal{N}^d$ *parity progress measure if the following conditions are held. For every* $(v,w) \in E$, $\rho(v) \geqq_{p(v)} \rho(w)$ *if* $p(v)$ *is even ,otherwise,* $\rho(v) >_{p(v)} \rho(w)$.

Remark 2. We can understand a parity progress measure as the following way: it assigns a value to every vertex; if the priority of a vertex is even, then its value will not be less than the values of its neighbors, and if the priority is odd, the value will larger than its neighbors'.

Definition 3. *We say a cycle in a parity graph is even, if the smallest priority of the vertex occurring in this cycle is even, otherwise, the cycle is odd.*

The following lemmas reveal some special relationships between parity progress measure and winning strategy.

Lemma 1 ([17]). *If there is a parity progress measure for a parity graph* $\mathcal{G}$ *then all cycles in* $\mathcal{G}$ *are even.*

Proof. Reduction to absurdity. We assume that there is a odd cycle $\gamma = v_1 v_2, \ldots, v_k v_1$. Without loss of generalization, we let v_1 has the smallest priority and it is odd. Then according to the definition, we have $\rho(v_1) >_{p(v1)} \rho(v2)$, $\rho(v_2) \geqq_{p(v_2)} \rho(v_3)$, $\ldots$,$\rho(v_k) \geqq_{p(v_k)} \rho(v_1)$; due to the property of lexicographic ordering, we have $\rho(v_1) >_{p(v_1)} \rho(v_1)$, a contradiction. □

.

The lemma tells us something but not enough. First, the range of the parity progress measure is too wide, so it's hard to be computed. Second, it's just a witness for player 0 to win all the vertices, obviously, it is unusual in parity games. Hence we need some other tools based on the lemma to conquer those problems.

We settle the first problem, i.e., we will restrict the range of parity progress measure.

Definition 4. *Given a parity game $\mathcal{G}$, let $\eta(v)$ be the set of all the paths whose starting node is v, and let $\#_i(\eta(v))$ be the maximal number of nodes with* priority *i occurring on any element in the set $\eta(v)$, at the same time, the element does not contain a vertex whose priority is smaller than i. If we cannot find such an element or the priority of v is smaller than i, $\#_i(\eta(v)) = 0$.*

If infinitely many vertices with priority i occur on some path with no smaller priority occurring on the path, the value is infinite.

Lemma 2. *Given a parity game $\mathcal{G}$, if all cycles in $\mathcal{G}$ are even, for any odd i, $\#_i(\eta(v))$ will not larger than the number of vertices with priority i.*

Proof. Firstly, some $\#_i(\eta(v))$ is finite for some odd i. Otherwise there is an infinite path originating in v and the path contains no vertex with lower priority than i. Obviously, a vertex with priority i occurs twice in the path, so there is a odd cycle, a contradiction to the assumption of the lemma. Secondly, assume that $\#_i(\eta(v))$ is larger than the number of vertices with priority i for some odd i, so there is a path starting in v such that $\#_i(\eta(v))$ vertices with priority i before a vertex with priority small than i, and the vertex occurs twice according to our assumption. So there is an odd cycles, a contradiction again. □

Definition 5. *Given a parity game $\mathcal{G}$ with the maximal priority d, we define $M_\mathcal{G}$ is an* annotation*, and $M_\mathcal{G}$ is constructed as follows: if d is even, $M_\mathcal{G} = [1] \times [\#_1(\eta(\cdot)) + 1] \times [1] \times [\#_3(\eta(\cdot)) + 1] \times \ldots \times [1] \times [\#_{d-1}(\eta(\cdot)) + 1]$, otherwise, $M_\mathcal{G} = [1] \times [\#_1(\eta(\cdot)) + 1] \times [1] \times [\#_3(\eta(\cdot)) + 1] \times \ldots \times [\#_{d-2}(\eta(\cdot)) + 1] \times [1]$.* ([d] = {0, 1, ..., d-1}).

Remark 3. We consider $M_\mathcal{G}$ as an annotation. In its odd position i the value is not larger than the number of vertices with priority i, and in even position it is zero. Please note that, in an intuitive way, our definition is similar to the definition given by [17], but there is a notable difference when we hope to take CSP as a tool to settle this problem. If adopting that definition, we will encounter a knotty problem - the domains of variables have huge possible values and no clue helps us to reduce it, but our definition avoid the weakness. We will demonstrate the advantages of our definition when translating parity game to an instance of CSP, especially, when we implement a unary constraint.

Lemma 3 (Small Parity Progress Measure [17]). *If all cycles in a parity graph $\mathcal{G}$ are even then there is a parity progress measure $\varrho : V \to M_\mathcal{G}$.*

Remark 4. What we want to explain is that, according to Definition. 5, for every $v \in V$, if $p(v)$ is even, $\varrho(v) = (0, \#_1(\eta(v)), 0, \#_3(\eta(v)), \ldots, 0, \#_{d-1}(\eta(v))$; if $p(v)$ is odd, $\varrho(v) = (0, \#_1(\eta(v)), 0, \#_3(\eta(v)), \ldots, \#_{d-2}(\eta(v), 0)$.

Now it's time to settle the second problem, namely, both players have a chance to win the game. To achieve this goal, we just need drop the constraint that all cycles are even. The following lemmas and theorems given by [17] assure it, and we don't modify them anymore, so we represent them here directly.

Definition 6. *Let $M_G^\top$ denote $M_G \bigcup \{\top\}$ where $\top$ is an extra element and larger than any elements of $M_\mathcal{G}$ under the order $>_i$, for all $i \in [d]$; and $\top =_i \top$. $Prog(\varrho, v, w)$ denotes the least $m \in M_G^\top$, such that $m \geq_{p(v)} \varrho(w)$, the inequality is strict if $p(v)$ is odd.*

Definition 7 (Game Parity Progress Measure). *A function $\varrho : V \to M_G^\top$ is a game parity progress measure if for all $v \in V$, we have:*

1. *if $v \in V_0$ then $\varrho(v) \geqq_{p(v)} Prog(\varrho, v, w)$ for* some *$w \in vE$, and*
2. *if $v \in V_1$ then $\varrho(v) \geqq_{p(v)} Prog(\varrho, v, w)$ for* all *$w \in vE$.*

Theorem 2 ([17]). *If ϱ is a game parity progress measure then we can determine who wins the game.*

Now, we know that a game parity progress measure is a witness of winning strategy, so we can solve a parity game by finding a game parity progress measure. It's one of theoretical foundations of our approach.

4.2 Encoding Parity Game to CSP

In this section, we will discuss how to transform solving a given parity game into finding a solution of CSP. Note that, the players in all the parity games graph we discuss in the paper are finite and every player has at least one successor. Given a parity game $\mathcal{G}$, we present our reduction as follows:

1. Every variable v_i denotes a vertex i on the parity game graph $\mathcal{G}$;
2. The domain of every variable is a set of all possible values, i.e., all possible values of $M_G^\top$. We arrange the elements of the domain ascending;
3. **Unary constraint:** The constraint is important for our approach since it can reduce the search space remarkably. The constraint stems from *definition* 4. For example, if the current vertex is v, and $p(v) = i$; it will remove all the possible assignments: for all $j > i$, its component $\#_j(v)$ is not equal to zero;
4. **Binary constraints:** The binary constraints proposed by our approach are composed of four types:
   (a) $V_0 - Odd$**:** If the current vertex $v \in V_0$, and $p(v)$ is odd, we will apply this constraint. This constraint guarantees the value of this vertex is strictly larger than the smallest value of some neighbors.
   (b) $V_0 - Even$**:** If the current vertex $v \in V_0$, and $p(v)$ even, we will apply this constraint. This constraint guarantees the value of the vertex is not less than the smallest value of some neighbors.

(c) $V_1 - Odd$**:** If the current vertex $v \in V_1$, and $p(v)$ odd, we will apply this constraint. This constraint guarantees the value of the vertex is strictly larger than the smallest value of all its neighbors.

(d) $V_1 - Even$**:** If the current vertex $v \in V_1$, and $p(v)$ even, we will apply this constraint. This constraint guarantees the value of this vertex is not less than the smallest value of all its neighbors.

Obviously, if a vertex has been assigned to $\top$, the corresponding constraint must be satisfied.

5. **Constraints propagation:** Assign a value to each variable in turn, with the requirement that the current assignment should not conflict with the previous assignments.

5 An CSP-Based Algorithm

5.1 The Algorithm

We present our algorithm as follows.

Algorithm 1. Parity-CSP (sketch)

```
Data: A parity game G = (A, p, v_1)
Result: "Player 0 wins" or "Player 1 wins"
begin
    Step.1 Select the node v_1 as root, and do breadth-first search, form a
    sequence v_1, v_2, ..., v_n, and add them to a queue CheckQueue;
    Step.2 for i = 1 to n do
        Unary-Constraint(v_i);
    Step.3 while CheckQueue ≠ ∅ do
        CV ← CheckQueue.head;
        if CV ∈ V_0 then
            if p(CV) is odd then Change := V_0 − Odd(CV);
            else Change := V_0 − Even(CV);
        if CV ∈ V_1 then
            if p(CV) is odd then Change := V_1 − Odd(CV);
            else Change := V_1 − Even(CV);
        if Change == True then
            forall (v, CV) ∈ E do
                CheckQueue.add(v)
    Step.4 if v_1.value ≠ ⊤ then return "player 0 wins"else return "player 1
    wins"
end
```

Remark 5. By using this reduction, we will benefit from taking various heuristic searching methods to accelerate the performance of the algorithm, but for the limit of space, we will not discuss it.

5.2 Correctness of the Algorithm

Theorem 3. *The algorithm is terminable, and if we have a solution of an instance, then we know whether player 0 wins the corresponding parity game or not.*

Proof. If the value of CV is $\top$, no new element will be added to the queue. Obviously, the algorithm will be terminated at some time. If we have an solution of an instance, the value of v_1 is $\top$ or not. If it is $\top$, we know that there is an odd j such that $\#_j(v_1) = \infty$, and it means that there is a path which must contain some vertices with priority j twice, then there is an odd cycle, hence player 1 can win the game according to the path. Otherwise, player 0 wins. □

6 Experimental Results

Our prototype employs JCL[3] package, which is developed by EPFL. For possessing the unary and the bi-constraints needed in our algorithm, we extend it. Please note, it is not for its performance that we choose JCL. Strictly speaking, it is not the fastest one, but it is the only package that we find can handle the soft constraint satisfaction problem. Despite not using this feature in the paper,

Table 1. The running time

# of V	$MAX\ of\ Pri. = 1$		$MAX\ of\ Pri. = \lceil\sqrt{\#}\rceil$	
	CSP	Solving	CSP	Solving
50	0.03s	0.03s	0.28s	0.18s
100	0.05	0.07s	0.34s	1.4s
200	0.9s	1.2s	1.8s	23.1s
300	0.18s	0.25s	2.8s	5m23s
400	0.25s	0.27s	5.6s	34s
500	0.36s	0.43s	6.7s	1m23s
600	0.58s	0.54s	8.9s	3m14s
700	0.77s	0.71s	15.1s	23m13s
800	0.86s	0.84s	17.8s	7m14s
900	1.23s	0.94s	27.1s	16m23s
1000	1.4s	1.23s	34.1s	2h34s
1200	1.56s	1.44s	56s	1h45s
1400	1.73s	1.56s	1m14s	3h34s

[3] http://liawww.epfl.ch/JCL/index.htm

it has established the foundation for our future work. In the present paper, all programs are implemented in java 5.0, and the running environment is Linux FC 5.0 on Pentium(R) 1.86G with 1G memory. The running time is presented in Table 1. We must point out that it is a premature implement, so the program still leave some room to improve its performance.

We generate those random graphs with different vertices, and let the out-degree of every vertex be larger than one. Since every parity game can be reduced to a corresponding parity game with constant out-degree two, we restrict the out-degree to be less than three in our experiment, and assign a priority to a vertex randomly. The column CSP depicts the time spent constructing an CSP instance, and the other column means the time spent solving it.

7 Conclusion and Future Work

Parity game plays an interesting connector between theoretic computer science and industrial application. Any effective approach for solving this problem can be used to settle model checking problem of modal mu-calculus, even other common formalisms. So it seems more important to find it at a fast speed from an empirical perspective.

In this paper, we put forward an algorithm based on constraints satisfaction problem, which takes a concept called game parity progress measure as foundation. In order to make this concept more fit CSP approach, we do some modification and give constructional interpretations to it. By this way, the searching space is reduced and computing speed of our implement is accelerated. We hope this approach will open a door to explore this hard problem from a different viewpoint.

There are two major tasks needed to be done in our future work.

1. speed up the algorithm by designing some heuristic searching strategies and make it more memory-efficient.
2. take this approach as a tool to settle some other well-accepted formalisms and games, especially, modal temporal logic and mean-payoff parity game.

References

1. Chen, X., Deng, X.: 3-NASH is PPAD-Complete. ECCC (2005) TR05-134
2. Chen, X., Deng, X.: Settling the Complexity of 2-player Nash Equilibrium. ECCC (2005) TR05-140
3. Daskalakis, C., Papadimitriou, C. H.: Three-Players Games are Hard. ECCC (2005) TR05-139
4. Nash, J.F.: Non-cooperative games. Annals of Mathematics (54), 286–295 (1951)
5. Condon, A.: The complexity of stochastic games. Information and Computation, 203–224 (1992)
6. David, S.J.: The NP-completeness column: Finding needles in haystacks. ACM Transactions on Algorithms 3(2), 24 (2007)

7. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On Model-Checking for Fragments of mu-Calculus. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 385–396. Springer, Heidelberg (1993)
8. Gartner, B., Rust, L.: Simple Stochastic Games and P-Matrix Generalized Linear Complementarity Problems. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 209–220. Springer, Heidelberg (2005)
9. Johnson, D.S., Papadimtriou, C.H., Yannakakis, M.: How easy is local search? Journal of Computer and System Sciences 37, 79–100 (1988)
10. Juba, B.: On the hardness of simple stochastic games (manuscript, 2004)
11. Jurdzinski, M.: Deciding the winner in parity games is in UP and co-UP. Information Processing Letters 68, 119–124 (1998)
12. Megiddo, N., Papadimitriou, C.H.: A note on total functions, existence theorems, and computational complexity. Theoretical Computer Science 81, 317–324 (1991)
13. Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. Journal of Computer and System Sciences 48, 498–532 (1994)
14. Hoffman, A.J., Karp, R.M.: On Nonterminating Stochastic Games. Management Science 12, 359–370 (1966)
15. Bjorklund, H., Sandberg, S., Vorobyov, S.: A discrete subexponential algorithm for parity games. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 663–674. Springer, Heidelberg (2003)
16. Jurdzinski, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: SODA 2006, Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 117–123 (2006)
17. Jurdzinski, M., Ics, B.R.: Small Progress Measures for Solving Parity Games. In: STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science Lille, France, February 17-19 (2000)
18. Klarlund, N., Kozen, D.: Rabin measures and their applications to fairness and automatatheory. In: LICS 1991, Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science, pp. 256–265 (1991)
19. Puri, A.: Theory of hybrid systems and discrete event systems. Ph.D thesis, University of California, Berkeley (1996)
20. Voge, J., Jurdziiiski, M.: A Discrete Strategy Improvement Algorithm for Solving Parity Games. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 202–215. Springer, Heidelberg (2000)
21. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
22. Dhar, V., Ranganathan, N.: Integer Programming vs Expert Systems: An Experimental Comparison. Communications of the ACM, 323–336 (1990)
23. de Kleer, J., Sussman, G.J.: Propagation of Constraints Applied to Circuit Synthesis. Circuit Theory and Applications, 127–144 (1980)
24. Davis, A.L., Rosenfeld, A.: Cooperating Processes for Low Level Vision: A Survey. Articial Intelligence, 245–263 (1981)
25. Russell, S., Norvig, P.: Artificial Intelligence: a Modern Approach. Prentice Hall, Englewood Cliffs (2000)
26. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: FOCS 1991, Proceedings of 32nd Annual Symposium on Foundations of Computer Science, pp. 368–377 (1991)
27. Mostowski, A.W.: Games with forbidden positions. Technical Report 78, University of Gdansk (1991)

Characterizing and Computing Minimal Cograph Completions*

Daniel Lokshtanov, Federico Mancini, and Charis Papadopoulos

Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{daniello,federico,charis}@ii.uib.no

Abstract. A cograph completion of an arbitrary graph G is a cograph supergraph of G on the same vertex set. Such a completion is called minimal if the set of edges added to G is inclusion minimal. In this paper we present two results on minimal cograph completions. The first is a a characterization that allows us to check in linear time whether a given cograph completion is minimal. The second result is a vertex incremental algorithm to compute a minimal cograph completion H of an arbitrary input graph G in $O(V(H) + E(H))$ time.

1 Introduction

Any graph can be embedded into a cograph by adding edges to the original graph and the resulting graph is called a *cograph completion*, whereas the added edges are called *fill edges*. A cograph completion with the minimum number of edges is called *minimum*, while it is called *minimal* if no proper subset of the fill edges produces a cograph when added to the original graph.

Computing a minimum completion of an arbitrary graph into a specific graph class is an important and well studied problem with applications in molecular biology, numerical algebra, and more generally areas involving graph modelling with missing edges due to lacking data [10,23,26]. Unfortunately minimum completions into most interesting graph classes, *including cographs*, are NP-hard to compute [5,8,17,23,33]. This fact encouraged researchers to focus on various alternatives that are computationally more efficient, at the cost of optimality or generality. Examples of the approaches that have been attempted include approximation [18], restricted input [28,29,30,31], parameterization [19,20,21,22,32] and minimal completions [12,13,14,16,25,27]. Here we consider the last alternative.

The reason why minimal completions can be used as a tool to understand minimum completions better, is that every minimum completion must also be a minimal one. Hence, if one is able to efficiently sample from the space of minimal completions, it is possible to pick the one in the sample with fewest fill edges and have good chances to produce a completion close to the minimum. This process, while only being a heuristic without any approximation guarantees, has proven to often be good enough for practical purposes [2,34]. In addition, the study

* This work is supported by the Research Council of Norway through grant 166429/V30.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 147–158, 2008.

of minimal completions gives a deep insight in the structure of the graph class we consider. It is often the case that new tools created to characterize minimal completions are applied to design new exact algorithms for minimum completions [9,35,36], or to efficiently solve other problems on the specific graph class in question. In particular, from a new minimal completion algorithm there can easily follow new recognition algorithms [3,37], since completion can be regarded as a generalization of recognition. Finally, as shown in [3] for the case of chordal graphs, completions can also be useful to efficiently solve problems that otherwise are hard on the original input graph.

In this paper we consider minimal cograph completions, and we study them both from a graph theoretic and from an algorithmic point of view. Our main graph theoretic result is a theorem that captures the essence of what makes a cograph completion minimal. We apply this characterization to obtain several algorithmic results. First we give a linear time algorithm for the *characterization problem*, namely checking whether a given cograph completion is minimal. Second we show how this algorithm can be applied to solve the *extraction problem*, i.e., extracting a minimal completion from a non-minimal one by removing fill edges. Finally we present our main algorithmic result, an algorithm for the *computation problem*. In this problem you are asked to compute a minimal cograph completion of an arbitrary input graph. Our algorithm can be viewed as a generalization of the cograph recognition algorithm given in [7], due to its incremental nature. We consider the input graph one vertex at the time, completing the graph locally in an on-line fashion. Due to this feature it is likely the algorithm can be extended to a dynamic completion algorithm, like the one for split graphs presented in [37]. The running time is linear in the size of the computed minimal cograph completion, and therefore optimal if an explicit representation of the output graph is required.

One should notice that, for cographs, as for other classes for which completions are interesting, an algorithm for the extraction problem can easily be applied to solve the computation problem as well. The reason why we provide a separate algorithm for each problem, is the big difference between their time complexity. While our computation algorithm is linear in output size, the one for the extraction problem runs in time $O(|V(G)|^4)$ in the worst case.

While we have argued why minimal completions are important in general, we have not yet explained why it is interesting to study minimal *cograph* completions. An obvious reason is that cographs arise naturally in many different fields. It is not by chance that they have been re-discovered various times and have so many different characterizations [6]. Even more interesting is the fact that many problems that are NP-hard on general graphs, can be solved very efficiently when the input is restricted to being a cograph (see [4] for a summary of such results).

However, as noticed by Corneil et. al [7], in most typical applications, the graphs encountered may not be cographs but in fact will be very close to being a cograph. Due to this they asked for good heuristics for the problem of adding and deleting as few edges of the input graph as possible to achieve a cograph.

Our computation alorithm can be used as such a heuristic, both in the case of adding and in the case of deleting edges. The reason for this is that the class cographs is self-complementary. Besides, an advantage of using a minimal completion algorithm as a heuristic is that the minimality guarantees that we never add unnecessary fill edges. Also, since our completion algorithm is fast it is possible to improve the performance of the heuristic by trying several different completions and picking the one with fewest edges.

Another reason to study cographs with respect to minimal completions, is that this graph class is not *sandwich monotone* (see [14] for an exact definition). If a graph class has this property, then a completion into the class is minimal if and only if no *single* fill edge can be removed keeping the completed graph in the class. Hence, for polynomial time recognizable classes with this property, it becomes trivial to solve the characterization problem, and very easy to solve both the extraction and the computation problems as well. Examples of algorithms that exploit sandwich monotonicity for efficiently extracting and computing a minimal completion, are those for chordal [2], split [12], threshold and chain graph [14] completions. In contrast, among the classes that do not have the sandwich monotone property, the only one for which a solution to the characterization and extraction probems is known, is the class of interval graphs [15].

When viewed from this perspective, our characterization of minimal cograph completions becomes interesting. It allows us to check minimality efficiently and provides a straightforward way to solve the extraction problem for cograph completions, even though cographs do not have the sandwich monotone property.

Before we begin the technical exposition, we should note that it is possible to adapt the algorithm for the cograph sandwich problem given in [11] to yield a polynomial time algorithm for the extraction problem. However such an algorithm would only be a smart brute force approach and would not give any graph theoretical characterization or intuition on how a minimal cograph completion should look like, which is what we aim for. Also the running time of the algorithm we would get from such an approach would be too high for any practical purpose.

The paper is organized in three main sections: Section 2 with background and definitions, Section 3 with the details of the characterization and finally, Section 4 with the computation algorithm. This section is split in two. The first part contains a high level description of the algorithm for easing the understanding and proving correctness. The second part contains a version more suitable for implementation, together with a running time analysis of the algorithm. The proofs have been omitted due to lack of space.

2 Preliminaries

We consider undirected finite graphs with no loops or multiple edges. For a graph $G = (V, E)$, $V(G) = V$ and $E(G) = E$. For $S \subseteq V$, the *subgraph of G induced by S* is denoted by $G[S]$. Moreover, we denote by $G - S$ the graph $G[V \setminus S]$ and by $G - v$ the graph $G[V \setminus \{v\}]$. We distinguish between *subgraphs*

and *induced subgraphs*. By a *subgraph* of G we mean a graph G' on the same vertex set containing a subset of the edges of G, and we denote it by $G' \subseteq G$. If G' contains a proper subset of the edges of G, we write $G' \subset G$.

The *neighborhood* of a vertex x of G is $N_G(x) = \{v \mid xv \in E\}$. The *degree* of x in G is $d_G(x)$. For $S \subseteq V$ $N_G(S) = \bigcup_{x \in S} N_G(x) \setminus S$. The *complement* $\overline{G}$ of a graph G consists of all vertices and all non-edges of G. A vertex x of G is *universal* if $N_G(x) = V \setminus \{x\}$ and is *isolated* if it has no neighbors in G. A *clique* is a set of pairwise adjacent vertices while an *independent set* is a set of pairwise non-adjacent vertices. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $V_1 \cap V_2 = \emptyset$, their *union* is $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. Their *join* $G_1 + G_2$ is the graph obtained from $G_1 \cup G_2$ by adding all the edges between the vertices of V_1 and V_2.

A *connected component* of a disconnected graph G is a connected subgraph of G with a maximal set of vertices and edges. The *co-connected components* of G are the connected components of $\overline{G}$. By $\mathcal{C}(G)$ and $\widehat{\mathcal{C}}(G)$ we denote the family of the vertex sets of the connected components and co-connected components, respectively, of G. More formally, $\mathcal{C}(G) = \{C_i \mid G[C_i]$ is a connected component of $G\}$ and $\widehat{\mathcal{C}}(G) = \{\widehat{C}_i \mid G[\widehat{C}_i]$ is a co-connected component of $G\}$.

Given an arbitrary graph $G = (V, E)$ and a graph class Π, a Π *completion* of G is a graph $H = (V, E \cup F)$ such that $H \in \Pi$, and H is a *minimal* Π completion of G if $(V, E \cup F')$ fails to be in Π for every $F' \subset F$. The edges added to the original graph in order to obtain a Π completion are called *fill edges*.

Cographs: The class of cographs, also known as *complement reducible graphs*, is defined recursively as follows: (i) a single vertex is a cograph, (ii) if G_1 and G_2 are cographs, then $G_1 \cup G_2$ is also a cograph, (iii) if G_1 and G_2 are cographs, then $G_1 + G_2$ is also a cograph. Here we shall use the following characterization of cographs.

Theorem 1 ([6]). *G is a cograph if and only if the complement of any nontrivial connected induced subgraph of G is disconnected.*

Along with other properties, it is known that cographs admit a unique tree representation, called a *cotree* [6]. For a cograph G its cotree, denoted by $T(G)$, is a rooted tree having $O(|V|)$ nodes. Notice that we also consider $T()$ as a function that, given a cograph as argument, returns the corresponding cotree. Similarly, we define the function $Co()$, that takes as an input a cotree and returns the corresponding cograph; that is, for a cograph G, $Co(T(G)) = G$. The vertices of G are precisely the leaves of $T(G)$ and every internal node of $T(G)$ is labelled by either 0 (0-node) or 1 (1-node). Two vertices are adjacent in G if and only if their least common ancestor in $T(G)$ is a 1-node. Moreover, if G has at least two vertices then each internal node of the tree has at least two children and any path from the root to any node of the tree consists of alternating 0- and 1-nodes. The complement of any cograph G is a cograph and the cotree of the complement of G is obtained from $T(G)$ with inverted labeling on the internal

nodes of $T(G)$. Cographs can be recognized and their cotrees can be computed in linear time [7].

For a node t of $T(G)$ we denote by T_t the subtree rooted at t. The set of t's children in $T(G)$ is denoted by $Q(t)$ and the set of leaves of T_t is denoted by $M(t)$. If $S \subseteq V(T(G))$ then $M(S) = \bigcup_{t \in S} M(t)$. Let $Q(t) = \{t_1, \ldots, t_q\}$. If t is a 0-node then $G[M(t)]$ is disconnected with q connected components and $M(t_i) = C_i$, for $C_i \in \mathcal{C}(G[M(t)])$. Otherwise, if t is a 1-node then $G[M(t)]$ is connected with q co-connected components and $M(t_i) = \widehat{C}_i$, for $\widehat{C}_i \in \widehat{\mathcal{C}}(G[M(t)])$.

Observation 2. *Let $G = (V, E)$ be a cograph, $T(G)$ be its cotree, and let $a(G)$ be the set of the 1-nodes of $T(G)$.* $\sum_{t \in a(G)} \sum_{t_i, t_j \in Q(t)} |M(t_i)| \cdot |M(t_j)| = |E|$.

3 Characterizing Minimal Cograph Completions

Here we exploit certain properties of cographs in order to characterize minimal cograph completions.

Lemma 3. *Let $H = (V, E \cup F)$ be a cograph completion of a graph $G = (V, E)$. H is a minimal cograph completion of G if and only if $H[C_i]$ is a minimal cograph completion of $G[C_i]$ for every $C_i \in \mathcal{C}(H)$.*

Let $H = (V, E \cup F)$ be a cograph completion of a graph $G = (V, E)$. Next we focus on a connected cograph completion H of G. Note that H has at least two co-connected components since it is connected. In order to characterize minimality of H, the idea is to consider any two co-connected components of H, remove all the fill edges between them in H, and then simply check the connectivity of the resulting graph. More formally, if $\overline{H}$ is disconnected, let $\widehat{\mathcal{C}}(H) = \{\widehat{C}_1, \ldots, \widehat{C}_\ell\}$. Given two vertex sets $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$, we consider the induced subgraph $H[\widehat{C}_u \cup \widehat{C}_v]$. We build a graph G_{uv} by taking $H[\widehat{C}_u \cup \widehat{C}_v]$ and removing all the fill edges between the two vertex sets $\widehat{C}_u$ and $\widehat{C}_v$. We define

$$G_{uv} = (\widehat{C}_u \cup \widehat{C}_v, E(H[\widehat{C}_u]) \cup E(H[\widehat{C}_v]) \cup E_{uv}),$$

where $E_{uv} = \{xy \mid x \in \widehat{C}_u, y \in \widehat{C}_v, xy \in E\}$. Let us consider now, the subgraphs of H induced by the vertex sets of the connected components of G_{uv}. We define $H_{uv} = \bigcup_{Y_i \in \mathcal{C}(G_{uv})} H[Y_i]$. Notice that if G_{uv} is connected, then $H_{uv} = H[\widehat{C}_u \cup \widehat{C}_v]$; otherwise H_{uv} is disconnected and $G[\widehat{C}_u \cup \widehat{C}_v] \subseteq G_{uv} \subseteq H_{uv} \subset H[\widehat{C}_u \cup \widehat{C}_v]$.

Lemma 4. *Let $H = (V, E \cup F)$ be a cograph completion of a graph $G = (V, E)$. H is a minimal cograph completion of G if and only if $H[\widehat{C}_i]$ is a minimal cograph completion of $G[\widehat{C}_i]$, for every $\widehat{C}_i \in \widehat{\mathcal{C}}(H)$ and G_{uv} is a connected graph for any two distinct $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$.*

As we are about to see, in order to check the connectivity of G_{uv} it is enough to consider only the edges between the co-connected components $\widehat{C}_u$ and $\widehat{C}_v$ and

not the ones that are inside $\widehat{C}_u$ and $\widehat{C}_v$. For that reason we introduce the graph G^*_{uv} which can be viewed as the graph obtained from G_{uv} by replacing every connected component of $H[\widehat{C}_u]$ and $H[\widehat{C}_v]$ with a single vertex.

We formally define the graph G^*_{uv} over the cotree $T(H)$ of H. Let t_u, t_v be two children of the root of $T(H)$. If $Q(t_u) \neq \emptyset$ then let $A_u = Q(t_u)$; otherwise let $A_u = \{t_u\}$. Similarly if $Q(t_v) \neq \emptyset$ then let $A_v = Q(t_v)$; otherwise let $A_v = \{t_v\}$. Observe that A_u and A_v contain nodes of $T(H)$. Given the two nodes t_u, t_v we define the graph G^*_{uv} as follows.

$$G^*_{uv} = (A_u \cup A_v, E^*_{uv}),$$

where $E^*_{uv} = \{(a_u, a_v) \in A_u \times A_v \mid (M(a_u) \times M(a_v)) \cap E \neq \emptyset\}$. In other words, edges are between A_u and A_v, and a vertex a_u of A_u is adjacent to a vertex a_v of A_v if and only if there is an edge $xy \in E$ such that $x \in M(a_u)$ and $y \in M(a_v)$. This also means that G^*_{uv} is a bipartite graph.

An example of the graphs G_{uv} and G^*_{uv} is give in Figure 1.

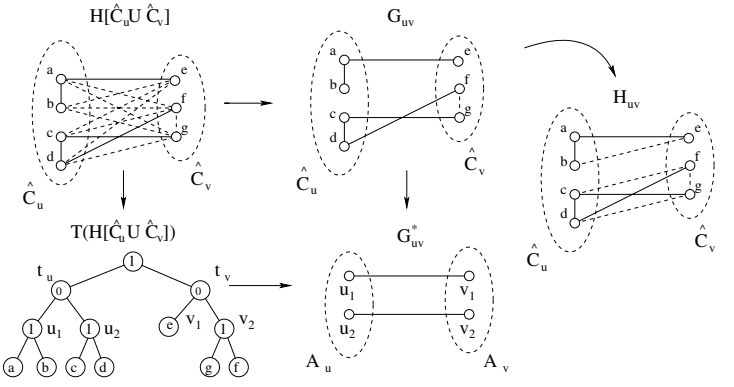


Fig. 1. *Example of G_{uv}, G^*_{uv} and H_{uv}. We can see how it is possible to obtain G_{uv} from $H[\widehat{C}_u \cup \widehat{C}_v]$ removing the fill edges between $H[\widehat{C}_u]$ and $H[\widehat{C}_v]$. Then we can think of G^*_{uv} as either obtained from G_{uv} contracting the connected component in each side, or directly defined on the cotree of $H[\widehat{C}_u \cup \widehat{C}_v]$ when we do not consider fill edges between the children of t_u and the children of t_v.*

Observation 5. *Let $T(H)$ be the cotree of a connected cograph completion H of G and let t be the root of $T(H)$ and $t_u, t_v \in Q(t)$. G^*_{uv} is connected if and only if G_{uv} is connected, where $\widehat{C}_u = M(t_u)$ and $\widehat{C}_v = M(t_v)$. Moreover, for any element $Y_i \in \mathcal{C}(G^*_{uv})$, $M(Y_i) \in \mathcal{C}(G_{uv})$.*

We can now rephrase Lemma 4 in terms of the cotree of H instead of H itself, and using G^*_{uv} instead of G_{uv}.

Theorem 6. *Let H be a cograph completion of a graph G and let $T(H)$ be its cotree. H is a minimal cograph completion of G if and only if for every 1-node t of $T(H)$ the graph G^*_{uv} is connected for any two nodes $t_u, t_v \in Q(t)$.*

It is quite straightforward to turn the previous theorem into a linear time algorithm for deciding whether a given cograph completion is minimal.

Theorem 7. *Let $H = (V, E \cup F)$ be a cograph completion of a graph $G = (V, E)$. Recognizing whether H is a minimal cograph completion of G can be done in $O(|V| + |E| + |F|)$ time.*

Using the previous theorem, we can give an algorithm for extracting a minimal cograph completion from a given one. The idea is very simple, on input G, H we use the algorithm from Theorem 7 to check whether H is a minimal cograph completion of G. If the answer is yes we can output H, while if the answer is no, there must be a 1-node t of $T(H)$ with children u and v such that the graph G^*_{uv} is disconnected. In that case H_{uv} is a cograph completion of $G[M(u) \cup M(v)]$ such that $H_{uv} \subset H[M(u) \cup M(v)]$. Thus $H' = (H \setminus (M(u) \cup M(v))) + H_{uv})$ is a cograph completion of G such that $H' \subset H$. We can now reiterate this process with H' as a candidate cograph completion. Since each such iteration can be done in $O(|V| + |E| + |F|)$ time and we remove at least one fill edge for each iteration, this algorithm runs in $O((|V| + |E| + |F|)|F|)$ time. One should notice that our extraction algorithm has many similarities with the generic extraction algorithm for sandwich monotone graph classes [14]. Similarly to sandwich monotonicity, our characterization states that in some sense, a cograph completion is minimal if and only if it is *locally* minimal.

Theorem 8. *Given a cograph completion $H = (V, E \cup F)$ of a graph $G = (V, E)$, a minimal cograph completion H' with $G \subseteq H' \subseteq H$, can be computed in time $O((|V| + |E| + |F|) \cdot |F|)$.*

4 Computing a Minimal Cograph Completion Directly

In this section we give an algorithm to solve the problem of computing a minimal cograph completion of an arbitrary input graph, in time linear in the size of the output graph. We use a vertex incremental scheme proposed in [13] for computing minimal cograph completions. At all times we maintain a minimal cograph completion of the part of the input graph that already has been considered. The algorithm starts off with the empty graph and adds in the vertices of the input graph one at a time, at each step updating the minimal completion by adding fill edges incident to the new vertex. The main technical part of this section is the design and analysis of an algorithm for one incremental step.

4.1 Adding a Vertex to a Cograph

In this section we give an algorithm for one incremental step of our completion algorithm. Hereafter we use $G = (V, E)$ to denote a cograph, unless otherwise specified. Given a vertex x together with a list of vertices $N_x \subseteq V$, we denote by G_x the graph obtained by adding x to G. That is, $G_x = (V \cup \{x\}, E \cup \{xy : y \in N_x\})$. Given a cograph G and a vertex set $N_x \subseteq V$ the algorithm computes

a vertex set $S \subseteq V$ such that $N_x \subseteq S$ and $H_x = (V \cup \{x\}, E \cup \{xy : y \in S\})$ is a minimal cograph completion of G_x.

The algorithm is fairly simple. We start off with G_x and consider $G = G_x - x$. If G is disconnected we only need to add edges to the connected components of G that x is already adjacent to. If x is adjacent to only one connected component we run the algorithm recursively on that connected component. However, if x is adjacent to more than one connected component of G we make x universal to all connected components of G that are adjacent to x. When G is connected we have to be a bit more careful. The basic idea is the following: We try adding x to a particular co-connected component $\widehat{C}$. To do this we have to make x universal to all other co-connected components of G and make sure not to make x universal to $\widehat{C}$. If we find out that x cannot be added to any co-connected component in this way, we make x universal to all co-connected components of G that x is adjacent to in G_x. In order to justify these choices, we will apply Theorem 3.

Algorithm: Minimal_x_Cograph_Completion – `MxCC` (G, N_x)
Input: A cograph G, and a set of vertices N_x which are to be made adjacent to a vertex $x \notin V$
Output: An inclusion minimal set $S \subseteq V$ such that $N_x \subseteq S$ and $H_x = (V \cup \{x\}, E \cup \{xy : y \in S\})$ is a cograph
if G is connected **then**
  if there are $\widehat{C}_i \in \widehat{\mathcal{C}}(G)$ and $C_j \in \mathcal{C}(G[\widehat{C}_i])$ s.t. $\widehat{C}_i \cap N_x \neq \emptyset$ and $C_j \cap N_x = \emptyset$ **then**
    $S = MxCC(G[\widehat{C}_i], N_x \cap \widehat{C}_i) \cup (V \setminus \widehat{C}_i)$;
  else
    $S = \bigcup_{\widehat{C}_i \in \widehat{\mathcal{C}}(G) \,:\, \widehat{C}_i \cap N_x \neq \emptyset} \widehat{C}_i$;
else
  if there is a $C_i \in \mathcal{C}(G)$ such that $N_x \subseteq C_i$ **then**
    $S = MxCC(G[C_i], N_x)$;
  else
    $S = \bigcup_{C_i \in \mathcal{C}(G) \,:\, C_i \cap N_x \neq \emptyset} C_i$;
return S;

Observe that the algorithm always terminates because each recursive call takes as an argument a subgraph of G induced by a strict subset of V. We are now ready to prove the correctness of Algorithm $MxCC$.

Lemma 9. *Given a cograph G and a set of vertices N_x, Algorithm $MxCC$ returns a set of vertices S such that H_x is a cograph completion of G_x.*

Observation 10. *If G is disconnected, $C_i \in \mathcal{C}(G)$, $N_x \cap C_i = \emptyset$, and $S = MxCC(G, N_x)$ then $S \cap C_i = \emptyset$.*

Theorem 11. *Given a cograph G and a set of vertices N_x, Algorithm $MxCC$ returns a set of vertices S such that H_x is a minimal cograph completion of G_x.*

4.2 Implementing Algorithm *MxCC* Using a Cotree Representation

In order to obtain a good running time for Algorithm $MxCC$ we give an algorithm that works directly on the cotree of the input graph. That is, we give an algorithm, namely $CMxCC$, that takes as an input the cotree $T(G)$ of a cograph G and a set N_x of vertices in G and returns a set of vertices S of G so that H_x is a minimal cograph completion of G_x. For a node t in $T(G)$, recall that $Q(t)$ is the set of t's children in $T(G)$. Let $Q_x(t) = \{t_i \in Q(t) : M(t_i) \cap N_x \neq \emptyset\}$. Thus $Q_x(t) \subseteq Q(t)$.

Algorithm: Cotree_Minimal_x_Cograph_Completion – CMxCC (T, N_x)
Input: A cotree T of a cograph $G = (V, E)$ and a set of vertices N_x which are to be made adjacent to a vertex $x \notin V$
Output: An inclusion minimal set $S \subseteq V$ such that $N_x \subseteq S$ and $H_x = (V \cup \{x\}, E \cup \{xy : y \in S\})$ is a cograph

$r = root(T)$;
if r is a 1-node **then**
  if there is a $t \in Q_x(r)$ such that $\emptyset \subset Q_x(t) \subset Q(t)$ **then**
    $S = CMxCC(T_t, N_x \cap M(t)) \cup (M(r) \setminus M(t))$;
  else
    $S = \bigcup_{t \in Q_x(r)} M(t)$;
else
  if there is a $t \in Q(r)$ such that $Q_x(r) \subseteq \{t\}$ **then**
    $S = CMxCC(T_t, N_x)$;
  else
    $S = \bigcup_{t \in Q_x(r)} M(t)$;
return S;

The correctness of the algorithm follows from the following two observations which imply that Algorithm $CMxCC$ returns the same set S as Algorithm $MxCC$.

Observation 12. *Let G be a connected cograph and let r be the root of $T(G)$. There are vertex sets $\widehat{C}_i \in \widehat{\mathcal{C}}(G)$ and $C_j \in \mathcal{C}(G[\widehat{C}_i])$ such that $\widehat{C}_i \cap N_x \neq \emptyset$ and $C_j \cap N_x = \emptyset$ if and only if there is a node $t \in Q_x(r)$ such that $\emptyset \subset Q_x(t) \subset Q(t) \neq \emptyset$.*

Observation 13. *Let G be a disconnected cograph and let r be the root of $T(G)$. There is a set $C_i \in \mathcal{C}(G)$ such that $N_x \subseteq C_i$ if and only if there is a node $t \in Q(r)$ such that $Q_x(r) \subseteq \{t\}$.*

Now we are ready to prove a bound on the running time for computing a minimal cograph completion H_x of G_x, and give the final theorem about the existence of an algorithm to compute a minimal cograph completion in time linear in the output graph.

Theorem 14. *Given a cograph G and its cotree $T(G)$, there is an algorithm for computing the set of vertices S that are adjacent to x in a minimal cograph completion of G_x which runs in $O(|S| + 1)$ time.*

Theorem 15. *There is an algorithm for computing a minimal cograph completion $H = (V, E \cup F)$ of an arbitrary graph $G = (V, E)$ in $O(|V| + |E| + |F|)$ time.*

5 Concluding Remarks

We have studied minimal cograph completions from two different points of view. Our results include an efficient algorithm for the computation problem and a precise characterization of minimal cograph completions. Such characterizations are rarely known for graph classes that do not have the sandwich monotone property. This makes our characterization and the consequent extraction algorithm particularly interesting. Observe that for other classes that are not sandwich monotone, like comparability and proper interval graphs, there exists a computation algorithm [13,25] while no characterization of minimal completions is known.

Three interesting problems we leave open are:

1. *Can one design a faster extraction algorithm, possibly linear in the size of the given completion?*
2. *Does there exist a computation algorithm running in time linear in the size of the input graph?*
3. *Is the problem of finding a minimum cograph completion of a graph obtained by adding one vertex to a cograph polynomial time solvable?*

To solve the first of these problems it might be enough to give a clever implementation of our naive extraction algorithm. For the second one, we can not use an explicit representation of the output graph H, since H can be much bigger than G. However, if we can accept the cotree of H as output, an algorithm with running time linear in the size of G could be achievable. Examples of completion algorithms that produce minimal completions in time linear in the size of input graph can be found in [12,14]. The third problem can be viewed as a generalization of the problem solved in [24].

References

1. Berry, A., Heggernes, P., Villanger, Y.: A vertex incremental approach for dynamically maintaining chordal graphs. Discrete Math. 306, 318–336 (2006)
2. Blair, J., Heggernes, P., Telle, J.A.: A practical algorithm for making filled graphs minimal. Theoretical Computer Science 250, 125–141 (2001)
3. Bodlaender, H.L., Koster, A.M.C.A.: Safe separators for treewidth. Discrete Math. 306, 337–350 (2006)
4. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. In: SIAM Monographs on Discrete Mathematics and Applications (1999)

5. Burzyn, P., Bonomo, F., Durán, G.: NP-completeness results for edge modification problems. Disc. Appl. Math. 154, 1824–1844 (2006)
6. Corneil, D.G., Lerchs, H., Stewart, L.K.: Complement reducible graphs. Disc. Appl. Math. 3, 163–174 (1981)
7. Corneil, D.G., Perl, Y., Stewart, L.K.: A linear recognition algorithm for cographs. SIAM J. Comput. 14, 926–934 (1985)
8. El-Mallah, E., Colbourn, C.: The complexity of some edge deletion problems. IEEE Transactions on Circuits and Systems 35, 354–362 (1988)
9. Fomin, F.V., Kratsch, D., Todinca, I.: Exact (Exponential) Algorithms for Treewidth and Minimum Fill-In. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 568–580. Springer, Heidelberg (2004)
10. Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. J. Comput. Bio. 2(1), 139–152 (1995)
11. Golumbic, M.C., Kaplan, H., Shamir, R.: Graph sandwich problems. J. Algorithms 19, 449–473 (1995)
12. Heggernes, P., Mancini, F.: Minimal Split Completions of Graphs. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 592–604. Springer, Heidelberg (2006)
13. Heggernes, P., Mancini, F., Papadopoulos, C.: Making Arbitrary Graphs Transitively Orientable: Minimal Comparability Completions. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 419–428. Springer, Heidelberg (2006)
14. Heggernes, P., Papadopoulos, C.: Single-Edge Monotonic Sequences of Graphs and Linear-Time Algorithms for Minimal Completions and Deletions. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 406–416. Springer, Heidelberg (2007)
15. Heggernes, P., Suchan, K., Todinca, I., Villanger, Y.: Characterizing Minimal Interval Completions. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 236–247. Springer, Heidelberg (2007)
16. Heggernes, P., Telle, J.A., Villanger, Y.: Computing minimal triangulations in time $O(n^{\alpha} \log n) = o(n^{2.376})$. In: Proceedings of SODA 2005 - 16th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 907–916 (2005)
17. Kashiwabara, T., Fujisawa, T.: An NP-complete problem on interval graphs. In: IEEE Symp. of Circuits and Systems, pp. 82–83 (1979)
18. Natanzon, A., Shamir, R., Sharan, R.: A polynomial approximation algorithm for the minimum fill-in problem. In: Proceedings of STOC 1998 - 30th Annual ACM Symposium on Theory of Computing, pp. 41–47 (1998)
19. Cai, L.: Fixed-Parameter Tractability of Graph Modification Problems for Hereditary Properties. Inf. Process. Lett. 58(4), 171–176 (1996)
20. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal and interval graphs: Minimum Fill-in and Physical Mapping. In: Proceedings of FOCS 2004 - 35th Annual Symposium on Foundations of Computer Science, pp. 780–791 (2004)
21. Heggernes, P., Paul, C., Telle, J.A., Villanger, Y.: Interval Completion is Fixed Parameter Tractable. In: Proceedings of STOC 2007 - 39th Annual ACM Symposium on Theory of Computing, pp. 374–381 (2007)
22. Dom, M., Guo, J., Hüffner, F., Niedermeier, R.: Error Compensation in Leaf Power Problems. Algorithmica 44(4), 363–381 (2006)
23. Natanzon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. Disc. Appl. Math. 113, 109–128 (2001)
24. Nikolopoulos, S.D., Palios, L.: Adding an Edge in a Cograph. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 214–226. Springer, Heidelberg (2005)

25. Rapaport, I., Suchan, K., Todinca, I.: Minimal Proper Interval Completions. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 217–228. Springer, Heidelberg (2006)
26. Rose, D.J.: A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In: Read, R.C. (ed.) Graph Theory and Computing, pp. 183–217. Academic Press, New York (1972)
27. Suchan, K., Todinca, I.: Minimal Interval Completion Through Graph Exploration. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 517–526. Springer, Heidelberg (2006)
28. Bodlaender, H.L., Kloks, T., Kratsch, D., Müller, H.: Treewidth and Minimum Fill-in on d-Trapezoid Graphs. J. Graph Algorithms Appl. 2(5), 1–28 (1998)
29. Kloks, T., Kratsch, D., Spinrad, J.: On treewidth and minimum fill-in of asteroidal triple-free graphs. Theor. Comput. Sci. 175(2), 309–335 (1997)
30. Broersma, H.J., Dahlhaus, E., Kloks, T.: A linear time algorithm for minimum fill-in and treewidth for distance hereditary graphs. Discrete Applied Mathematics 99(1), 367–400 (2000)
31. Kloks, T., Kratsch, D., Wong, C.K.: Minimum Fill-in on Circle and Circular-Arc Graphs. Journal of Algorithms 28(2), 272–289 (1998)
32. Mancini, F.: Minimum Fill-In and Treewidth of Split+ ke and Split+ kv Graphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 881–892. Springer, Heidelberg (2007)
33. Yannakakis, M.: Computing the minimum fill-in is NP-complete. SIAM J. Alg. Disc. Meth. 2, 77–79 (1981)
34. Berry, A., Heggernes, P., Simonet, G.: The Minimum Degree Heuristic and the Minimal Triangulation Process. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 58–70. Springer, Heidelberg (2003)
35. Bouchitté, V., Todinca, I.: Treewidth and Minimum Fill-in: Grouping the Minimal Separators. SIAM J. Comput. 31(1), 212–232 (2001)
36. Villanger, Y.: Improved Exponential-Time Algorithms for Treewidth and Minimum Fill-In. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 800–811. Springer, Heidelberg (2006)
37. Heggernes, P., Mancini, F.: Dinamically Mantaining Split Graphs. Tech report: http://www.ii.uib.no/~federico/papers/dynsplit-rev2.pdf
38. Bretscher, A., Corneil, D.G., Habib, M., Paul, C.: A Simple Linear Time LexBFS Cograph Recognition Algorithm. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 119–130. Springer, Heidelberg (2003)
39. Aho, A.V., Hopcroft, I.E., Ullman, J.D.: The design and analysis of computer algorithms, ex. 2.12, p. 71. Addison-Wesley, Reading (1974)

Efficient First-Order Model-Checking Using Short Labels*

Bruno Courcelle**, Cyril Gavoille, and Mamadou Moustapha Kanté

Université de Bordeaux, LaBRI, CNRS, France
{courcell,gavoille,mamadou.kante}@labri.fr

Abstract. We prove that there exists an $O(\log(n))$-labeling scheme for every first-order formula with free set variables in every class of graphs that is *nicely locally cwd-decomposable*, which contains in particular, the *nicely locally tree-decomposable classes*. For every class of graphs of *bounded expansion* we prove that every *bounded formula* has an $O(\log(n))$-labeling scheme. We also prove that every quantifier-free formula has an $O(\log(n))$-labeling scheme in graphs of bounded *arboricity*. Some of these results are extended to counting queries.

1 Introduction

The model-checking problem for a class of structures $\mathcal{C}$ and a logical language $\mathcal{L}$ consists in deciding, for given $S \in \mathcal{C}$ and for some fixed sentence $\varphi \in \mathcal{L}$ if $S \models \varphi$, i.e., if S satisfies the property expressed by φ. More generally, if φ is a formula with free variables $x_1, \ldots, x_m$ one asks whether $S \models \varphi(d_1, \ldots, d_m)$ where $d_1, \ldots, d_m$ are values given to $x_1, \ldots, x_m$. One may also wish to list the set of m-tuples $(d_1, \ldots, d_m)$ that satisfy φ in S, or simply count them.

Polynomial time algorithms for these problems (for fixed φ) exist for certain classes of structures and certain logical languages. In this sense graphs of bounded degree "fit" with first-order (FO for short) logic [17,7] and graphs of bounded tree-width or clique-width "fit" with monadic second-order (MSO for short) logic. Frick and Grohe [8,9,11] have defined *Fixed Parameter Tractable* (FPT for short) algorithms for FO model-checking problems on classes of graphs that may have unbounded degree and tree-width (Definitions and Examples are given in Section 4). We will also use graph classes introduced by Nešetřil and Ossona de Mendez [15].

We will use the same tools for the following labeling problem: let be given a class of graphs $\mathcal{C}$ and a property $P(x_1, \ldots, x_m, Y_1, \ldots, Y_q)$ of vertices $x_1, \ldots, x_m$ and of sets of vertices $Y_1, \ldots, Y_q$ of graphs in $\mathcal{C}$. We want two algorithms, an algorithm $\mathcal{A}$ that attaches to each vertex u of every n-vertex graph of $\mathcal{C}$ a label $L(u)$, defined as a sequence of 0's and 1's of length $O(\log(n))$ or $\log^{O(1)}(n)$, and an algorithm $\mathcal{B}$ that checks property $P(a_1, \ldots, a_m, W_1, \ldots, W_q)$ by using

* Research supported by the project "Graph decompositions and Algorithms (GRAAL)" of "Agence Nationale pour la Recherche".

** Member of "Institut Universitaire de France".

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 159–170, 2008.

the labels. This latter algorithm must take as input the labels $L(a_1), \ldots, L(a_m)$ and the sets of labels $L(W_1), \ldots, L(W_q)$ of the sets $W_1, \ldots, W_q$ and tell whether $P(a_1, \ldots, a_m, W_1, \ldots, W_q)$ is true. Moreover each label $L(u)$ identifies the vertex u in the graph, which is possible with a sequence of length $\lfloor \log(n) \rfloor$. An $f(n)$-*labeling scheme* for a class of structures $\mathcal{C}$ is a pair $(\mathcal{A}, \mathcal{B})$ of functions solving the labeling problem and using labels of size at most $f(n)$ for n-vertex graphs of $\mathcal{C}$. Results of this type have been established for MSO logic by Courcelle and Vanicat [5] and, for particular properties (connectivity queries, that are expressible in MSO logic) by Courcelle and Twigg in [4] and by Courcelle et al. in [2].

Let us review the motivations for looking for *compact labelings of graphs*. By *compact*, we mean of length of order less than $O(n)$, where n is the number of vertices of the graph, hence in particular of length $\log^{O(1)}(n)$.

In distributed computing over a communication network with underlying graph G, nodes must act according to their local knowledge only. This knowledge can be updated by message passing. Due to space constraints on the local memory of each node, and on the sizes of messages, a distributed task cannot be solved by representing the whole graph G in each node or in each message, but it must rather manipulate more compact representations of G. Typically, the routing task may use routing tables, that are sublinear in the size of G (preferably of poly-logarithmic size), and short addresses transmitted in the headers of messages (of poly-logarithmic size too). As surveyed in [12] many distributed tasks can be optimized by the use of labels attached to vertices. Such labels should be usable even when the network has node or link crashes. They are called *forbidden-set labeling* schemes in [4]. In this framework local informations can be updated just by transmitting to all surviving nodes the list of (short) labels of all defected nodes and links, so that the surviving nodes can update their local information, e.g., their routing tables.

Let us comment about using set arguments. The forbidden (or defective) parts of a network are handled as a set of vertices passed to a query as an argument. This means that algorithm $\mathcal{A}$ computes the labels once and for all, independently of the possible forbidden parts of the network. In other words the labeling supports node deletions from the given network. (Edge deletions are supported in the labelings of [2] and [4].) If the network is augmented with new nodes and links, the labels must be recomputed. We leave this incremental extension as a topic for future research. Set arguments can be used to handle deletions, but also constraints, or queries like "what are the nodes that are at distance at most 3 of X and Y" where X and Y are two specified sets of nodes.

2 Notations and Definitions

All graphs and relational structures are finite. Let $\varphi(x_1, \ldots, x_m, Y_1, \ldots, Y_q)$ be a FO formula with free FO variables among $x_1, \ldots, x_m$ and free set variables among $Y_1, \ldots, Y_q$. Set variables are allowed in FO formulas but are not quantified. They occur in atomic formulas of the form "$y \in Y_i$". Gaifman's Theorem

[10] and its stronger versions are valid for such formulas because "$y \in Y_i$" is the same as "$R_i(y)$ holds" where R_i is a unary relation representing Y_i.

Let S be a relational structure of type relevant with signature $\mathcal{R}$, $S = \langle D_S, (R_S)_{R\in\mathcal{R}}\rangle$ with domain D_S. A *labeling* of S is an injective mapping $J : D_S \to \{0,1\}^*$ (or into some more convenient set Σ^* where Σ is a finite alphabet). If $Y \subseteq D_S$ we let $J(Y)$ be the family $(J(y))_{y\in Y}$. Clearly Y is defined from $J(Y)$.

For a formula $\varphi(x_1,\ldots,x_m,Y_1,\ldots,Y_q)$ and a class of structures $\mathcal{C}$ we are interested in the construction of two algorithms $\mathcal{A}$ and $\mathcal{B}$ doing the following:

1. $\mathcal{A}$ constructs for each $S \in \mathcal{C}$ a labeling J of S such that $|J(a)| = O(\log(n))$ for every $a \in D_S$, where $n = |D_S|$.
2. If J is computed from S by $\mathcal{A}$, then $\mathcal{B}$ takes as input an $(m+q)$-tuple $(J(a_1),\ldots,J(a_m),J(W_1),\ldots,J(W_q))$ and says correctly whether:

$$S \models \varphi(a_1,\ldots,a_m,W_1,\ldots,W_q).$$

In this case we say that the pair $(\mathcal{A},\mathcal{B})$ defines an $O(\log(n))$-*labeling supporting* the query defined by φ for the structures in $\mathcal{C}$.

Labelings based on logical descriptions of queries have been defined by Courcelle and Vanicat [5] for MSO queries and graphs of bounded clique-width (whence also of bounded tree-width). Applications to distance and connectivity queries in graphs of bounded clique-width and in planar graphs have been given by Courcelle and Twigg in [4] and by Courcelle, Gavoille, Kanté and Twigg in [2]. In the present article, we consider classes of graphs of unbounded clique-width and in particular, classes that are *locally decomposable* (Frick and Grohe [8,9]) and classes of *bounded expansion* (Nešetřil and Ossona de Mendez [15]). So, MSO logic cannot be achieved, we are thus obliged to consider FO logic.

In this extended abstract we only consider vertex-labeled graphs. The extension to structures can be done in a standard way through the so-called *Gaifman graphs*. A Σ-labeled graph is $G = \langle V_G, edg_G(\cdot,\cdot), (lab_{a,G})_{a\in\Sigma}\rangle$ (vertices, edge relations and unary relation for vertex labels).

By replacing everywhere "clique-width", "local clique-width", etc. by "tree-width", "local tree-width", etc., one can handle formulas with edge-set quantifications.

Definition 1 (Logic)
An FO formula $\varphi(x_1,\ldots,x_m,Y_1,\ldots,Y_q)$ is *basic bounded* if for some $p \in \mathbb{N}$ we have the following equivalence for all graphs G, all $a_1,\ldots,a_m \in V_G$ and all $W_1,\ldots,W_q \subseteq V_G$

$$G \models \varphi(a_1,\ldots,a_m,W_1,\ldots,W_q) \text{ iff } G[X] \models \varphi(a_1,\ldots,a_m,W_1\cap X,\ldots,W_q\cap X)$$

for some $X \subseteq V_G$ such that $|X| \leq p$ and $a_1,\ldots,a_m \in X$. (If this is true for X, then $G[Y] \models \varphi(a_1,\ldots,a_m,W_1\cap Y,\ldots,W_q\cap Y)$ for every $Y \supseteq X$.)

An FO formula is *bounded* if it is a Boolean combination of basic bounded formulas. In particular, the negation of a basic bounded formula is not (in general) basic bounded, but it is bounded.

An FO formula $\varphi(x_1,\ldots,x_m,Y_1,\ldots,Y_q)$ is *t-local around* $(x_1,\ldots,x_m)$ if for every G and, every $a_1,\ldots,a_m \in V_G,\ W_1,\ldots,W_q \subseteq V_G$ we have

$$G \models \varphi(a_1,\ldots,a_m,W_1,\ldots,W_q) \text{ iff } G[N] \models \varphi(a_1,\ldots,a_m,W_1\cap N,\ldots,W_q\cap N)$$

where $N = N_G^t(a_1,\ldots,a_m) = \{y \in V_G \mid d(y,a_i) \leq t \text{ for some } i = 1,\ldots,m\}$ and $d(u,v)$ is the length of a shortest undirected path between u and v in G.

An FO sentence is *basic* (t,s)*-local* if it is equivalent to a sentence of the form

$$\exists x_1 \cdots \exists x_s \left(\bigwedge_{1\leq i<j\leq s} d(x_i,x_j) > 2t \ \wedge \bigwedge_{1\leq i\leq s} \psi(x_i) \right)$$

where $\psi(x)$ is t-local around its unique free variable x.

Remark. The query $d(x,y) \leq r$ is basic bounded ($p = r+1$) and t-local with $t = \lfloor r/2 \rfloor$. Its negation $d(x,y) > r$ is t-local and bounded (but not basic bounded).

3 Graphs

We are interested in on-line checking properties of networks in case of (reported) failures. Hence for each property of interest $\varphi(x_1,\ldots,x_m)$ we are not only interested in checking if $G \models \varphi(a_1,\ldots,a_m)$ by using $J(a_1),\ldots,J(a_m)$ where $a_1,\ldots,a_m \in V_G$ but also in checking $G\backslash X \models \varphi(a_1,\ldots,a_m)$ by using $J(a_1),\ldots,J(a_m)$ and $J(X)$ where $X \subseteq V_G - \{a_1,\ldots,a_m\}$ and $G\backslash X$ is the subgraph of G induced on $V_G - X$.

However, $G\backslash X \models \varphi(a_1,\ldots,a_m)$ for a FO formula $\varphi(x_1,\ldots,x_m)$ is equivalent to $G \models \varphi'(a_1,\ldots,a_m,X)$ and to $G_X \models \varphi''(a_1,\ldots,a_m)$ for FO formulas $\varphi'(x_1,\ldots,x_m,Y)$ and $\varphi''(x_1,\ldots,x_m)$ that are easy to write. We denote by G_X the graph G equipped with an additional vertex-label. Hence, we consider G_X as the structure G augmented with a unary relation *lab* such that $lab_{G_X}(u)$ holds iff $u \in X$. We will handle "holes" in graphs by means of set variables.

A graph has *arboricity at most* k if it is the union of k-edge disjoint forests (independently of the orientations of its edges).

Classes with *bounded expansion*, defined in [15] have several equivalent characterizations. We will use the following one: a class $\mathcal{C}$ has *bounded expansion* if for every integer p, there exists a constant $N(\mathcal{C},p)$ such that for every $G \in \mathcal{C}$, one can partition V_G in at most $N(\mathcal{C},p)$ parts such that any $i \leq p$ of them induce a subgraph of tree-width at most $i-1$. (For $i = 1$ this implies that each part is a stable set, hence the partition can be seen as a *proper vertex-coloring*.)

4 Locally Decomposable Classes

We refer to [16] and to [3,5] for the definitions of *tree-width* and of *clique-width* respectively. (We denote by $cwd(G)$ the clique-width of a graph G). We will use the same notations as in [8,9]. Definition 2 is analogous to [9, Definition 5.1].

Definition 2

1. The *local clique-width* of a graph G is the function $lcw^G : \mathbb{N} \to \mathbb{N}$ defined by $lcw^G(t) := \max\{cwd(G[N^t_G(a)]) \mid a \in V_G\}$.
2. A class $\mathcal{C}$ of graphs has *bounded local clique-width* if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that $lcw^G(t) \leq f(t)$ for every $G \in \mathcal{C}$ and $t \in \mathbb{N}$.

Examples

1. Every class of graphs of bounded clique-width has also bounded local clique-width since $cwd(G[A]) \leq cwd(G)$ for every $A \subseteq V_G$ (see [3]).
2. The classes of graphs of bounded local tree-width have bounded local clique-width since every class of graphs of bounded tree-width has bounded clique-width (see [3]). We can cite graphs of bounded degree and minor-closed classes of graphs that do not contain all apex-graphs (see [8,9]) as examples of classes of bounded local tree-width.
3. The class of unit-interval graphs has bounded local clique-width (using results from [14]) but neither bounded clique-width nor bounded local tree-width.
4. The class of interval graphs has not bounded local clique-width.

In order to obtain an $O(\log(n))$-labeling for certain classes of graphs of bounded local clique-width, we cover them as in [8,9], by graphs of small clique-width in a suitable way. In [8] a notion of *nicely locally tree-decomposable* class of structures was introduced. We will define a slightly more general notion.

Definition 3. Let $r, l \geq 1$ and $g : \mathbb{N} \to \mathbb{N}$. An (r, l, g)-*cwd cover* of a graph G is a family $\mathcal{T}$ of subsets of V_G such that:

1. For every $a \in V_G$ there exists a $U \in \mathcal{T}$ such that $N^r_G(a) \subseteq U$.
2. For each $U \in \mathcal{T}$ there exist less than l many $V \in \mathcal{T}$ such that $U \cap V \neq \emptyset$.
3. For each $U \in \mathcal{T}$ we have $cwd(G[U]) \leq g(1)$.

An (r, l, g)-cwd cover is *nice* if condition 3 is replaced by condition 3' below:

3'. For all $U_1, \ldots, U_q \in \mathcal{T}$ and $q \geq 1$ we have $cwd(G[U_1 \cup \cdots \cup U_q]) \leq g(q)$.

A class $\mathcal{C}$ of graphs is *locally cwd-decomposable* (resp. *nicely locally cwd-decomposable*) if there is a polynomial time algorithm that given a graph $G \in \mathcal{C}$ and $r \geq 1$, computes an (r, l, g)-cwd cover (resp. a nice (r, l, g)-cwd cover) of G for suitable l, g depending on r. (These two definitions are the same as in [9,8] where we substitute clique-width to tree-width.)

Examples

1. It is clear that every nicely locally cwd-decomposable class is locally cwd-decomposable and the converse is not true.
2. Each class of nicely locally tree-decomposable structures [8] is nicely locally cwd-decomposable.

3. Let G be a unit-interval graph. Using results from [14, Theorems 1,3 and Corollary 5] one can prove that G has an $(r, r, f(2r+1))$-cwd cover where f is the function that bounds local clique-width of unit-interval graphs. Thus every class of unit-interval graphs is locally cwd-decomposable.
4. Figure 1 shows inclusion relations between the many classes defined in Sections 3 and 4. It completes the diagram [9, Figure 2].

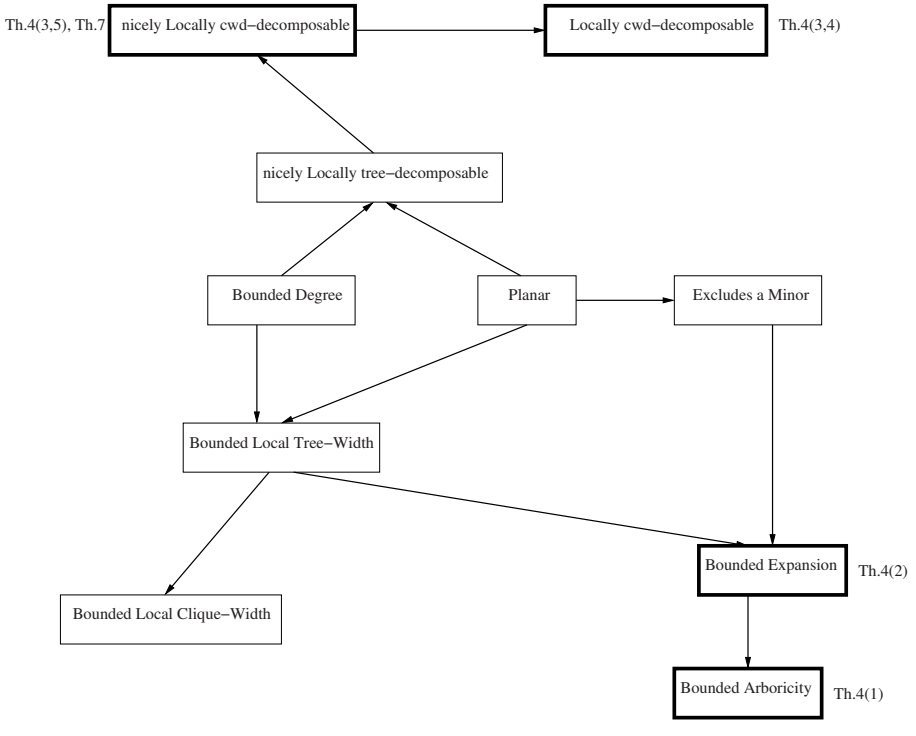


Fig. 1. Inclusion diagram indicating which results apply to which classes. An arrow means an inclusion of classes. Bold boxes are proved in this paper.

5 Results

The main results are as follows. In each case we consider labeled graphs over a finite set Σ of vertex-labels.

Theorem 4 (First Main Theorem). *There exist $O(\log(n))$-labeling schemes for the following queries and graph classes:*

1. *Quantifier-free queries in graphs of bounded arboricity.*
2. *Bounded FO queries for each class of graphs of bounded expansion.*
3. *Local queries with set arguments on locally cwd-decomposable classes.*

4. *FO queries without set arguments on locally cwd-decomposable classes.*
5. *FO queries with set arguments on nicely locally cwd-decomposable classes.*

We recall that for graphs G of clique-width at most k, there exists a cubic time algorithm that computes a cwd-term that defines G without being optimal [13]. (It uses $2^{k+1}-1$ labels, hence does not witness $cwd(G) \leq k$; however this term is enough for using [5].) And if a graph G has tree-width at most k, there exists a linear time algorithm that computes a tree-decomposition of width k of G [1]. We will also use results by Gaifman [10], Frick and Grohe [9,8] recalled below.

Theorem 5 ([10]). *Let $\varphi(\bar{x})$ be a FO formula where $\bar{x} = (x_1, \ldots, x_m)$. Then φ is logically equivalent to a Boolean combination $B(\varphi_1(\bar{u_1}), \ldots, \varphi_p(\bar{u_p}), \psi_1, \ldots, \psi_h)$ where:*

- *each φ_i is a t-local formula around $\bar{u_i} \subseteq \bar{x}$.*
- *each ψ_i is a basic (t', s)-local sentence.*

Moreover B can be computed effectively and, t, t' and s can be bounded in terms of m and the quantifier-rank of φ.

We will use a stronger form from [8]. Let $m, t \geq 1$. The *t-distance type* of an m-tuple $\bar{a}$ is the undirected graph $\epsilon = ([m], edg_\epsilon)$ where $edg_\epsilon(i,j)$ iff $d(a_i, a_j) \leq 2t+1$. The satisfaction of a t-distance type by an m-tuple can be expressed by a t-local formula:

$$\rho_{t,\epsilon}(x_1, \ldots, x_m) := \bigwedge_{(i,j) \in edg_\epsilon} d(x_i, x_j) \leq 2t+1 \ \wedge \bigwedge_{(i,j) \notin edg_\epsilon} d(x_i, x_j) > 2t+1.$$

We recall that Gaifman's Theorem and its variants extend to FO formulas with set variables.

Lemma 1 ([8]). *Let $\varphi(\bar{x}, Y_1, \ldots, Y_q)$ be a t-local formula around $\bar{x} = (x_1, \ldots, x_m)$, $m \geq 1$. For each t-distance type ϵ with $\epsilon_1, \ldots, \epsilon_p$ as connected components, one can compute a Boolean combination $F^{t,\epsilon}(\varphi_{1,1}, \ldots, \varphi_{1,j_1}, \ldots, \varphi_{p,1}, \ldots, \varphi_{p,j_p})$ of formulas $\varphi_{i,j}$ with FO free variables among those of $\bar{x}$ and set arguments in $\{Y_1, \ldots, Y_q\}$ such that:*

- *The FO free variables of each $\varphi_{i,j}$ are among $\bar{x} \mid \epsilon_i$ ($\bar{x} \mid \epsilon_i$ is the restriction of $\bar{x}$ to ϵ_i).*
- *$\varphi_{i,j}$ is t-local around $\bar{x} \mid \epsilon_i$.*
- *For each m-tuple $\bar{a}$, each q-tuple of sets $W_1, \ldots, W_q$, $G \models \rho_{t,\epsilon}(\bar{a}) \wedge \varphi(\bar{a}, W_1, \ldots, W_q)$ iff $G \models \rho_{t,\epsilon}(\bar{a}) \ \wedge \ F^{t,\epsilon}(\ldots, \varphi_{i,j}(\bar{a} \mid \epsilon_i, W_1, \ldots, W_q), \ldots)$.*

The lemma below is an easy adaptation of the results in [9].

Lemma 2 ([9]). *Let G be in a locally cwd-decomposable class. Every basic (t,s)-local sentence can be decided in polynomial time.*

We now give the proofs of each statement of Theorem 4 (except statement *1* because of space constraints). For clarity, we give them separately.

Proof (of Theorem 4 (2)). Let φ be a basic bounded formula with bound p and at least one free FO variable. We let $N = N(\mathcal{C}, p)$ and we partition V_G into $V_1 \uplus V_2 \uplus \cdots \uplus V_N$ as in the definition, $V_i \neq \emptyset$.

For every $\alpha \subseteq [N]$ of size p we let $V_\alpha = \bigcup_{i\in\alpha} V_i$ so that the tree-width of $G[V_\alpha]$ is at most $p-1$. Each vertex u belongs to less than $(N-1)^{p-1}$ sets V_α.

Hence a basic bounded formula $\varphi(x_1, \ldots, x_m, Y_1, \ldots, Y_q)$ is true in G iff it is true in some $G[X]$ with $|X| \leq p$, hence in some $G[V_\alpha]$ such that $x_1, \ldots, x_m \in V_\alpha$. For each α we construct a labeling J_α of $G[V_\alpha]$ (of tree-width at most $p-1$) supporting query φ by using [5]. We let $J(x) = \big(\ulcorner x \urcorner, \{(\ulcorner \alpha \urcorner, J_\alpha(x)) \mid x \in V_\alpha\}\big)$. We have $|J(x)| = O(\log(n))$.

We now explain how to decide φ by using the labels only. Given $J(a_1)$, $\ldots, J(a_m)$ we can determine all those sets α such that V_α contains $a_1, \ldots, a_m$. Using the components $J_\alpha(\cdot)$ of $J(a_1), \ldots, J(a_m)$ and the labels in $J(W_1), \ldots, J(W_q)$ we can determine if for some α, $G[V_\alpha] \models \varphi(a_1, \ldots, a_m, W_1 \cap V_\alpha, \ldots, W_q \cap V_\alpha)$ hence whether $G \models \varphi(a_1, \ldots, a_m, W_1, \ldots, W_q)$.

It remains to consider the case of a basic bounded formula of the form $\varphi(Y_1, \ldots, Y_q)$. For each α we determine the truth value t_α of $\varphi(\emptyset, \ldots, \emptyset)$ in $G[V_\alpha]$. The family of pairs (α, t_α) is of fixed size (depending on p) and is appended to $J(x)$ defined as above. From $J(W_1), \ldots, J(W_q)$ we get $D = \{\alpha \mid V_\alpha \cap (W_1 \cup \cdots \cup W_q) \neq \emptyset\}$.

By using the $J_\alpha(\cdot)$ components of the labels in $J(W_1) \cup \cdots \cup J(W_q)$ we can determine if for some $\alpha \in D$ we have $G[V_\alpha] \models \varphi(W_1 \cap V_\alpha, \ldots, W_q \cap V_\alpha)$. If one is found we conclude positively. Otherwise we look for some $t_\beta = True$ where $\beta \notin D$. This gives the final answer.

For a Boolean combination of basic bounded formulas $\varphi_1, \ldots, \varphi_t$ with associated labelings $J_1, \ldots, J_t$ we take the concatenation $J_1(x) \bullet J_2(x) \bullet \cdots \bullet J_t(x)$. It is of size $O(\log(n))$ and gives the desired result. □

Proof (of Theorem 4 (3)). Let $\varphi(\bar{x}, Y_1, \ldots, Y_q)$ be a t-local formula around $\bar{x} = (x_1, \ldots, x_m)$, $m \geq 1$. Then $G \models \varphi(\bar{a}, W_1, \ldots, W_q)$ iff $G[N_G^t(\bar{a})] \models \varphi(\bar{a}, W_1 \cap N_G^t(\bar{a}), \ldots, W_q \cap N_G^t(\bar{a}))$. Let ϵ be a t-distance type with $\epsilon_1, \ldots, \epsilon_p$ as connected components. By Lemma 1, $G \models \rho_{t,\epsilon}(\bar{a}) \wedge \varphi(\bar{a}, W_1, \ldots, W_q)$ iff $G \models \rho_{t,\epsilon}(\bar{a}) \wedge F^{t,\epsilon}(\varphi_{1,1}(\bar{a} \mid \epsilon_1, W_1, \ldots, W_q), \ldots, \varphi_{p,j_p}(\bar{a} \mid \epsilon_p, W_1, \ldots, W_q))$.

We let $\mathcal{T}$ be an (r, l, g)-cwd cover of G where $r = m(2t+1)$. We use such an r in order to warranty that if $a_1, \ldots, a_m$ are in a connected component of a t-distance type, there exists a $U \in \mathcal{T}$ such that $N_G^t(a_1, \ldots, a_m) \subseteq U$. For each vertex x there exist less than l many $V \in \mathcal{T}$ such that $x \in V$. We assume that each $U \in \mathcal{T}$ has an index encoded as a bit string $\ulcorner U \urcorner$. There are at most $n \cdot l$ sets in $\mathcal{T}$. Hence $\ulcorner U \urcorner$ has length $O(\log(n))$.

By the results of [5] we can label each vertex with a label $K(x)$ of length $O(\log(n))$ and decide in $O(\log(n))$-time if $d(u, v) \leq 2t+1$ or not by using $K(u)$ and $K(v)$[1]. We build a labeling K_U for each $U \in \mathcal{T}$; then for each x we let $K(x) = \Big(\ulcorner x \urcorner,\ \{\big(\ulcorner U \urcorner, K_U(x)\big) \mid N(x) \subseteq U\},\ \{\big(\ulcorner U \urcorner, K_U(x)\big) \mid N(x) \not\subseteq U\}\Big)$. where $N(x) = N_G^{2t+1}(x)$. (We always assume that $x \in N_G^t(x)$ for all $t \in \mathbb{N}$.)

[1] For checking if $d(u, v) \leq 2t+1$, an (r', l', g')-cwd cover suffices, with $r' = 2t+1$.

By [5] for each $\varphi_{i,j}(\bar{x} \mid \epsilon_i, Y_1, \ldots, Y_q)$ and each $U \in \mathcal{T}$ we can label each vertex $x \in U$ with $J^{\epsilon}_{i,j,U}(x)$ of length $O(\log(n))$ and decide $\varphi_{i,j}(\bar{a} \mid \epsilon_i, W_1, \ldots, W_q)$ in $G[U]$ by using $\left(J^{\epsilon}_{i,j,U}(b)\right)_{b \in \bar{a} \mid \epsilon_i}$ and $J^{\epsilon}_{i,j,U}(W_1 \cap U), \ldots, J^{\epsilon}_{i,j,U}(W_q \cap U)$. For each x we let

$$J_\epsilon(x) := \left\{ \left(\ulcorner U \urcorner, J^{\epsilon}_{1,1,U}(x), \ldots, J^{\epsilon}_{1,j_1,U}(x), \ldots, J^{\epsilon}_{p,1,U}(x), \ldots, J^{\epsilon}_{p,j_p,U}(x) \right) \mid N^t_G(x) \subseteq U \right\}.$$

It is clear that $|J_\epsilon(x)| = O(\log(n))$ since each x is in less than l many $V \in \mathcal{T}$. There exist at most $k' = 2^{k(k-1)/2}$ t-distance type graphs; we enumerate them by $\epsilon^1, \ldots, \epsilon^{k'}$. For each x we let $J(x) := (\ulcorner x \urcorner, K(x), J_{\epsilon^1}(x), \ldots, J_{\epsilon^{k'}}(x))$.

From the labels $K(x)$, we can determine $\{\ulcorner U \urcorner \mid U \in \mathcal{T},\ x \in U\}$, hence the sets $U \in \mathcal{T}$ such that $W \cap U \neq \emptyset$, $W \subseteq V_G$, where W is a set argument. It is clear that $J(x)$ is of length $O(\log(n))$ and is computed in polynomial time since $\mathcal{T}$ is computed in polynomial time and each J_ϵ is computed in polynomial time. We now explain how to decide whether $G \models \varphi(a_1, \ldots, a_m, W_1, \ldots, W_q)$ by using $J(a_1), \ldots, J(a_m)$ and $J(W_1), \ldots, J(W_q)$.

By using $K(a_1), \ldots, K(a_m)$ from $J(a_1), \ldots, J(a_m)$ we can construct the t-distance type ϵ satisfied by $a_1, \ldots, a_m$; let $\epsilon_1, \ldots, \epsilon_p$ be the connected components of ϵ. From each $J(a_i)$ we can recover $J_\epsilon(a_i)$. For each $\bar{a} \mid \epsilon_i$ there exists at least one $U \in \mathcal{T}$ such that $N^t_G(\bar{a} \mid \epsilon_i) \subseteq U$. We can recover them (there are less than l) from the $J(b)$, $b \in \bar{a} \mid \epsilon_i$. We can now decide whether $G \models F^{t,\epsilon}(\varphi_{1,1}(\bar{a} \mid \epsilon_1, W_1 \cap U_1, \ldots, W_q \cap U_1), \ldots, \varphi_{p,j_p}(\bar{a} \mid \epsilon_p, W_1 \cap U_p, \ldots, W_q \cap U_p))$ for some $U_1, \ldots, U_p$ determined from $J(a_1), \ldots, J(a_m)$. By using also $J(W_1), \ldots, J(W_q)$ we can determine the sets $W_i \cap U_j$ and this is sufficient by Lemma 1. □

Proof (of Theorem 4 (4)). Let $\varphi(x_1, \ldots, x_m)$ be a FO formula without set arguments. By Theorem 5 φ is equivalent to a Boolean combination $B(\varphi_1(\bar{x}), \ldots, \varphi_p(\bar{x}), \psi_1, \ldots, \psi_h)$ where φ_i is t-local and ψ_i is a basic (t', s)-local sentence for suitable t, t', s.

By Lemma 2 one can decide in polynomial time each sentence ψ_i. Let $b = (b_1, \ldots, b_h)$ where $b_i = 1$ if G satisfies ψ_i and 0 otherwise. For each $1 \leq i \leq p$ we construct a labeling J_i supporting query φ_i by Theorem 4 (3) (G belongs to a locally cwd-decomposable class and φ_i is a t-local formula around $\bar{x}$). For each x we let $J(x) := (\ulcorner x \urcorner, J_1(x), \ldots, J_p(x), b)$.

It is clear that $|J(x)| = O(\log(n))$. Since from b one can recover the truth value of each sentence ψ_i, we can decide whether $G \models \varphi(a_1, \ldots, a_m)$ by using $J(a_1), \ldots, J(a_m)$, the truth values of $\varphi_i(\bar{a})$ and b. □

Proof (of Theorem 4 (5)). By Theorem 4 (3) it is sufficient to consider FO formulas $\varphi(Y_1, \ldots, Y_q)$ of the form:

$$\exists x_1 \cdots \exists x_m \left(\bigwedge_{1 \leq i < j \leq m} d(x_i, x_j) > 2t \ \wedge \bigwedge_{1 \leq i \leq m} \psi(x_i, Y_1, \ldots, Y_q) \right)$$

where $\psi(x, Y_1, \ldots, Y_q)$ is t-local around x. We show how to check their validity by means of $O(\log(n))$-labelings.

We consider for purpose of clarity the particular case of $m = 2$. Let $\mathcal{T}$ be a nice (r, l, g)-cwd cover of G where $r = 2t + 1$. We let $K(U) = \{x \in U \mid N_G^{2t}(x) \subseteq U\}$ (the $2t$-kernel of U (see [8])).

We let γ be a distance-2 coloring of the intersection graph of $\mathcal{T}$ (vertices at distance 1 or 2 have different colors). For every 2 colors i, j we let $G_{i,j}$ be the graph induced by the union of the blocks $U \in \mathcal{T}$ of colors i and j.

Claim 1. *$cwd(G_{i,j}) \leq g(2)$.*

Proof (of Claim 1). $G_{i,j}$ is a disjoint union of sets U in $\mathcal{T}$ and of unions $U \cup U'$ with $U \cap U' \neq \emptyset$ for $U, U' \in \mathcal{T}$. This union is disjoint because if $U \cup U'$ with $U \cap U' \neq \emptyset$ would meet some $U'' \in \mathcal{T}$, $U'' \neq U$, $U'' \neq U'$, then we would have $\gamma(U) = i$, $\gamma(U') = j$ and U'' meets U or U'. It cannot have color i or j because γ is a distance-2 coloring. Since $cwd(G[U \cup U']) \leq g(2)$, we are done. □

Claim 2. *Let $x \in K(U)$ and $y \in K(U')$ for some $U, U' \in \mathcal{T}$. Then $d_G(x, y) > 2t$ iff $d_{G[U \cup U']}(x, y) > 2t$.*

Proof (of Claim 2). The "if direction" is clear since distance increases if we go to induced subgraphs.

For the "converse direction", we let $d_G(x, y) \leq 2t$; there exists a path of length $\leq 2t$ from x to y. This path is in $U \cup U'$ since $x \in K(U)$ and $y \in K(U')$. Hence it is also in $G[U \cup U']$, hence $d_{G[U \cup U']} \leq 2t$. □

Let us now give to each vertex x of G the smallest color i such that $x \in K(U)$ and $\gamma(U) = i$. Hence a vertex has one and only one color. For each pair i, j we consider the formula $\psi_{i,j}$ (possibly $j = i$):

$$\exists x, y \Big(d(x, y) > 2t \ \wedge \ \psi(x, Y_1, \ldots, Y_q) \wedge \psi(y, Y_1, \ldots, Y_q) \ \wedge \text{“}x \text{ has color } i\text{”} \ \wedge \text{“}y \text{ has color } j\text{”} \Big)$$

We use [5] to construct a labeling $J_{i,j}$ for the formula $\psi_{i,j}$ in the graph $G_{i,j}$ (with vertices colored by i or j, that is, we use new unary "color" predicates). We compute the truth value $b_{i,j}$ of $\psi_{i,j}(\emptyset, \ldots, \emptyset)$ in $G_{i,j}$; we get a vector $\boldsymbol{b}$ of fixed length. We also label each vertex x by its color. We concatenate to that b and the $J_{i,j}(x)$ for $x \in V_{G_{i,j}}$, giving $J(x)$.

From $J(W_1), \ldots, J(W_q)$ we can determine those $G_{i,j}$ such that $V_{G_{i,j}} \cap (W_1 \cup \cdots \cup W_q) \neq \emptyset$, and check if for one of them $G_{i,j} \models \psi_{i,j}(W_1, \ldots, W_q)$. If one is found we are done. Otherwise we use the $b_{i,j}$'s to look for $G_{i,j}$ such that $G_{i,j} \models \psi_{i,j}(\emptyset, \ldots, \emptyset)$ and $(W_1 \cup \cdots \cup W_q) \cap V_{G_{i,j}} = \emptyset$. This gives the correct results because of the following facts:

- If x, y satisfy the formula φ, then $x \in K(U)$, $y \in K(U')$ (possibly $U = U'$) and $d_G(x, y) > 2t$ implies $d_{G_{i,j}}(x, y) > 2t$, hence $G_{i,j} \models \psi_{i,j}(W_1, \ldots, W_q)$ where $i = \gamma(U)$ and $j = \gamma(U')$.
- If $G_{i,j} \models \psi_{i,j}(W_1, \ldots, W_q)$ then we get $G \models \varphi(W_1, \ldots, W_q)$ by similar argument (in particular $d_{G_{i,j}}(x, y) > 2t$ implies $d_{G[U \cup U']}(x, y) > 2t$ which implies that $d_G(x, y) > 2t$ by Claim 2).

For $m = 1$, the proof is similar with γ a proper (distance-1) coloring and we use G_i instead of $G_{i,j}$.

For the case $m > 2$, the proof is the same: one takes for γ a distance-m proper coloring of the intersection graph, one considers graphs $G_{i_1,\ldots,i_m}$ defined as (disjoint) unions of sets $U_1 \cup \cdots \cup U_m$ for $U_1, \ldots, U_m$ in $\mathcal{T}$, of respective colors $i_1, \ldots, i_m$ and $cwd(G[U_1 \cup \cdots \cup U_m]) \leq g(m)$. This terminates the proof of Theorem 4. □

Let us ask a very general question: what can be done with $O(\log(n))$ labels ? Here is a fact that limits the extension of these results.

Proposition 1. *There exists a constant $c > 0$ such that one cannot handle all local or bounded FO queries for n-vertex graphs of arboricity at most 2 with labels of size $c\sqrt{n}$.*

We now discuss extension to counting queries. Let $\varphi(x_1, \ldots, x_m, Y_1, \ldots, Y_q)$ be a MSO formula and S be a finite structure. For $W_1, \ldots, W_q \subseteq D_S$ we define

$$\#_S\varphi(W_1, \ldots, W_q) := \left| \{ (a_1, \ldots, a_m) \in D_S^m \mid S \models \varphi(a_1, \ldots, a_m, W_1, \ldots, W_q) \} \right|$$

A *counting query* consists in determining $\#_S\varphi(W_1, \ldots, W_q)$ for given $(W_1, \ldots, W_q)$. We will need the following extension of the results of [5].

Theorem 6. *Let $\varphi(x_1, \ldots, x_m, Y_1, \ldots, Y_q)$ be a MSO formula over labeled graphs and $k \in \mathbb{N}$. There exists an $O(\log^2(n))$-labeling scheme for n-vertex graphs of clique-width or tree-width at most k supporting the counting query $\#_G\varphi$. For computing $\#_G\varphi(W_1, \ldots, W_q)$ modulo some fixed integer s, or up to s (threshold counting) we need only labels of size $O(\log(n))$.*

We now state our second main theorem. The proof is omitted because of space constraints.

Theorem 7 (Second Main Theorem). *There exists an $O(\log^2(n))$-labeling scheme for counting queries based on FO formulas for nicely locally cwd-decomposable classes. $O(\log(n))$ is enough for modulo counting.*

We conjecture that the results of Theorem 4 (3,4,5) extend to classes of graphs excluding, or locally excluding a minor [6,11].

Question. Does there exist an $O(\log(n))$-labeling scheme for FO formulas with set arguments on locally cwd-decomposable classes ?

References

1. Bodlaender, H.L.: A Linear-Time Algorithm for Finding Tree-Decompositions of Small Tree-width. SIAM J. Comput. 25(6), 1305–1317 (1996)
2. Courcelle, B., Gavoille, C., Kanté, M.M., Twigg, A.: Optimal Labeling for Connectivity Checking in Planar Networks with Obstacles. (manuscript, 2008); An extended abstract will appear in Electronic Notes in Discrete Mathematics. In: Proceedings of the first Conference Topological and Geometric Graph Theory, Paris (2008)

3. Courcelle, B., Olariu, S.: Upper Bounds to the Clique-Width of Graphs. Discrete Applied Mathematics 101(1-3), 77–114 (2000)
4. Courcelle, B., Twigg, A.: Compact Forbidden-Set Routing. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 37–48. Springer, Heidelberg (2007)
5. Courcelle, B., Vanicat, R.: Query Efficient Implementation of Graphs of Bounded Clique-Width. Discrete Applied Mathematics 131(1), 129–150 (2003)
6. Dawar, A., Grohe, M., Kreutzer, S.: Locally Excluding a Minor. In: 22^{nd} IEEE Symposium on Logic in Computer Science (LICS), pp. 270–279. IEEE Computer Society, Los Alamitos (2007)
7. Durand, A., Grandjean, E.: First-Order Queries on Structures of Bounded Degree are Computable with Constant Delay. ACM Trans. Comput. Log 8(4) (2007)
8. Frick, M.: Generalized Model-Checking over Locally Tree-Decomposable Classes. Theory Comput. Syst. 37(1), 157–191 (2004)
9. Frick, M., Grohe, M.: Deciding First-Order Properties of Locally Tree-Decomposable Structures. J. ACM 48(1), 1184–1206 (2001)
10. Gaifman, H.: On Local and Non-Local Properties. In: Proceedings of the Herbrand Symposium Logic Colloquium 1981, pp. 105–135 (1982)
11. Grohe, M.: Logic, Graphs and Algorithms. In: Flum, Grädel, Wilke (eds.) Logic, Automata, History and Perspectives, pp. 357–422. Amsterdam University Press (2007)
12. Gavoille, C., Peleg, D.: Compact and Localized Distributed Data Structures. Distributed Computing 16(2-3), 111–120 (2003)
13. Hliněný, P., Oum, S.: Finding Branch-Decompositions and Rank-Decompositions. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 163–174. Springer, Heidelberg (2007)
14. Lozin, V.: Clique-Width of Unit Interval Graphs. arXiv:0709.1935 (manuscript, 2007)
15. Nešetřil, J., Ossona de Mendez, P.: Linear Time Low Tree-Width Partitions and Algorithmic Consequences. In: Kleinberg, J.M. (ed.) 38^{th} Annual ACM Symposium on Theory of Computing (STOC), pp. 391–400. ACM, New York (2006)
16. Robertson, N., Seymour, P.: Graph Minors V: Excluding a Planar Graph. J. Combin. Theory Ser. B 41(1), 92–114 (1986)
17. Seese, D.: Linear Time Computable Problems and First-Order Descriptions. Mathematical Structures in Computer Science 6(6), 505–526 (1996)

Matching for Graphs of Bounded Degree

Yijie Han

School of Computing and Engineering
University of Missouri at Kansas City
Kansas City, MO 64110, USA
hanyij@umkc.edu

Abstract. We show that there exists a matching with $\frac{4m}{5k+3}$ edges in a graph of degree k and m edges.

Keywords: Matching, lower bound.

1 Introduction

Matching is an extensively studied topic. The question we want to address here is the size of a maximum matching in a graph. Earlier researchers studied the problem of the existence of a perfect matching (i.e. a matching of size $n/2$ with n vertices in a graph). Petersen [5] showed that a bridgeless cubic graph has a perfect matching. König [3] showed that there exists a perfect matching in any k-regular bipartite graph. Tutte [6] characterizes when a graph has a perfect matching. For graphs without a perfect matching the size of a maximum matching is studied. Nishizeki and Baybars [4] showed that any 3-connected planar graphs has a matching of size at least $(n+4)/3$ for $n \geq 22$. Biedl et al. [1] raised the question whether a bound better than $m/(2k-1)$ can be obtained for the size of a maximum matching in a graph of m edges and degree k. Recently Feng et al. [2] showed a lower bound of $2m/(3k-1)$ (for $k \geq 3$) for this problem.

In this paper we give a lower bound of a $4m/(5k+3)$-size matching for a graph of m edges and degree k. Here the degree of a graph is the maximum degree of any vertex of the graph.

2 The Bound

Assume that a maximum matching M is obtained for the input graph with m edges. Vertices incident to an edge in the matching are saturated vertices. Vertices not incident to any edge in the matching are unsaturated vertices. Without loss of generality we can assume that the input graph is connected. If the graph has no unsaturated vertex then for each edge e in the matching we can have at most $1+(2k-2)$ edges incident to e. Among which 1 is the edge in the matching and $2k-2$ are nonmatching edges. However, when we are counting this way the nonmatching edges are counted twice, once from each vertex they are incident to. Therefore we have that $|M|(2+(2k-2)) \geq 2m$, i.e. $|M| \geq m/k$. Therefore we assume that there is at least one unsaturated vertex.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 171–173, 2008.

Since any alternating path starting from an unsaturated vertex a cannot end at another unsaturated vertex b (for otherwise an augmenting path exists), the only possibility for an alternating path to start from an unsaturated vertex and end at an unsaturated vertex is that the starting unsaturated vertex and the ending unsaturated vertex are the same vertex. If this happens and say such an alternating path is $a, a_1, a_2, ..., a_r, a$, then both a_1 and a_r have only one neighbor which is unsaturated (which is a in this case) for otherwise an augmenting path exists. In this case we remove edge (a_r, a) and therefore all alternating paths starting from an unsaturated vertex do not end at an unsaturated vertex. We have thus removed no more than $|M|$ nonmatching edges from the input graph. Let (a, b) be an edge in the matching, If a or b has a neighboring vertex which is unsaturated we say that (a, b) is an outer matching edge. Otherwise we say that (a, b) is an inner matching edge. Let M_1 be the set of outer matching edges and M_2 be the set of inner matching edges. For each matching edge in M_1 we will call the vertex of the edge which has an unsaturated neighbor the outer vertex and the vertex which has no unsaturated neighbors the inner vertex. There are no more than $|M_1|(k-1)$ nonmatching edges incident to outer vertices. Any nonmatching edge cannot be incident to two inner vertices for otherwise an augmenting path exists. We put nonmatching edges in three sets N_1, N_2, N_3. N_1 is the set of edges incident to an outer vertex. N_2 is the set of edges incident to an inner vertex and a vertex in M_2. N_3 is the set of edges incident to vertices in M_2 only. Then

$$|N_1| + |N_2| + |N_3| + |M| \geq m - |M|.$$

The number of nonmatching edges incident to vertices in M_2 is no more than $|M_2|(2k-2)$. Among which there are $2|N_3|$ edges incident to vertices in M_2 only and $|N_2|$ edges incident to vertices both in M_1 and M_2. Thus we have $|N_3| + |N_2|/2 \leq |M_2|(k-1)$. Also we have that $|N_1| \leq |M_1|(k-1)$ and $|M_1| + |M_2| = |M|$. Therefore we have that

$$|M_1|(k+1) + |M_2|(k+1) + |N_2|/2 \geq |N_1| + |N_2| + |N_3| + 2|M| \geq m.$$

We have that $|M_1|(k-1) \geq |N_2|$ (these edges cannot be incident to outer vertices). Now for each edge (a, b) in M_2 it cannot happen that one edge in N_2 is incident to a and an inner vertex c and another edge in N_2 is incident to b and another inner vertex d, for otherwise an augmenting path exists if $c \neq d$. If in this case $c = d$ then both a and b cannot be incident to other edges in N_2 besides (a, c) and (b, d). Therefore we partition M_2 into M_{21}, M_{22} and M_{23} with edges in M_{21} having only one vertex of the edge incident to edges in N_2 and edges in M_{22} having both vertices incident to edges in N_2 (i.e., an inner vertex is the only inner vertex neighbor of bother vertices of the edge in M_{22}) and edges in M_{23} has no vertex incident to edges in N_2 (i.e. they are incident to edges in N_3 only).

We have that $|M_{21}|(k-1) + 2|M_{22}| \geq |N_2|$. Therefore we have that $|M_2|(k-1) \geq |N_2|$ if $k \geq 3$. Now we have that $(1/2)|M_1|(k-1) + (1/2)|M_2|(k-1) \geq |N_2|$. Thus we have that

$$|M_1|(k+1)+|M_2|(k+1)+(1/4)|M_1|(k-1)+(1/4)|M_2|(k-1)=((5k+3)/4)|M|\geq m.$$

Therefore we obtain that $|M| \geq 4m/(5k+3)$ if $k \geq 3$.
For $k = 1, 2$, $|M| \geq 4m/(5k+3)$ obviously holds. Therefore we have

Theorem 1. For a graph of m edges and degree k there exists a matching of size $4m/(5k+3)$.

A reviewer of this paper posted the following question: Is there an upper bound of the form cm/k for some c? The answer to this question is negative. Consider the graph of vertex set $\{v_0, v_1, v_2, ..., v_k\}$ (assume that k is even) and edge set $\{(v_0, v_1), (v_0, v_2), ..., (v_0, v_k), (v_1, v_2), (v_3, v_4), ..., (v_{k-1}, v_k)\}$. This graph has $m = 3k/2$. The maximum matching has size $k/2 = m/3$. Thus cm/k cannot be used as an upper bound.

References

1. Biedl, T., Demaine, E., Duncan, C., Fleischer, R., Kobourov, S.: Tight bounds on the maximal and maximum matchings. Discrete Math. 285(1-3), 7–15 (2004)
2. Feng, W., Qu, W., Wang, H.: Lower bounds on the cardinality of maximum matchings in graphs with bounded degrees. In: Proc. 2007 Int. Conf. on Foundations of Computer Science, Las Vegas, pp. 110–113 (2007)
3. König, D.: Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. Math. Ann. 77, 453–456 (1916)
4. Nishizeki, T., Baybars, I.: Lower bounds on the cardinality of the maximum matchings of planar graphs. Discrete Math. 28(3), 255–267 (1979)
5. Petersen, J.: Die Theorie der regulären graphs (The theory of regular graphs). Acta Math. 15, 193–220 (1891)
6. Tutte, W.T.: The factorization of linear graphs. J. London Math. Soc. 22, 107–111 (1947)

Searching Trees with Sources and Targets*

Chris Worman and Boting Yang

Department of Computer Science, University of Regina
{worman2c,boting}@cs.uregina.ca

Abstract. We consider a new pursuit-evasion problem on trees where a subset of vertices, called *sources*, are initially occupied by searchers. We also consider the scenario where some of the searchers must end their search at certain vertices called *targets*. We incrementally consider such problems, first considering only sources, then only targets, and finally we consider the case where there are both sources and targets. For each case we provide a polynomial-time algorithm for computing the search number, i.e. the minimum number of searchers required to clear the tree, and an optimal search strategy. We also demonstrate that each search model is monotonic, i.e. for each case their exists an optimal search strategy such that the set of cleared edges grows monotonically as the search progresses.

1 Introduction

Imagine a scenario where a group of police officers is attempting to capture a fugitive who is hiding in a building. The police officers could enter the building and hope to capture the fugitive by exploring the building in an ad hoc manner. A better approach is to systematically explore the building to ensure the fugitive is captured. If we can guarantee that we can capture the fugitive by carefully choosing our search strategy, the following question becomes relevant: "What is the minimum number of police officers required to capture the fugitive?" This question motivates so-called *graph searching* problems, which are pursuit-evasion problems that take place on graphs.

In a typical graph searching problem introduced by Megiddo et al. [4], a group of searchers must capture an exceedingly fast and clever fugitive that is hiding on a graph. The graph is meant to represent some real-world domain, such as the hallways and rooms in a building. The searchers proceed by clearing edges, i.e. visiting edges to guarantee that the fugitive is not on the edge; initially all edges are *dirty*. Three searcher actions are allowed: (1) place a searcher on a vertex in the graph, (2) remove a searcher from a vertex, and (3) slide a searcher along an edge from one end vertex to the other. An edge uv is *cleared* when either (1) at least two searchers are located on a vertex u, and one of them slides along uv, or (2) a searcher is located on u, and all edges incident on u, except uv, are clear and the searcher slides along uv. A *search strategy* is a sequence of searcher actions that

* Research was supported in part by NSERC.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 174–185, 2008.

result in every edge in the graph being cleared. Researchers are often interested in computing the *search number* of a graph G, denoted $s(G)$, which is the minimum number of searchers required to clear G. A search strategy is called optimal if at each step it uses at most $s(G)$ searchers. Another important consideration for searching problems is that of monotonicity. Define A_i to be the set of edges that are clear after the i^{th} action of some searching strategy. We define $A_0 = \emptyset$. A search strategy $S = \{m_1, m_2, \ldots, m_r\}$ is called *monotonic* if $A_{i-1} \subseteq A_i$, for all $1 \leq i \leq r$. A searching problem is in turn called *monotonic* if there exists an optimal monotonic search strategy for each instance of the searching problem. For non-monotonic search strategies, if a searcher is removed or slides from a vertex such that the resulting graph contains a path, which is not occupied by any searchers, connecting a dirty edge to a clear edge then the clear edge becomes *recontaminated* instantaneously because the fugitive moves exceedingly fast.

Graph searching was first studied by Parsons [5], who studied a continuous version of the graph searching problem. This model was subsequently discretized by Megiddo et al. [4], who showed that deciding the search number of a graph is NP-hard [4]. The monotonicity result of LaPaugh [3] demonstrates that deciding the search number is in NP, and hence deciding the search number of a graph is NP-complete. Megiddo et al. [4] also provided an $O(n)$ time algorithm for computing the search number of a tree that can be extended to compute an optimal search strategy in $O(n \log n)$ time. Subsequently, Peng et al. [6] improved this result by giving an $O(n)$ time algorithm for computing an optimal search strategy of a tree. The search number of a graph has been shown to be related to some important graph parameters, such as vertex separation number and pathwidth. Other graph searching models have been studied; see [1,2] for surveys.

All the searching problems discussed in this paper take place on trees. In the problems that we study, two disjoint subsets of vertices of a tree T have been identified: the *sources*, denoted V_s, and the *targets*, denoted V_t. We use V_s and V_t to define a constrained graph searching problem as follows. At the beginning of the search, each source vertex is occupied by exactly one searcher. Furthermore, for the entire duration of the search strategy, each source vertex remains clear, i.e. is occupied by a searcher or all incident edges are clear. Also, once a searcher occupies a vertex $v \in V_t$, v must be occupied by at least one searcher for the remainder of the search strategy. We refer to such searching problems as *Source Target Searching (STS)* problems. In an STS problem, we call the searchers that were initially on sources *starting searchers*. All other searchers are called *additional searchers*. During the progression of a search strategy, a starting searcher may be removed. A starting searcher is called *free* when it is not currently occupying a vertex on T.

One of our primary motivations for studying STS problems comes from an algorithm development point of view: our algorithms for STS problems can be used as subroutines for searching algorithms for other classes of graphs, such as cycle-disjoint graphs [7], in which lots of induced subgraphs are trees. In this application, the starting searchers are placed as a result of some other algorithm. The algorithms presented herein can then be used to complete the search while

still ensuring that the already cleared portions of the graph remain clear. Target vertices represent vertices where certain searchers must terminate. In this application, the target vertices represent portals to other parts of the graph that are contaminated, but that we are not willing to visit at this point in the search. Thus a target searching algorithm can be used as a subroutine for searching algorithms for other classes of graphs that first clear an induced subgraph that is a tree, and then clear the remaining parts of the graph.

Another motivation for studying STS problems comes from scenarios where the graph contains vertices that are portals to uncontaminated or already cleared areas of the graph. For example, consider the scenario where the graph models a network of city streets where a fugitive is hiding. In this case, we may be able to restrict our search to a particular neighborhood. We can place starting searchers at intersections connecting the neighborhood with the rest of the city. These intersections are modeled by the source vertices in our searching model. The source vertices are initially protected by the starting searchers, and remain protected throughout the search, ensuring the fugitive remains in the neighborhood that we are searching. Extending this example to target vertices, we may wish to search for the fugitive "neighborhood by neighborhood." In this scenario, we first search a particular neighborhood, ensuring that when a searcher encounters an intersection leading another neighborhood, this searcher stays at the intersection until the original neighborhood is cleared. These intersections are modeled by the target vertices in an instance of the STS problem. Thus STS problems can be seen as a particular type of subgraph searching.

In Section 2 we describe an $O(|V_s|n)$ time algorithm for computing the search number of a tree when the tree contains sources, but does not contain any targets. In Section 3 we demonstrate an inverse relationship between searching with sources and searching with targets. We exploit this relationship in order to use the algorithm from Section 2 to compute the search number of a tree that contains only targets in $O(|V_t|n)$ time. In Section 4 we combine these results and extend the algorithm from Section 2 so that we can compute the search number of a tree with both sources and targets in $O(|V_s||V_t|n)$ time. In all cases, we show that the algorithms only output monotonic search strategies, thus establishing the monotonicity of each search model. Due to space constraints, some details have been omitted.

2 Searching Trees with Sources

In this section we consider the STS problem on trees that contain only sources. We call this type of searching *Source Searching*, or simply *SS*. Let T be a tree and $V_s \subseteq V(T)$ be a set of source vertices such that each vertex in V_s is initially occupied by a starting searcher. We define the *source search number* with respect to T an V_s, denoted $ss(T, V_s)$, as the minimum number of additional searchers required to clear T. We deem a vertex v to be *clear* if all edges incident with v are clear. A searcher occupying a vertex v in V_s is *moveable* if all but one edges incident with v are clear. We can formally state our searching problem as follows:

Source Searching (SS)
Instance: A tree T and a set of source vertices $V_s \subseteq V(T)$, such that each $v \in V_s$ is occupied by exactly one searcher.

Question: Is there a search strategy of T using k additional searchers such that each vertex from V_s remains clear during the entire search strategy?

Recall that in the SS problem, all edges are initially dirty. In order to make our analysis more simple, we actually describe an algorithm for a more general searching problem, which we call *Partial-Clear Source Searching (PSS)*, which allows certain subtrees to be clear at the beginning of the problem. In order to define this new searching problem, we require some notation. Given a tree T and a subset of vertices $V_s \subseteq V(T)$, define $T \ominus V_s$ to be the set of maximal induced subtrees $\{T_1, T_2, ..., T_k\}$ of T such that if $v \in V_s$ and $v \in T_i$, then v is a leaf in T_i, and $\bigcup\{T_i\} = T$, $1 \leq i \leq k$.

Partial-Clear Source Searching (PSS)
Instance: A tree T with a set of cleared edges $E_c \subseteq E(T)$, and a set of source vertices $V_s \subseteq V(T)$, such that each $v \in V_s$ is occupied by exactly one searcher and each subtree in $T \ominus V_s$ is either completely clear or completely dirty.

Question: Is there a search strategy of T using k additional searchers such that each vertex from V_s remains clear during the entire search strategy?

The associated search number for this searching problem is denoted $pss(T, V_s, E_c)$. One can see that an instance of the SS problem is a valid instance of the PSS problem with $E_c = \emptyset$. Moreover, since the two search problems have the same objective, any algorithm for the PSS problem can be used to solve the SS problem. Thus, for the remainder of this section we focus on the PSS problem, keeping in mind that the results also hold for the SS problem.

Our algorithm for computing $pss(T, V_s, E_c)$ recursively performs two major steps in order to clear T: firstly, the algorithm calls a subroutine called `Reposition`, which moves some of the searchers occupying sources, and secondly the algorithm clears a specially chosen subtree of T. Specifically, the algorithm will clear the dirty subtree from $T \ominus V_s$ with the smallest search number. Then a recursive call is made in order to search the remaining tree. The algorithm is formally stated in Figures 1 and 2. The algorithm makes use of a *global* variable f that records how many of the starting searchers have been removed thus far, i.e. f is equal to the number of free starting searchers. Thus, $f = 0$ initially. The algorithm computes a value s_{max}, which we claim is equal to $pss(T, V_s, E_c)$.

Now we turn our attention to proving the correctness of our algorithm. Intuitively, our proof approach is the following. Firstly, Lemma 1 essentially proves that the actions performed by the `Reposition` algorithm are part of some optimal strategy. Secondly, Lemma 2 essentially shows that $pss(T, V_s, E_c)$ is at least as large as the search number of the first subtree that the algorithm clears. Since our algorithm is recursive, once these two Lemmas are shown, the correctness of our algorithm follows. We begin with the following observation:

`Algorithm Reposition`(T, V_s, E_c)

1. `While there exists a moveable searcher` λ `on vertex` $v \in V_s$`:`
   (a) `Let` vu `be the only dirty edge incident with` v.
   (b) `Slide` λ `from` v `to` u`, clearing the edge` vu`, and` $V_s \leftarrow (V_s - \{v\}) \cup \{u\}$, $E_c \leftarrow E_c \cup \{vu\}$.
   (c) `If` u `contains two searchers then remove` λ `and` $f \leftarrow f + 1$.
2. `While there exists a searcher` λ `on a clear vertex` $v \in V_s$`:`
   (a) `Remove` λ `and` $V_s \leftarrow V_s - \{v\}$.
   (b) $f \leftarrow f + 1$.

Fig. 1. The algorithm for repositioning the starting searchers

`Algorithm ST-S`(T, V_s, E_c) (Search Tree with Sources)

1. `If` T `is clear then return 0.`
2. `Call Reposition`(T, V_s, E_c).
3. `Compute` $T \ominus V_s$`, discarding all clear subtrees, to obtain` $\{T_1, T_2, ..., T_k\}$. `Without loss of generality, suppose that` $s(T_1) \leq s(T_i)$`, for` $2 \leq i \leq k$.
4. `Clear all edges of` T_1 `using` $s(T_1)$ `searchers and` $E_c \leftarrow E_c \cup E(T_1)$.
5. $s_{max} \leftarrow max\{s(T_1) - f,$ `ST-S`$(T, V_s, E_c)\}$.
6. `Return` s_{max}.

Fig. 2. The algorithm for computing $pss(T, V_s, E_c)$

Lemma 1. *Let* (T, V_s, E_c) *be an instance of the PSS problem. If* V_s' *and* E_c' *are the results of running the* `Reposition`(T, V_s, E_c) *algorithm, then* $pss(T, V_s', E_c') \leq pss(T, V_s, E_c) + f$*, where* f *is the number of starting searchers removed by the* `Reposition`(T, V_s, E_c) *algorithm.*

Proof. Let S be an optimal PSS strategy for (T, V_s, E_c). We construct a PSS strategy S' for (T, V_s', E_c') that does exactly what S does, except in a few cases. Before describing these cases, we introduce some notation and make some observations. Let λ_v be the starting searcher located on vertex v. Define $(v, v^1, v^2, ..., v^l)$ to be the path that λ_v slides along during the execution of the `Reposition`(T, V_s, E_c) algorithm. From the condition of the while loop in step 1 of the `Reposition` algorithm, we know that every v^i, $1 \leq i \leq l - 1$, has degree two. Notice that the path $(v, v^1, ..., v^l)$ is dirty in (T, V_s, E_c) and has been cleared in (T, V_s', E_c').

Now we describe the changes made to S in order to obtain S'. The following change ensures that λ_v is located on T so that it can mimic the movements of λ_v in S.

1. For each λ_v that was removed by the `Reposition` algorithm, S' begins by placing λ_v on v^l.

The next change ensures that $v^l \in V_s'$ does not become incident with a dirty edge while it is unoccupied in S:

2. Whenever a searcher λ_* is placed on or slides to a vertex $v^i \in \{v, v^1, v^2, ..., v^{l-1}\}$, in S' we place λ_* on v^l, unless in S' λ_* is already occupying v^l, in which case in S' we do nothing. If in S λ_* is subsequently removed from v^i or slides to a vertex not in $\{v, v^1, v^2, ..., v^{l-1}, v^l\}$, then λ_* is removed from v^l, placed on v^i, and then λ_* performs this action.

Let $a_i \in S$ be the first action that either removes or slides λ_v. In S', prior to performing a_i, we do the following:

3. Remove λ_v from v^l, place λ_v on v, and then perform a_i.

Notice that since S clears T, then S' clears T. In order to complete the proof we must show that S' uses at most $pss(T, V_s, E_c) + f$ additional searchers and that all members of V_s' remain clear during the progression of S'. Notice that in S', the only new searchers that are added are those that were removed by the `Reposition` algorithm (see change (1) above), and there are exactly f searchers removed by the `Reposition` algorithm. Hence S' uses $pss(T, V_s, E_c)+f$ searchers.

All that remains is to demonstrate that all members of V_s' remain clear during the progression of S'. We can focus on the actions introduced in the changes given above since in all other cases, S' performs the same actions as in S. Specifically, we must consider the case where λ_v is removed from v^l in change (3), and when λ_* is removed from v^l (and then placed on v^i) in change (2). In either case, we have that in S there is no searcher occupying a vertex from $\{v, v^1, v^2, ..., v^{l-1}\}$ when λ_v performs a_i, otherwise this searcher would have been placed on v^l in change (2). Thus in S the dirty edge e is connected to $v \in V_s$ via a path containing no searchers, and v is unoccupied, which is a contradiction. □

In the following lemma, T_1 is defined as in step 3 of the `ST-S` algorithm; that is, after the call to the `Reposition` algorithm.

Lemma 2. *For any instance (T, V_s, E_c) of the PSS problem, $pss(T, V_s) \geq s(T_1) - f$, where T_1 is the subtree computed after steps 1-3 of the* `ST-S`(T, V_s, E_c) *algorithm, and f is the number of free starting searchers removed during the call to* `Reposition`(T, V_s, E_c) *in step 2 of the* `ST-S`(T, V_s, E_c) *algorithm.*

Proof. Let V_s' and E_c' be the results of running the `Reposition`(T, V_s, E_c) algorithm. By Lemma 1, we have that $pss(T, V_s', E_c') - f \leq pss(T, V_s, E_c)$, and hence it suffices to show that $pss(T, V_s', E_c') \geq s(T_1)$. Let S^* be any optimal PSS strategy for the instance (T, V_s', E_c'). Define T^* to be the first subtree from $\{T_1, T_2, ..., T_k\}$ (as defined in the algorithm) that S^* completely clears[1]. Recall that $s(T_1) \leq s(T_i)$, for $2 \leq i \leq k$. If S^* only uses additional searchers to clear T^* then $pss(T, V_s', E_c') \geq s(T*) \geq s(T_1)$ and the Lemma holds. Thus we only need to consider the case where S^* uses at least one starting searcher to aid in clearing T^*. In what follows, we show that this implies a contradiction by demonstrating that for each starting searcher used to clear T^*, there exists a corresponding additional searcher on the tree T.

[1] S^* may have cleared other edges of T prior to completely clearing T^*.

We begin by demonstrating a property of S^* that we will use in the rest of the proof. Suppose that at some point during S^*, a source vertex $v \in V_s$ is occupied, but is not occupied by the original starting searcher λ_v that occupied v in the instance (T, V_s', E_c'). We can adjust S^* as follows: if λ_v is removed from v, then either all edges incident with v are clear, in which case no searcher needs to occupy v for the remainder of S^*, or v is occupied by another searcher, and we can remove this searcher instead of removing λ_v. If λ_v slides off v at some point, then after it slides off either all incident edges are clear, in which case no searcher needs to occupy v for the remainder of S^*, or there is another searcher occupying v, and we can slide this searcher instead of λ_v. So without loss of generality, during the progression of S^*, if a source vertex $v \in V_s$ is occupied, then it is occupied by λ_v, where λ_v is the starting searcher that occupies v in the instance (T, V_s', E_c'). We use this fact throughout the remainder of the proof.

Define $\{\lambda_1, \lambda_2, ..., \lambda_b\}$ to be the set of starting searchers occupying T^* at some moment during S^*. We define a procedure for associating a unique additional searcher with each starting searcher in $\{\lambda_1, \lambda_2, ..., \lambda_b\}$, but first we require some definitions. We begin by defining two sets of source vertices V^* and U^*, which form a partition of the currently unoccupied source vertices. Define $V^* = \{v_1, v_2, ..., v_b\}$ to be the set of source vertices that $\{\lambda_1, \lambda_2, ..., \lambda_b\}$ initially occupied. Define $U^* = \{u_1, u_2, ..., u_k\}$ to be the set of unoccupied source vertices whose original starting searchers are not currently occupying T^*. For a subset of source vertices $V_s' \subseteq V_s$, define $P(V_s') = \{T_{i_1}, T_{i_2}, ..., T_{i_k}\}$ to be the subset of trees from $\{T_1, T_2, ..., T_k\}$ such that for each $T_i \in P(V_s')$ there exists a $v \in V_s'$ such that v is a leaf in T_i.

We define two graphs G and H as folllows. Define $G = (V(G), E(G))$ as $V(G) = V^* \cup P(V^*)$ and $\{v, T_i\} \in E(G)$ if v is a leaf in T_i. Define $H = (V(H), E(H))$ as $V(H) = U^* \cup P(U^*)$, and $\{u, T_i\} \in E(G)$ if u is a leaf in T_i. One can easily show that both G and H are forests since T is a tree.

Now consider the following procedure for assigned a unique additional searcher, which is currently occupying T, to each of $\{\lambda_1, \lambda_2, ..., \lambda_b\}$. Let T_i be a leaf of G (notice that leaves in G cannot be source vertices since (T, V_s', E_c') is the result of running the `Reposition` algorithm). Let $v_j \in V^*$ be the parent of T_i in G. Since initially each source vertex is adjacent to two or more dirty edges, and since T^* is the first tree from $\{T_1, T_2, ..., T_k\}$ to be cleared, there must be a searcher, say λ_*, in T_i which is protecting v_i from becoming adjacent with a dirty edge.

If λ_* is an additional searcher, then we associate λ_* with λ_j, and remove v_j and T_i from G. Otherwise, the searcher located in T_i is a starting searcher. By our assumption about S^* (i.e. during the progression of S^*, if a source vertex $v \in V_s$ is occupied, then it is occupied by λ_v, where λ_v is the starting searcher that occupies v in the instance (T, V_s', E_c')), this means that λ_*'s original source vertex v_* is currently unocuppied, and hence $v_* \in V(H)$. Since (T, V_s', E_c') is result of running the `Reposition` algorithm, the vertex v_* is adjacent to at least two trees T_k and T_l in H. Since T^* is the first tree from $\{T_1, T_2, ..., T_k\}$ to be cleared, there must be a searcher, say λ_k, in T_k. If λ_k is an additional searcher, then associate λ_k with λ_j, and we remove T_k and λ_* from H, and remove v_j and

T_i from G. If λ_k is not an additional searcher, then we can recursively consider the trees adjacent to v_k in H, and then consider the searcher that must be located in one of the subtrees incident to v_k. Since H is a finite tree, this process must eventually stop when an additional searcher is found to be located in a subtree incident with some source vertex in H. When this additional searcher is found, we remove the appropriate source vertices and subtrees from H, remove v_j and T_i from G, and associate this additional searcher with λ_j.

We repeat the procedure given above for each leaf in G until G is empty. This procedure associates a unique additional searcher with each starting searcher currently occupying a vertex in T^*, as required. □

It is clear that the search strategy output by the `ST-S` algorithm actually clears T. Thus we can use induction and Lemma 2 to obtain one of our main results:

Theorem 1. *The value of s_{max} at the termination of the* `ST-S` *algorithm is equal to $pss(T, V_s, E_c)$.*

Theorem 2. *Let (T, V_s, E_c) be an instance of the PSS problem with $|V(T)| = n$. The source search number and the optimal search strategy of (T, V_s, E_c) can be computed in $O(|V_s|n)$ time.*

Theorem 3. *The PSS problem is monotonic.*

3 Searching Trees with Targets

In this section we study the problem where instead of having only *sources,* we have only *targets.*

Target Searching (TS)
Instance: A tree T with a set of target vertices $V_t \subseteq V(T)$.
Question: Is there a search strategy of T using k additional searchers such that once a searcher occupies a vertex $v \in V_t$, v remains occupied by at least one searcher for the remainder of the search?

Associated with this problem, we define the target search number, denoted $ts(T, V_t)$, to be the minimum number of searchers required to clear the tree T under the TS model. The TS problem can be seen as a kind of inversion of the SS problem. Given a search strategy S, define the *reverse* of a action $a \in S$, denote a^{-1}, as follows:

- If a is "slide λ from v to u", then a^{-1} is "slide λ from u to v".
- If a is "remove λ from v", then a^{-1} is "place λ on v".
- If a is "place λ on v", then a^{-1} is "remove λ from v".

Given a strategy $S = (a_1, a_2, ..., a_k)$, define the *inverse* of S, denoted S^{-1}, to be $S^{-1} = (a_k^{-1}, a_{k-1}^{-1}, \ldots, a_1^{-1})$.

Lemma 3. *Let (T, V_s) be an instance of the SS problem and (T, V_t) be an instance of the TS problem such that $V_s = V_t$. Then S is a monotonic SS strategy for (T, V_s) if and only if S^{-1} is a monotonic TS strategy for (T, V_t).*

Proof. (sketch) Let $V' = V_s (= V_t)$. If S is a TS strategy for (T, V'), then it is clear that S^{-1} ends with V' occupied by searchers. Conversely, if S^{-1} is a TS strategy for (T, V'), then $(S^{-1})^{-1} = S$ begins with V' occupied by searchers. To complete the proof, we must show that S monotonically clears T if and only if S^{-1} monotonically clears T. Since S and S^{-1} are inverses, it suffices to demonstrate the following property: if S clears T monotonically then S^{-1} clears T monotonically. For any sequence of actions $(b_1, b_2, ..., b_l)$, define $A(b_1, b_2, \ldots, b_l)$ to be the set of edges cleared by a sequence of actions $(b_1, b_2, \ldots, b_l)$. We want to show that $A(a_k^{-1}, a_{k-1}^{-1}, \ldots, a_1^{-1}) = A(a_1, a_2, ..., a_k)$.

We proceed by induction on the number of actions completed by S^{-1}. Initially no edges are cleared in the TS model, and the last action of a TS model is a remove operation, and hence no edge is cleared by the last action in S. This provides the base case for induction.

Assume $A(a_k^{-1}, a_{k-1}^{-1}, \ldots, a_p^{-1}) = A(a_p, a_{p+1}, ..., a_k)$, for $p \leq k$. By carefully considering a_{p-1}^{-1} in S^{-1}, we can show that no recontamination occurs in S^{-1} unless it occurs in S, and the Lemma follows. □

In light of this observation, and Theorem 1, Theorem 3, and Theorem 2 from the previous section, we have the following two results:

Theorem 4. *Let (T, V_t) be an instance of the TS problem with $|V(T)| = n$. The target search number and the optimal search strategy of (T, V_t) can be computed in $O(|V_t|n)$ time.*

Theorem 5. *The TS problem is monotonic.*

4 Searching Trees with Sources and Targets

Now we study the STS problem. Recall that in this problem some searchers start on sources, and the search ends with some searchers occupying the targets. As in Section 2, we will actually study a slightly more general problem in order to make our analysis simpler:

Partial-Clear Source Target Searching (PSTS)
Instance: A tree T with a set of cleared edges $E_c \subseteq E(T)$, a set of source vertices $V_s \subseteq V(T)$, and a set of target vertices $V_t \subseteq V$, such that $V_t \cap V_s = \emptyset$, and each $v \in V_s$ is initially occupied by exactly one searcher.
Question: Is there a search strategy of T using k additional searchers such that once a searcher occupies a vertex $v \in V_t$, v remains occupied by at least one searcher for the remainder of the search, and each vertex from V_s remains clear during the entire search strategy?

We denote the search number associated with this searching problem as $psts(T, V_s, V_t, E_c)$. Our algorithm for the PSTS problem is a modified version of the algorithm presented in Section 2. The modifications take into account the addition of target vertices. Unlike the algorithm for the PSS problem, our algorithm for the PSTS problem does not necessarily first clear the subtree from $T \ominus V_s$ with the smallest search number. Instead, the algorithm first clears the subtree from $T \ominus V_s$ with the smallest search number *that does not contain a target vertex.* If all subtrees in $T \ominus V_s$ contain targets, then these subtrees are searched in increasing order by the number of target vertices they contain. Figure 3 illustrates why we must do this. In this example, $s(T_3) = 10$. If T_1 is cleared first then an additional searcher must be left behind to occupy v_1. To clear the remaining tree requires at least 10 additional searchers, resulting in a total of 11 additional searchers. The situation is the same if T_2 is cleared first. If T_3 is cleared first using 10 additional searchers, then T_1 can then be cleared using 2 of these additional searchers, leaving one behind on v_1, and then T_2 can be cleared using 2 more additional searchers, resulting in a total of 10 additional searchers.

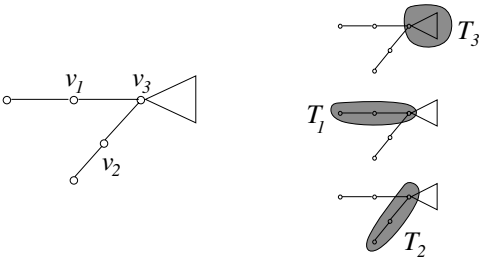


Fig. 3. A PSTS problem where $v_1, v_2 \in V_t$, $v_3 \in V_s$, and $s(T_3) = 10$

Now we present the new "reposition" algorithm that takes into account target vertices. A searcher λ occupying a vertex $v \in V_t$ is called *frozen* if it is the only searcher occupying v. The `Reposition-T` algorithm behaves the same as the `Reposition` algorithm from Section 2, except that frozen searchers are ignored (see Figure 4). We claim the following:

Lemma 4. *Let (T, V_s, V_t, E_c) be an instance of the PSTS problem. If V_s' and E_c' are results of running the* `Reposition-T` *algorithm, then $psts(T, V_s', V_t, E_c') \leq psts(T, V_s, V_t, E_c) + f$, where f is the number of starting searchers removed by the* `Reposition-T` *algorithm.*

Now we describe the main algorithm for computing $psts(T, V_s, V_t, E_c)$. The main difference between this algorithm and the one presented in Section 2 is that we must now take into account the target vertices. This is reflected in the choice that the algorithm makes with regard to the subtree that it chooses to clear first. Our algorithm is given in Figure 5.

Intuitively, the following Lemma shows that it is correct for the algorithm to clear T_1 first. In the following, T_{source} defined in step 4 of the `ST-ST` algorithm for the instance (T, V_s, V_t, E_c).

Algorithm Reposition-T(T, V_s, V_t, E_c) (Reposition with Targets)

1. While there exists a non-frozen moveable searcher λ on vertex $v \in V_s$:
   (a) Let vu be the only dirty edge incident with v.
   (b) Slide λ from v to u, clearing the edge vu, and $V_s \leftarrow (V_s - \{v\}) \cup \{u\}$, $E_c \leftarrow E_c \cup \{u\}$.
   (c) If u contains two searchers then remove λ and $f \leftarrow f + 1$.
2. While there exists a non-frozen searcher λ on a clear vertex $v \in V_s$:
   (a) Remove λ and $V_s \leftarrow V_s - \{v\}$.
   (b) $f \leftarrow f + 1$.

Fig. 4. The algorithm for repositioning the starting searchers when the tree contains targets

Algorithm ST-ST(T, V_s, V_t, E_c) (Search Tree with Sources and Targets)

1. If T is clear then return 0.
2. Call Reposition-T(T, V_s, V_t, E_c).
3. Compute $T \ominus V_s$, discarding all clear subtrees, to obtain $\{T_1, T_2, ..., T_k\}$.
4. Partition $T_1, T_2, \ldots, T_k$ into two sets T_{source} and T_{target} such that $T_i \in T_{target}$ if and only if T_i contains a target vertex. Sort T_{source} in increasing order by search number, and sort T_{target} in increasing order by the number of target vertices in the subtree. Furthermore, subtrees in T_{target} that have the same number of target vertices are sorted by their target search number.
5. If $T_{source} \neq \emptyset$ then,
   (a) Mark all edges in the first tree from T_{source}, say T_1, as clear. (explicitly clear this subtree if we are computing an actual search strategy, rather than just computing $psts(T, V_s, V_t, E_c)$.)
   (b) $s \leftarrow s(T_1)$ and go to step 7.
6. Otherwise, do the following:
   (a) Clear the first subtree from T_{target}, say T_1, using the algorithm from Section 3.
   (b) $s \leftarrow ts(T_1, V_t \cap V(T_1))$.
7. $s_{max} \leftarrow max\{s - f + |V_t \cap V(T_1)|,$ ST-ST$(T, V_s, V_t, E_c)\}$.
8. Return s_{max}.

Fig. 5. The algorithm for computing $psts(T, V_s, V_t, E_c)$

Lemma 5. *Let (T, V_s, V_t, E_c) be an instance of the PSTS problem. If $T_{source} \neq \emptyset$ then*
$psts(T, V_s, V_t, E_c) \geq s(T_1) - f$. Otherwise $psts(T, V_s, V_t, E_c) \geq ts(T_1) - f + |V_t \cap V(T_1)|$.

As in Section 2, we can use induction and Lemma 5, along with Lemma 2 and 4 to show the following:

Theorem 6. *Let (T, V_s, V_t, E_c) be an instance of the PSTS problem with $|V(T)| = n$. The source target search number and the optimal search strategy of (T, V_s, V_t, E_c) can be computed in $O(|V_s||V_t|n)$ time.*

An analysis of the actions generated by the `Reposition-T` algorithm and Theorem 5 implies the following:

Theorem 7. *The PSTS problem is monotonic.*

5 Conclusion and Future Work

We have introduced and studied a new kind of graph searching problem involving sources and targets. We showed that each search model studied is monotonic. We have provided an $O(|V_s|n)$ time algorithm for the problem of searching a tree with $|V_s|$ sources, an $O(|V_t|n)$ time algorithm for searching a tree with $|V_t|$ targets, and an $O(|V_s||V_t|n)$ time algorithm for searching a tree with $|V_s|$ sources and $|V_t|$ targets. We conjecture that algorithms exist for all these problems that run in $O(n \log n)$ time. We are interested in studying source and target searching problems on larger classes of graphs. We are also interested in relating the source- and target-searching parameters to other graph parameters such as vertex separation and treewidth.

References

1. Dendris, N., Kirousis, L., Thilikos, D.: Fugitive-search games on graphs and related parameters. Theoretical Computer Science 172, 233–254 (1997)
2. Fomin, F., Petrov, N.: Pursuit-evasion and search problems on graphs. Congressus Numerantium 122, 47–58 (1996)
3. LaPaugh, A.: Recontamination does not help to search a graph. Journal of ACM 40, 224–245 (1993)
4. Megiddo, N., Hakimi, S., Garey, M., Johnson, D., Papadimitriou, C.: The complexity of searching a graph. Journal of ACM 35, 18–44 (1998)
5. Parsons, T.: Pursuit-evasion in a graph. In: Theory and Applications of Graphs. Lecture Notes in Mathematics, pp. 426–441. Springer, Heidelberg (1976)
6. Peng, S., Ho, C., Hsu, T., Ko, M., Tang, C.: Edge and Node Searching Problems on Trees. Theoretical Computer Science 240, 429–446 (2000)
7. Yang, B., Zhang, R., Cao, Y.: Searching Cycle-Disjoint Graphs. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) COCOA 2007. LNCS, vol. 4616, pp. 32–43. Springer, Heidelberg (2007)

Ranking of Closeness Centrality for Large-Scale Social Networks

Kazuya Okamoto[1], Wei Chen[2], and Xiang-Yang Li[3]

[1] Kyoto University
okia@kuis.kyoto-u.ac.jp
[2] Microsoft Research Asia
weic@microsoft.com
[3] Illinois Institute of Technology and Microsoft Research Asia
xli@cs.iit.edu

Abstract. Closeness centrality is an important concept in social network analysis. In a graph representing a social network, closeness centrality measures how close a vertex is to all other vertices in the graph. In this paper, we combine existing methods on calculating exact values and approximate values of closeness centrality and present new algorithms to rank the top-k vertices with the highest closeness centrality. We show that under certain conditions, our algorithm is more efficient than the algorithm that calculates the closeness-centralities of all vertices.

1 Introduction

Social networks have been the subject of study for many decades in social science research. In recent years, with the rapid growth of Internet and World Wide Web, many large-scale online-based social networks such as Facebook, Friendster appear, and many large-scale social network data, such as coauthorship networks, become easily available online for analysis [Ne04a, Ne04b, EL05, PP02]. A social network is typically represented as a graph, with individual persons represented as vertices, the relationships between pairs of individuals as edges, and the strengths of the relationships represented as the weights on edges (for the purpose of finding the shortest weighted distance, we can treat lower-weight edges as stronger relationships). Centrality is an important concept in studying social networks [Fr79, NP03]. Conceptually, centality measures how central an individual is positioned in a social network. Within graph theory and network analysis, various measures (see [KL05] for details) of the centrality of a vertex within a graph have been proposed to determine the relative importance of a vertex within the graph. Four measures of centrality that are widely used in network analysis are *degree centrality*, *betweenness centrality*[1], *closeness centrality*,

[1] For a graph $G = (V, E)$, the betweenness centrality $C_B(v)$ for a vertex v is $C_B(v) = \sum_{s,t:s\neq t\neq v} \frac{\sigma_v(s,t)}{\sigma(s,t)}$ where $\sigma(s,t)$ is the number of shortest paths from s to t, and $\sigma_v(s,t)$ is the number of shortest paths from s to t that pass through v.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 186–195, 2008.

and *eigenvector centrality*[2]. In this paper, we focus on *shortest-path closeness centrality* (or closeness centrality for short) [Ba50, Be65]. The closeness centrality of a vertex in a graph is the inverse of the average shortest-path distance from the vertex to any other vertex in the graph. It can be viewed as the efficiency of each vertex (individual) in spreading information to all other vertices. The larger the closeness centrality of a vertex, the shorter the average distance from the vertex to any other vertex, and thus the better positioned the vertex is in spreading information to other vertices.

The closeness centrality of all vertices can be calculated by solving all-pairs shortest-paths problem, which can be solved by various algorithms taking $O(nm + n^2 \log n)$ time [Jo77, FT87], where n is the number of vertices and m is the number of edges of the graph. However, these algorithms are not efficient enough for large-scale social networks with millions or more vertices. In [EW04], Eppstein and Wang developed an approximation algorithm to calculate the closeness centrality in time $O(\frac{\log n}{\epsilon^2}(n \log n + m))$ within an additive error of $\epsilon\Delta$ for the inverse of the closeness centrality (with probability at least $1 - \frac{1}{n}$), where $\epsilon > 0$ and Δ is the diameter of the graph.

However, applications may be more interested in ranking vertices with high closeness centralities than the actual values of closeness centralities of all vertices. Suppose we want to use the approximation algorithm of [EW04] to rank the closeness centralities of all vertices. Since the average shortest-path distances are bounded above by Δ, the average difference in average distance (the inverse of closeness centrality) between the ith-ranked vertex and the $(i+1)$th-ranked vertex (for any $i = 1, \ldots, n-1$) is $O(\frac{\Delta}{n})$. To obtain a reasonable ranking result, we would like to control the additive error of each estimate of closeness centrality to within $O(\frac{\Delta}{n})$, which means we set ϵ to $\Theta(\frac{1}{n})$. Then the algorithm takes $O(n^2 \log n(n \log n + m))$ time, which is worse than the exact algorithm.

Therefore, we cannot use either purely the exact algorithm or purely the approximation algorithm to rank closeness centralities of vertices. In this paper, we show a method of ranking top k highest closeness centrality vertices, combining the approximation algorithm and the exact algorithm. We first provide a basic ranking algorithm TOPRANK(k), and show that under certain condi-

[2] Given a graph $G = (V, E)$ with adjacency matrix A, let x_i be the eigenvector centrality of the ith node v_i. Then vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$ is the solution of equation $A\mathbf{x} = \lambda\mathbf{x}$, where λ is the greatest eigenvalue of A to ensure that all values x_i are positive by the Perron-Frobenius theorem. Google's PageRank [BP98] is a variant of the eigenvector centrality measure. The PageRank vector $\mathbf{R} = (r_1, r_2, \cdots, r_n)^T$, where r_i is the PageRank of webpage i and n is the total number of webpages, is the solution of the equation

$$\mathbf{R} = \frac{1-d}{n} \cdot \mathbf{1} + d\mathbf{L}\mathbf{R}.$$

Here d is a damping factor set around 0.85, $\mathbf{L}$ is a modified webpage-adjacency matrix: $l_{i,j} = 0$ if page j does not link to i, and normalised such that, for each j, $\sum_{i=1}^{n} l_{i,j} = 1$, i.e., $l_{i,j} = \frac{a_{i,j}}{d_j}$ where $a_{i,j} = 1$ only if page j has link to page i, and $d_j = \sum_{i=1}^{n} a_{i,j}$ is the out-degree of page j.

tions, the algorithm ranks all top k highest closeness centrality vertices (with high probability) in $O((k+n^{\frac{2}{3}}\cdot\log^{\frac{1}{3}} n)(n\log n+m))$ time, which is better than $O(n(n\log n+m))$ (when $k=o(n)$), the time needed by a brute-force algorithm that simply computes all average shortest distances and then ranks them. We then use a heuristic to further improve the algorithm. Our work can be viewed as the first step toward designing and evaluating efficient algorithms in finding top ranking vertices with highest closeness centralities. We discuss in the end several open problems and future directions of this work.

2 Preliminary

We consider a connected weighted undirected graph $G=(V,E)$ with n vertices and m edges ($|V|=n, |E|=m$). We use $d(v,u)$ to denote the length of a shortest-path between v and u, and Δ to denote the diameter of graph G, i.e., $\Delta=\max_{v,u\in V} d(v,u)$. The *closeness centrality* c_v of vertex v [Be65] is defined as

$$c_v=\frac{n-1}{\Sigma_{u\in V} d(v,u)}. \tag{2.1}$$

In other words, the closeness centrality of v is the inverse of the average (shortest-path) distance from v to any other vertex in the graph. The higher the c_v, the shorter the average distance from v to other vertices, and v is more important by this measure. Other definitions of closeness centralities exist. For example, some define the closeness centrality of a vertex v as $\frac{1}{\Sigma_{u\in V} d(v,u)}$ [Sa66], and some define the closeness centrality as the mean geodesic distance (i.e the shortest path) between a vertex v and all other vertices reachable from it, i.e., $\frac{\Sigma_{u\in V} d(v,u)}{n-1}$, where $n\geq 2$ is the size of the network's connected component V reachable from v. In this paper, we will focus on closeness centrality defined in equation (2.1).

The problem to solve in this paper is to find the top k vertices with the highest closeness centralities and rank them, where k is a parameter of the algorithm. To solve this problem, we combine the exact algorithm [Jo77, FT87] and the approximation algorithm [EW04] for computing average shortest-path distances to rank vertices on the closeness centrality. The exact algorithm iterates Dijkstra's single-source shortest-paths (SSSP for short) algorithm n times for all n vertices to compute the average shortest-path distances. The original Dijkstra's SSSP algorithm [Di59] computes all shortest-path distances from one vertex, and it can be efficiently implemented in $O(n\log n+m)$ time [Jo77, FT87].

The approximation algorithm RAND given in [EW04] also uses Dijkstra's SSSP algorithm. RAND samples ℓ vertices uniformly at random and computes SSSP from each sample vertex. RAND estimates the closeness centrality of a vertex using the average of ℓ shortest-path distances from the vertex to the ℓ sample vertices instead of to all n vertices. The following bound on the accuracy of the approximation is given in [EW04], which utilizes the Hoeffding's theorem [Ho63]:

$$\Pr\{|\frac{1}{\hat{c}_v}-\frac{1}{c_v}|\geq\epsilon\Delta\}\leq\frac{2}{n^{2\ell\frac{\epsilon^2}{\log n}(\frac{n-1}{n})^2}}, \tag{2.2}$$

for any small positive value ϵ, where $\hat{c}_v$ is the estimated closeness centrality of vertex v. Let a_v be the average shortest-path distance of vetex v, i.e.,

$$a_v = \frac{\Sigma_{u \in V} d(v,u)}{n-1} = \frac{1}{c_v}.$$

Using the average distance, inequality (2.2) can be rewritten as

$$\Pr\{|\hat{a}_v - a_v| \geq \epsilon \Delta\} \leq \frac{2}{n^{2\ell \frac{\epsilon^2}{\log n}(\frac{n-1}{n})^2}}, \tag{2.3}$$

where $\hat{a}_v$ is the estimated average distance of vertex v to all other vertices. If the algorithm uses $\ell = \alpha \frac{\log n}{\epsilon^2}$ samples ($\alpha > 1$ is a constant number) which will cause the probability of $\epsilon\Delta$ error at each vertex to be bounded above by $\frac{1}{n^2}$, the probability of $\epsilon\Delta$ error anywhere in the graph is then bounded from above by $\frac{1}{n}$ ($\geq 1 - (1 - \frac{1}{n^2})^n$). It means that the approximation algorithm calculates the average lengths of shortest-paths of all vertices in $O(\frac{\log n}{\epsilon^2}(n \log n + m))$ time within an additive error of $\epsilon\Delta$ with probability at least $1 - \frac{1}{n}$, i.e., with high probability (w.h.p.).

3 Ranking Algorithms

Our top-k ranking algorithm is based on the approximation algorithm as well as the exact algorithm. The idea is to first use the approximation algorithm with ℓ samples to obtain estimated average distances of all vertices and find a candidate set E of top-k' vertices with estimated shortest distances. We need to guarantee that all final top-k vertices with the exact average shortest distances are included in set E with high probability. Thus, we need to carefully choose number $k' > k$ using the bound given in formula (2.3). Once we find set E, we can use the exact algorithm to compute the exact average distances for all vertices in E and rank them accordingly to find the final top-k vertices with the highest closeness centralities. The key of the algorithm is to find the right balance between sample size ℓ and the candidate set size k': If we use a too small sample size ℓ, the candidate set size k' could be too large, but if we try to make k' small, the sample size ℓ may be too large. Ideally, we want an optimal ℓ that minimizes $\ell + k'$, so that the total time of both the approximation algorithm and the computation of exact closeness centralities of vertices in the candidate set is minimized. In this section we will show the basic algorithm first, and then provide a further improvement of the algorithm with a heuristic.

3.1 Basic Ranking Algorithm

We name the vertices in V as $v_1, v_2, \ldots, v_n$ such that $a_{v_1} \leq a_{v_2} \leq \cdots \leq a_{v_n}$. Let $\hat{a}_v$ be the estimated average distance of vertex v using approximation algorithm based on sampling. Figure 1 shows our basic ranking algorithm TOPRANK(k), where k is the input parameter specifying the number of top ranking vertices

the algorithm should extract. The algorithm also has a configuration parameter ℓ, which is the number of samples used by the RAND algorithm in the first step. We will specify the value of ℓ in Lemma 2. Function $f(\ell)$ in step 4 is defined as follows: $f(\ell) = \alpha' \sqrt{\frac{\log n}{\ell}}$ (where $\alpha' > 1$ is a constant number), such that the probability of the estimation error for any vertex being at least $f(\ell) \cdot \Delta$ is bounded above by $\frac{1}{2n^2}$, based on inequality (2.3) (when setting $\epsilon = f(\ell)$).

Algorithm ***TOPRANK(k)***

1 Use the approximation algorithm RAND with a set S of ℓ sampled vertices to obtain the estimated average distance $\hat{a}_v$ for each vertex v.
// Rename all vertices to $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_n$ such that $\hat{a}_{\hat{v}_1} \leq \hat{a}_{\hat{v}_2} \leq \cdots \leq \hat{a}_{\hat{v}_n}$.
2 Find $\hat{v}_k$.
3 Let $\hat{\Delta} = 2 \min_{u \in S} \max_{v \in V} d(u, v)$.
// $d(u, v)$ for all $u \in S, v \in V$ have been calculated at step 1 and $\hat{\Delta}$ is determined in $O(\ell n)$ time.
4 Compute candidate set E as the set of vertices whose estimated average distances are less than or equal to $\hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \hat{\Delta}$.
5 Calculate exact average shortest-path distances of all vertices in E.
6 Sort the exact average distances and find the top-k vertices as the output.

Fig. 1. Algorithm for ranking top-k vertices with the highest closeness centralities

Lemma 1. *Algorithm TOPRANK(k) given in Figure 1 ranks the top-k vertices with the highest closeness centralities correctly w.h.p., with any configuration parameter ℓ.*

Proof. We show that the set E computed at step 4 in algorithm TOPRANK(k) contains all top-k vertices with the exact shortest distances w.h.p.

Let $T = \{v_1, \ldots, v_k\}$ and $\hat{T} = \{\hat{v}_1, \ldots, \hat{v}_k\}$. Since for any vertex v, the probability of the estimate $\hat{a}_v$ exceeding the error range of $f(\ell) \cdot \Delta$ is bounded above by $\frac{1}{2n^2}$, i.e., $\Pr\left(\neg\{a_v - f(\ell) \cdot \Delta \leq \hat{a}_v \leq a_v + f(\ell) \cdot \Delta\}\right) \leq \frac{1}{2n^2}$, we have

$$\Pr\left(\neg\{\bigwedge_{v \in T} \hat{a}_v \leq a_v + f(\ell) \cdot \Delta \leq a_{v_k} + f(\ell) \cdot \Delta\}\right) \leq \frac{k}{2n^2}; \text{ and}$$

$$\Pr\left(\neg\{\bigwedge_{\hat{v} \in \hat{T}} a_{\hat{v}} \leq \hat{a}_{\hat{v}} + f(\ell) \cdot \Delta \leq \hat{a}_{\hat{v}_k} + f(\ell) \cdot \Delta\}\right) \leq \frac{k}{2n^2}.$$

The latter inequality means that, with error probability of at most $\frac{k}{2n^2}$, there are at least k vertices whose real average distances are less than or equal to $\hat{a}_{\hat{v}_k} + f(\ell) \cdot \Delta$, which means $a_{v_k} \leq \hat{a}_{\hat{v}_k} + f(\ell) \cdot \Delta$ with error probability bounded above by $\frac{k}{2n^2}$. Then $\hat{a}_v \leq a_{v_k} + f(\ell) \cdot \Delta \leq \hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \Delta$ for all $v \in T$ with error probability bounded above by $\frac{k}{n^2}$. Moreover, we have $\Delta \leq \hat{\Delta}$, because for any $u \in S$, we have

$$\begin{aligned} \Delta &= \max_{v, v' \in V} d(v, v') \leq \max_{v, v' \in V} (d(u, v) + d(u, v')) \\ &= \max_{v, v' \in V} d(u, v) + \max_{v, v' \in V} d(u, v') = 2 \max_{v \in V} d(u, v). \end{aligned}$$

and thus

$$\Delta \leq 2 \min_{u \in S} \max_{v \in V} d(u,v) = \hat{\Delta}.$$

Therefore, for all $v \in T$, $\hat{a}_v \leq \hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \hat{\Delta}$ with probability at least $(1 - \frac{1}{n})$ (because $k \leq n$). Hence, TOPRANK(k) includes all top-k vertices with exact average distances in E in step 4, and TOPRANK(k) finds these exact k vertices in steps 5 and 6, with high probability. This finishes the proof of the lemma. □

We now evaluate the complexity of algorithm TOPRANK(k). The major computation tasks are ℓ computations of SSSP in step 1 and $|E|$ computations of SSSP in step 5. We need to choose an appropriate ℓ to minimize the sum of these computations. The number of computations of SSSP in step 5 depends on the distribution of estimated average distances of all vertices. The following lemma provides an answer when this distribution is uniform.

Lemma 2. *If the distribution of estimated average distances is uniform with range* $\mathrm{c}\Delta$ *(*c *is a constant number), then TOPRANK(k) takes* $O((k + n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n)(n \log n + m))$ *time, when we choose* $\ell = \Theta(n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n)$.

Proof. TOPRANK(k) takes $O(\ell(n \log n + m))$ time at step 1 because SSSP algorithm takes $O(n \log n + m)$ time and TOPRANK(k) iterates SSSP algorithm ℓ times.

Since the distribution of estimated average distances is uniform with range $\mathrm{c}\Delta$, there are $n \cdot \frac{2f(\ell) \cdot \hat{\Delta}}{\mathrm{c}\Delta}$ vertices between $\hat{a}_{\hat{v}_k}$ and $\hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \hat{\Delta}$, and $n \cdot \frac{2f(\ell) \cdot \hat{\Delta}}{\mathrm{c}\Delta}$ is $O(nf(\ell))$ because $\hat{\Delta} = 2 \min_{u \in S} \max_{v \in V} d(u,v) \leq 2 \max_{u,v \in V} d(u,v) = 2\Delta$. So, the number of vertices in E is $k + O(nf(\ell))$ and TOPRANK(k) takes $O((k + O(nf(\ell)))(n \log n + m))$ time at step 5.

Therefore, we select an ℓ that could minimize the total running time at step 1 and 5. In other words, we choose an ℓ to minimize $\ell + nf(\ell)$, which implies $\ell = \Theta(n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n)$. Then TOPRANK($k$) takes $O(n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n(n \log n + m))$ time at step 1, and takes $O((k + n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n)(n \log n + m))$ at step 5. Obviously TOPRANK(k) takes $O(n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n(n \log n + m))$ time at the other steps. So, TOPRANK(k) takes $O((k + n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n)(n \log n + m))$ total running time. □

Combining Lemmata 1 and 2, we arrive at the following theorem.

Theorem 1. *If the distribution of estimated average distances is uniform with range* $\mathrm{c}\Delta$ *(*c *is a constant number), then algorithm TOPRANK(k) given in Figure 7 ranks the top-k vertices with the highest closeness centralities in* $O((k + n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n)(n \log n + m))$ *time w.h.p., when we choose* $\ell = \Theta(n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n)$.

Theorem 1 only addresses the case when the distribution of estimated average distances is uniform. In this case, the complexity of TOPRANK(k) is better than a brute-force algorithm that simply computes all average shortest distances and ranks them, which takes $O(n(n \log n + m))$ time (assuming $k = o(n)$). Even

though the theorem is only for the case of uniform distribution, it could be applied to more general situations, as explained now. Given an estimated average distance x, its density $d(x)$ is the number of vertices whose estimate average distance is around x. The uniform distribution means that the density $d(x)$ is the same anywhere in the range of x. For any other distribution, it has an average density of d, which is the average of $d(x)$ over all x's. Suppose that the distribution is such that when x is sufficiently small, $d(x) \leq d$ (this property requires further investigation but we believe it is reasonable for social networks). Let x_0 be the largest value such that for all $x \leq x_0$, $d(x) \leq d$. Then, in our algorithm, as long as $\hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \hat{\Delta} \leq x_0$, the number of vertices between $\hat{a}_{\hat{v}_k}$ and $\hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \hat{\Delta}$ is at most $n \cdot \frac{2f(\ell) \cdot \hat{\Delta}}{c\Delta}$, as given in the proof of Lemma 2. Thus, Lemma 2 uses a conservative upper bound for this number, and it will still hold for the distributions with the above property.

Even with the above generalization, however, the savings from $O(n(n \log n + m))$ to $O((k + n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n)(n \log n + m))$ is still not very significant. Ideally, we would like a ranking algorithm that is $O(poly(k)(n \log n + m))$, where $poly(k)$ is a polynomial of k, which means the number of SSSP calculations is only related to k, not n. This is possible for small k when the distribution of estimated average distances is not uniform but other distributions like the normal distribution. In this case, the number of additional SSSP computations for vertices in the range from $\hat{a}_{\hat{v}_k}$ to $\hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \hat{\Delta}$ could be small and not related to n. We leave this as a future research work (see more discussion in Section 4).

3.2 Improving the Algorithm with a Heuristic

The algorithm in Figure 1 spends its majority of computation on the following two steps: (1) step 1 computing SSSP for ℓ samples, and (2) step 5 computing

---

Algorithm ***TOPRANK2(k)***

1 Use the approximation algorithm RAND with a set S of ℓ sampled vertices to obtain the estimated average distance $\hat{a}_v$ for each vertex v.
// Rename all vertices to $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_n$ such that $\hat{a}_{\hat{v}_1} \leq \hat{a}_{\hat{v}_2} \leq \cdots \leq \hat{a}_{\hat{v}_n}$.
2 Find $\hat{v}_k$.
3 Let $\hat{\Delta} = 2 \min_{u \in S} \max_{v \in V} d(u, v)$.
4 Compute candidate set E as the set of vertices whose estimated average distances are less than or equal to $\hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \hat{\Delta}$.
5 **repeat**
6 $p \leftarrow |E|$
7 Select additional q vertices S^+ as new samples uniformly at random.
8 Update estimated average distances of all vertices using new samples in S^+ (need to compute SSSP for all new sample vertices).
9 $S \leftarrow S \cup S^+; \ell \leftarrow \ell + q; \hat{\Delta} \leftarrow \min(\hat{\Delta}, 2 \min_{u \in S^+} \max_{v \in V} d(u, v))$
// Rename all vertices to $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_n$ such that $\hat{a}_{\hat{v}_1} \leq \hat{a}_{\hat{v}_2} \leq \cdots \leq \hat{a}_{\hat{v}_n}$.
10 Find $\hat{v}_k$.
11 Compute candidate set E as the set of vertices whose estimated average distances are less than or equal to $\hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \hat{\Delta}$.
12 $p' \leftarrow |E|$
13 **until** $p - p' \leq q$
14 Calculate exact average shortest-path distances of all vertices in E.
15 Sort the exact average distances and find the top-k vertices as the output.

Fig. 2. Improved algorithm for ranking vertices with top k closeness centralities

SSSP for all candidates in set E. The key in reducing the running time of the algorithm is to find the right sample size ℓ to minimize $\ell + |E|$, the total number of SSSP calculations. However, this number is difficult to obtain before running the algorithm, especially when the distribution of average distances is unknown. In this section, we improve the algorithm by a heuristic to achieve the above goal.

The idea of the heuristic is to incrementally add new samples to compute more accurate average distances of all vertices. In each iteration, q new sample vertices are added. After computing the new average distances with these q new vertices, we obtain a new candidate set E. If the size of the candidate set E decreases more than q, then we know that the savings by reducing the number of candidates outweigh the cost of adding more samples. In this case, we continue the next iteration of adding more samples. This procedure ends when the cost of adding more samples outweighs the savings obtained by the reduced number of candidates. Figure 2 provides this heuristic algorithm. Essentially, this is the dynamic way of finding the optimal ℓ to minimize $\ell + |E|$ (or to make $\Delta\ell = -\Delta|E|$, where $\Delta\ell$ is the small change in ℓ and $\Delta|E|$ is the corresponding change in $|E|$).

The initial value of the ℓ in step 1 can be obtained based on Theorem 1 if we know that the distribution of the estimated average distances is uniform. Otherwise, we can choose a basic value, for example k, since we need to compute at least k SSSP in step 14 in any case. The incremental unit q could be a small value, for example, $\log n$. However, we do not know yet if $|E|$ strictly decreases when the sample size ℓ for estimating average distances increases, and if the rate of decrease of $|E|$ slows down when adding more and more sample vertices. Therefore, it is not guaranteed that the heuristic algorithm will always stop at the optimal sample size ℓ. An open problem is to study the conditions under which the change of $|E|$ with respect to the change of sample size ℓ indeed has the above properties, and thus the heuristic algorithm indeed provides the most efficient solution.

4 Conclusion and Discussions

This paper can be viewed as the first step towards the design of more efficient algorithms in obtaining highest ranked closeness centrality vertices. By combining the approximation algorithm with the exact algorithm, we obtain an algorithm that has better complexity than the brute-force exact algorithm.

There are many directions to extend this study.

- First, as mentioned in the previous section, we are interested in more efficient algorithms such that the number of SSSP computations is only related to k, not to n. This may be possible for some classes of social networks with certain properties on their average distance distributions.
- Second, the condition under which the heuristic algorithm results in the least number of SSSP computation is an open problem and would be quite interesting to study.

- Third, we may be able to obtain faster algorithm if we can relax the problem requirement. Instead of finding all top-k vertices with high probability, we may allow the output to have an error bound t, which is the number of vertices that should be ranked within top-k vertices but are missed in the output. Generally, given an algorithm for top-k query of various centralities, we define the *hit-ratio*, denoted as $\eta(\mathcal{A})$, of an output $\mathcal{A} = \{v_1, v_2, \cdots, v_\kappa\}$ (not necessarily of size k) as $\frac{|\mathcal{A} \cap \mathcal{Q}_k|}{k}$, where $\mathcal{Q}_k$ is the actual set of top-k vertices. Let $r(v_i)$ denote the the actual rank of v_i. Here, we require that $r(v_i) < r(v_{i+1})$, for $i \in [1, \kappa - 1]$. Thus we have $r(v_i) \geq i$. We define the *accuracy* of $\mathcal{A}$, denoted as $\alpha(\mathcal{A})$, as $\frac{\kappa(\kappa+1)}{2\sum_{i=1}^{\kappa} r(v_i)}$ (other definitions of accuracy are possible). Clearly, $\eta(\mathcal{A}) \in [0, 1]$ and $\alpha(\mathcal{A}) \in [0, 1]$. The hit-ratio and accuracy of an algorithm are then its worst performance over all possible inputs. We then may only require that $\Pr(\eta(\mathcal{A}) \geq 1 - \varrho, \alpha(\mathcal{A}) \geq 1 - \epsilon) \geq 1 - \delta$, for sufficiently small ϱ, ϵ and δ. Such relaxations in the requirement may allow much more efficient algorithms, since we observe that the complexity of our algorithm is mainly because we need to include all vertices in the extra range from $\hat{a}_{\hat{v}_k}$ to $\hat{a}_{\hat{v}_k} + 2f(\ell) \cdot \hat{\Delta}$ in order to include all top-k vertices with high probability.
- Fourth, we would like to study the stability of our algorithms for top-k query using random sampling. Costenbader et al. [CV03] studied the stability of various centrality measures. Their study shows that, with a 50% sample of the original network nodes, average correlations for the closeness measure ranged from 0.54 to 0.71.
- Finally, we can look into other type of centralities and see how to rank them efficiently using the technique in this paper. For example, Brandes [Br00] presents an algorithm for betweenness centrality of weighted graph with time-complexity $n(m + n \log n)$. Brandes et al. [BP00] present the first approximation algorithm for betweenness centrality. Improvements over their method were presented recently in [GS08, BK07].

To conclude, we would like to provide a brief comparison of our algorithms with the well known PageRank algorithm [BP98]. PageRank is an algorithm used to rank the importance of the webpages, which are viewed as vertices connected by directed edges (hyperlinks). As explained in Footnote 2, PageRank is a variant of the eigenvector centrality. Thus, it is a different measure from closeness centrality. More importantly, by definition the PageRank of a vertex (a webpage) depends on the PageRanks of other vertices linking to it, so the PageRank calculation requires computing all PageRank values of all vertices, even if only the top-k PageRank vertices are desired. However, for closeness centrality measure, our algorithms do not need to compute closeness centralities for all vertices. Instead, we may start with rough estimates of closeness centralities of vertices, and through refining the estimates we reduce the candidate set containing the top-k vertices. This results in reduced time complexity in our computation.

References

[BK07] Bader, D.A., Kintali, S., Madduri, K., Mihail, M.: Approximating Betweenness Centrality. In: The 5th Workshop on Algorithms and Models for the Web-Graph, pp. 124–137 (2007)

[Ba50] Bavelas, A.: Communication patterns in task-oriented groups. The Journal of the Acoustical Society of America 22(6), 725–730 (1950)

[Be65] Beauchamp, M.A.: An improved index of centrality. Behavioral Science 10(2), 161–163 (1965)

[Br00] Brandes, U.: Faster Evaluation of Shortest-Path Based Centrality Indices. Konstanzer Schriften in Mathematik und Informatik 120 (2000)

[BP00] Brandes, U., Pich, C.: Centrality Estimation in Large Networks. Intl. Journal of Bifurcation and Chaos in Applied Sciences and Engineering 17(7), 2303–2318

[BP98] Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30, 107–117 (1998)

[CV03] Costenbader, E., Valente, T.W.: The stability of centrality measures when networks are sampled. Social Networks 25, 283–307 (2003)

[Di59] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)

[EL05] Elmacioglu, E., Lee, D.: On six degrees of separation in DBLP-DB and more. ACM SIGMOD Record 34(2), 33–40 (2005)

[EW04] Eppstein, D., Wang, J.: Fast Approximation of Centrality. Journal of Graph Algorithms and Applications 8, 39–45 (2004)

[FT87] Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34(3), 596–615 (1987)

[Fr79] Freeman, L.C.: Centrality in social networks conceptual clarification. Social Networks 1(3), 215–239 (1978/79)

[GS08] Geisberger, R., Sanders, P., Schultes, D.: Better Approximation of Betweenness Centrality. In: Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX 2008), pp. 90–100. SIAM, Philadelphia (2008)

[Ho63] Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the ACM 58(1), 13–30 (1963)

[Jo77] Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. Journal of the ACM 24(1), 1–13 (1977)

[KL05] Koschutzki, D., Lehmann, K.A., Peeters, L., Richter, S., Tenfelde-Podehl, D., Zlotowski, O.: Centrality indices. Network Analysis, 16–61 (2005)

[Ne04a] Newman, M.E.J.: Coauthorship networks and patterns of scientific collaboration. Proceedings of the National Academy of Sciences 101, 5200–5205 (2004)

[Ne04b] Newman, M.E.J.: Who is the best connected scientist? A study of scientific coauthorship networks. Complex Networks, 337–370 (2004)

[NP03] Newman, M.E.J., Park, J.: Why social networks are different from other types of networks. Physical Review E 68, 036122 (2003)

[PP02] Potterat, J.J., Phillips-Plummer, L., Muth, S.Q., Rothenberg, R.B., Woodhouse, D.E., Maldonado-Long, T.S., Zimmerman, H.P., Muth, J.B.: Risk network structure in the early epidemic phase of HIV transmission in Colorado Springs. Sexually Transmitted Infections 78, i159–i163 (2002)

[Sa66] Sabidussi, G.: The centrality index of a graph. Psychometrika 31(4), 581–603 (1966)

Mixed Search Number of Permutation Graphs

Pinar Heggernes and Rodica Mihai

Department of Informatics, University of Bergen, N-5020 Bergen, Norway
pinar@ii.uib.no, rodica@ii.uib.no

Abstract. Search games in graphs have attracted significant attention in recent years, and they have applications in securing computer networks against viruses and intruders. Since graph searching is an NP-hard problem, polynomial-time algorithms have been given for solving it on various graph classes. Most of these algorithms concern computing the node search number of a graph, and only few such algorithms are known for computing the mixed search or edge search numbers of specific graph classes. In this paper we show that the mixed search number of permutation graphs can be computed in linear time, and we describe an algorithm for this purpose. In addition, we give a complete characterization of the edge search number of complete bipartite graphs.

1 Introduction

The graph searching problem concerns a team of searchers who are trying to capture a fugitive moving along the edges of the graph. The fugitive is assumed to be very fast and invisible, and he knows the search strategy of the searchers. The minimum number of searchers that can guarantee the capture of the fugitive under this worst case scenario for the searchers is the search number of the graph, and the problem is to compute this number. The study of the graph searching problem started in 1970s when it was independently introduced by Parsons [25] and Petrov [28], and since that time it has been studied extensively [3,2,21,22,18,26]. It fits into the broader class of pursuit-evasion, search, and rendezvous problems on which a large number of results have appeared [1].

In a computer network setting, the graph searching problem serves as a mathematical model for protecting networks against viruses and other unwanted agents, like spyware or eavesdroppers [2,13]. A practical example is the problem of finding a successful strategy for a group of collaborating software programs that are designed to clean the network from a virus [11].

In the above mentioned original version of graph searching by Parsons and Petrov, later called edge searching [19], a search step consists of placing a searcher on a vertex or removing a searcher from a vertex or sliding a searcher along an edge. An edge is cleared by sliding a searcher from one of its endpoints to the other endpoint. Kirousis and Papadimitriou [19] introduced a variant of graph searching called node searching. In this version an edge is cleared if both its endpoints contain searchers. A new version of the graph searching was introduced by Bienstock and Seymour in [3]. This version, called mixed searching, combines

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 196–207, 2008.

features of both edge searching and node searching. An edge is cleared either by sliding or by placing searchers at each endpoint. In the mixed searching game, a contaminated edge of the graph is cleared if either both its two endpoints contain searchers or a searcher is slided along it. The allowable moves are placing a searcher on a vertex, removing a searcher from a vertex and sliding a searcher along an edge.

The minimum number of the searchers sufficient to perform searching and ensure the capture of the fugitive for each of the models are respectively the edge, node, and mixed search numbers, and computations of these are all NP-hard [3,22,18]. The node search number of a graph is known to be equal to its pathwidth plus one; the mixed search number of a graph is equal to its proper pathwidth [31].

Polynomial-time algorithms are known for computing the node search number of trees [27,29], interval graphs [7], cographs [6], k-starlike graphs for fixed k [26], d-trapezoid graphs [5], block graphs [9], split graphs [17], circular-arc graphs [30], and permutation graphs [4,23]. However, only for a few of these graph classes polynomial-time algorithms are known for computing mixed search or edge search numbers. Edge search number of trees [22,27], interval graphs and split graphs [26,15] can be computed in polynomial time. For computing the mixed search number, polynomial-time algorithms exist so far only for interval graphs and split graphs [12].

In this paper we show that the mixed search number of permutation graphs can be computed in linear time, thereby resolving the computational complexity of this problem on this graph class. Permutation graphs are a well-studied graph class with significant theoretical importance [16,8]. In addition, we show how to compute the edge search number for a subclass of permutation graphs, namely complete bipartite graphs. In fact we give a complete characterization of both edge and mixed search numbers on complete bipartite graphs.

2 Preliminaries

We work with simple and undirected graphs $G = (V, E)$, with vertex set $V(G) = V$ and edge set $E(G) = E$, and we let $n = |V|$, $m = |E|$. The set of *neighbors* of a vertex x is denoted by $N(x) = \{y \mid xy \in E\}$. A vertex set C is a *clique* if every two vertices in C are adjacent, and a *maximal clique* if no superset of C is a clique. The subgraph of G induced by a vertex set $A \subseteq V$ is denoted by $G[A]$.

A *path* is a sequence $v_1, v_2, ..., v_p$ of distinct vertices of G, where $v_i v_{i+1} \in E$ for $1 \leq i < p$, in which case we say that this is a path *between* v_1 and v_p. A path $v_1, v_2, ..., v_p$ is called a *cycle* if $v_1 v_p \in E$. A *chord* of a cycle (path) is an edge connecting two non-consecutive vertices of the cycle (path).

A vertex set $S \subset V$ is a *separator* if $G[V \setminus S]$ is disconnected. Given two vertices u and v, S is a u, v-*separator* if u and v belong to different connected components of $G[V \setminus S]$, and S is then said to *separate* u and v. Two separators S and T are said to be *crossing* if S is a u, v-separator for a pair of vertices $u, v \in T$, in which case T is an x, y-separator for a pair of vertices $x, y \in S$

[20,24]. A u, v-separator S is *minimal* if no proper subset of S separates u and v. In general, S is a *minimal separator* of G if there exist two vertices u and v in G such that S is a minimal u, v-separator. It can be easily verified that S is a minimal separator if and only if $G[V \setminus S]$ has two distinct connected components C_1 and C_2 such that $N_G(C_1) = N_G(C_2) = S$. In this case, C_1 and C_2 are called *full components*.

2.1 Chordal Graphs, Interval Graphs, and Pathwidth

Permutation graphs and complete bipartite graphs will be introduced in the sections in which they are studied. In this subsection we mention the graph classes and graph parameters that are central in graph searching.

A graph is *chordal* if every cycle of length at least 4 has a chord. A *triangulation* of a graph G is a chordal graph H on the same vertex set as G such that G is a subgraph of H. If there is no proper subgraph of H that is a triangulation of G then H is said to be a *minimal triangulation* of G. The following central characterization of minimal triangulations is useful for understanding our results on permutation graphs. A triangulation of G is minimal if and only if it is obtained by adding edges to make into cliques a maximal set of non-crossing minimal separators of G [24]. A set C of vertices of G is a *potential maximal clique* if there is a minimal triangulation of G in which C is a maximal clique.

A *path-decomposition* of a graph $G = (V, E)$ is a linearly ordered sequence of subsets of V, called *bags*, such that the following three conditions are satisfied: 1. Every vertex $x \in V$ appears in some bag. 2. For every edge $xy \in E$ there is a bag containing both x and y. 3. For every vertex $x \in V$, the bags containing x appear consecutively. The *width* of a decomposition is the size of the largest bag minus one, and the *pathwidth* of a graph G, $pw(G)$, is the minimum width over all possible path decompositions. A path decomposition of width $pw(G)$ is called an *optimal* path decomposition of G.

A graph is an *interval graph* if intervals of the real line can be associated to its vertices such that two vertices are adjacent if and only if their corresponding intervals overlap. An important characterization of interval graphs is that a graph G is an interval graph if and only if it has an optimal path decomposition where every bag is a maximal clique of G [14]. Such an optimal path decomposition is called a *clique-path*. It is well known that the pathwidth of an interval graph is one less than the size of its largest clique. Clique-paths of interval graphs can be computed in linear time [7]. For an arbitrary graph G, every path decomposition of G corresponds to an interval graph obtained by adding edges to G until each bag of the path decomposition is a clique. The mentioned path decomposition is then a clique path of this interval graph.

2.2 Search Games

The *mixed search game* can be formally defined as follows. Let $G = (V, E)$ be a graph to be searched. A *search program* consists of a sequence of discrete steps which involves searchers. Initially there is no searcher on the graph. Every step is one of the following three types

– Some searchers are placed on some vertices of G (there can be several searchers located in one vertex);
– Some searchers are removed from G;
– A searcher slides from a vertex u to a vertex v along edge uv.

At every step of the search program the edge set of G is partitioned into two sets: *cleared* and *contaminated* edges. Intuitively, the agile and omniscient fugitive with unbounded speed who is invisible for the searchers, is located somewhere on a contaminated territory, and cannot be on cleared edges. Initially all edges of G are contaminated, i.e., the fugitive can be anywhere. A contaminated edge uv becomes cleared at some step of the search program either if both its endpoints contain searchers, or if at this step a searcher located in u slides to v along uv.

A cleared edge e is (re)contaminated at some step if at this step there exists a path P containing e and a contaminated edge and no internal vertex of P contains a searcher. For example, if a vertex u is incident to a contaminated edge e, there is only one searcher at u and this searcher slides from u to v along edge $uv \neq e$, then after this step the edge uv, which is cleared by sliding, is immediately recontaminated.

A search program is *winning* if after its termination all edges are cleared. The *mixed search number* of a graph G, denoted by $ms(G)$, is the minimum number of searchers required for a winning program of mixed searching on G. The differences between mixed, edge, and node searching are in the way the edges can be cleared. In node searching an edge is cleared only if both its endpoints are occupied (no clearing by sliding). In edge searching an edge can be cleared only by sliding. So mixed searching can be seen as a combination of node and edge searching. The *edge* and *node search numbers* of a graph G are defined similarly to the mixed search number, and are denoted by $es(G)$ and $ns(G)$, respectively. A winning mixed search program using $ms(G)$ steps (analogously, a winning edge search program using $es(G)$ steps) is called *optimal*. The following result is central and gives the relation between the three graph searching parameters.

Lemma 1 ([31]). *Let G be an arbitrary graph.*

– $ns(G) = pw(G) + 1$.
– $pw(G) \leq ms(G) \leq pw(G) + 1$.
– $pw(G) \leq es(G) \leq pw(G) + 2$.

Note that, although the node search number of a graph is known, it might be difficult to decide its mixed search number or edge search number. Hence although $pw(G)$ of a graph G can be computed easily, it might be difficult to decide whether $ms(G) = pw(G)$ or $ms(G) = pw(G) + 1$.

A search program is called *monotone* if at any step of this program no recontamination occurs. For all three versions of graph searching, recontamination does not help to search the graph with fewer searchers [3,21], i.e., on any graph with {edge, mixed, node} search number k there exists a winning monotone {edge, mixed, node} search program using k searchers. Thus in this paper we consider only monotone search programs.

3 Mixed Search Number of Permutation Graphs

Let π be a permutation of $\{1, ..., n\}$. We define $G(\pi)$ to be the graph with vertex set $\{1, ..., n\}$ and edge set $\{ij \mid (i-j)\cdot(\pi^{-1}(i)-\pi^{-1}(j)) < 0\}$. Hence, two vertices i, j of $G(\pi)$ are adjacent if and only if the permutation π changes their natural order. An undirected graph G is called a *permutation graph* if there exists a permutation π such that G is isomorphic to $G(\pi)$.

In this section we give a linear-time algorithm to compute the mixed search number of permutation graphs. Permutation graphs are a well studied graph class with subject to many theoretical results, and they have many characterizations [16]. If G is a permutation graph then all minimal triangulations of G are interval graphs [4]. In addition, permutation graphs have a linear number of minimal separators [23]. Pathwidth of permutation graphs, and hence their node search number, can be computed in linear time [4,23]. No polynomial-time algorithm has been known for computing their mixed search number.

We start by relating mixed search number to proper pathwidth, and then giving a new general result, before we move to permutation graphs. A path decomposition is called *proper* if no three bags of the same size s all intersect in the same $s-1$ vertices. The *proper pathwidth* of a graph G, denoted by $ppw(G)$ is the minimum width over all proper path decompositions of G.

Theorem 1 ([31]). *For any graph G, $ms(G) = ppw(G)$.*

Thus computing the mixed search number and the proper pathwidth are equivalent problems. This, in combination with the following result, is the main tool that we use to compute the mixed search number of permutation graphs.

Theorem 2 ([12]). *For an interval graph G, $ms(G) = pw(G)$ if and only if no three maximum cliques intersect in $pw(G)$ vertices.*

We define a *good path decomposition* to be an optimal path decomposition that does not contain three consecutive bags intersecting in the same $pw(G)$ vertices. We now add the following new result for general graphs that strengthens Theorem 1.

Theorem 3. *For any graph G, $ms(G) = pw(G)$ if and only if G has a good path decomposition. (Otherwise $ms(G) = pw(G) + 1$.)*

Proof. First we show that in any optimal path decomposition P, if there are three bags of maximum size intersecting in the same $pw(G)$ vertices then there are three consecutive bags intersecting in the same $pw(G)$ vertices. Let B_i, B_j, B_k be three bags from left to right (not necessarily consecutive) in P such that $|S = B_i \cap B_j \cap B_k| = pw(G)$. Then by the definition of a path decomposition, S is a subset of every bag of P between B_i and B_k. Since no bag is a subset of another bag, it means that all bags between B_i and B_k must be of maximum size and contain S. Hence any three consecutive bags between B_i and B_k are of size $pw(G) + 1$ and contain the same $pw(G)$ vertices.

If $ms(G) = pw(G)$ then by Theorem 1, $pw(G) = ppw(G)$, and by the above argument there exists a good path decomposition of G.

If G has a good path decomposition P then let H be the interval graph obtained by making each bag of P into a clique by adding edges. Since P is an optimal path decomposition of G, $pw(H) = pw(G)$. Since G is a subgraph of H, $ms(G) \leq ms(H)$. No three bags of P of maximum size overlap in the same $pw(G)$ vertices, and since there is a one-to-one correspondence between the bags of P and the maximal cliques of H, no three maximum cliques of H overlap on the same $pw(G)$ vertices. Hence $ms(H) = pw(H)$ by Theorem 2. Combining all of the above, we obtain that $ms(G) \leq pw(G)$, and the result follows from Lemma 1. □

With this general result, we are now ready to move to permutation graphs, and the computation of their mixed search number.

Lemma 2. *Let G be a permutation graph. If $ms(G) = pw(G)$ then G has a good path decomposition that corresponds to a minimal triangulation of G.*

Proof. Since $ms(G) = pw(G)$, by Theorem 3 G has a good path decomposition P. Assume that the interval graph H that has P as a clique path is not a minimal triangulation of G. Then H has a chordal subgraph H' which is a minimal triangulation of G. Since all minimal triangulations of permutation graphs are interval graphs, H' is an interval graph and has a clique path P' which is a path decomposition of G. We argue that P' is a good path decomposition of G. Observe that the size of the largest bag in P' cannot be larger than the size of the largest bag in P since H' is a subgraph of H, and thus P' is an optimal path decomposition. Removal of edges from H might create three new bags of size s that intersect at the same $s-1$ vertices. However, if this happens then s cannot be equal to $pw(G)+1$ because if the removal of an edge splits a maximal clique into two new maximal cliques, then the new maximal cliques will be of size at least 1 less than the size of the old maximal clique. Hence, P' is a good path decomposition of G that corresponds to the minimal triangulation H', and the proof is complete. □

In the remaining of this section, let $G = G(\pi)$ be a permutation graph for a permutation π of $\{1, ..., n\}$. A *permutation diagram* of G is obtained in the following way. Take two copies of the real line between 0.5 and $n + 0.5$; place one of them below the other; put consecutive labels from 1 to n on the integer points of the above one; put consecutive labels from $\pi(1)$ to $\pi(n)$ on the integer points of the other; draw lines between a point on the upper line and a point on the lower line if and only if the two points have the same labels. Each line $(i, \pi^{-1}(i))$ will be called the *line of vertex i*. It is easy to see that two vertices i and j are adjacent in G if and only if their lines intersect in the permutation diagram for G.

A *scanline* of G is a pair (a, e) where $a, e \in \{0.5, 1.5, ..., n + 0.5\}$. We also define the following two scanlines $s_0 = (0.5, 0.5)$ and $s_e^n = (n+0.5, n+0.5)$. Each scanline s_i is associated with a set of vertices S_i of G such that S_i consists of exactly those vertices whose lines cross s_i. For each scanline s_i, the corresponding

vertex set S_i is a separator of G [4]. A *special scanline* is a scanline s_i such that S_i is a minimal separator of G and s_i is between two full components of $G[V \setminus S_i]$ in the permutation diagram [23].

Meister defined the potential maximal clique graph of a permutation graph, and used this to compute the pathwidth of permutation graphs in linear time [23]. We will heavily rely on this algorithm. The *potential maximal clique graph* of a permutation graph $G(\pi)$ is a directed graph $\mathcal{PC}(\pi)$ defined as follows: $\mathcal{PC}(\pi)$ has a vertex for every special scanline of G, and there is an arc from vertex s_i to vertex s_j if and only if the corresponding special scanline s_i is on the left side of the special scanline s_j and there is no other special scanline strictly between them (non-intersecting with any of them). Hence the set of vertices of $\mathcal{PC}(\pi)$ correspond exactly to the set of minimal separators of $G(\pi)$, and arcs go between non-crossing minimal separators. For each arch $s_i s_j$ of $\mathcal{PC}(\pi)$ we define C_{ij} to be the set of vertices whose lines cross s_i or s_j and vertices whose lines are between s_i and s_j in the permutation diagram of G. For each arc $s_i s_j$ in $\mathcal{PC}(\pi)$, C_{ij} is a potential maximal clique of G, and each potential maximal clique of G corresponds to an edge in $\mathcal{PC}(\pi)$. Furthermore, the number of vertices of $\mathcal{PC}(\pi)$ is $O(n+m)$, $\mathcal{PC}(\pi)$ is acyclic and can be generated in linear time [23]. A *source-sink path* in $\mathcal{PC}(\pi)$ is any directed path from s_0 to s_e^n.

Theorem 4 ([23]). *Let $G = G(\pi)$ be a permutation graph. An interval graph H is a minimal triangulation of G if and only if there is a source-sink path $s_0, ..., s_{k-1}, s_e^n$ in $\mathcal{PC}(\pi)$ such that $C_{01}, ..., C_{k-1,k}$ is a clique path of H.*

We will call a source-sink path $s_0, ..., s_{k-1}, s_e^n$ a *good path* if $|C_{i-1,i}| \leq pw(G)+1$, and $S_{i-1} \neq S_i$ whenever $|S_{i-1}| = |S_i| = pw(G)$, for all $i \in \{1, 2, \ldots, k\}$. We can now give the following main structural result.

Theorem 5. *Let $G = G(\pi)$ be a permutation graph. Then $ms(G) = pw(G)$ if and only if there exists a good path in $\mathcal{PC}(\pi)$. (Otherwise $ms(G) = pw(G) + 1$.)*

Proof. If there exists a good path in $\mathcal{PC}(\pi)$ then by Theorem 4 there exists a good path decomposition of G. Hence by Theorem 3, $ms(G) = pw(G)$. If $ms(G) = pw(G)$ then by Lemma 2 there exists a good path decomposition P of G that corresponds to a minimal triangulation. Observe that the condition that no three consecutive bags of P intersect in the same $pw(G)$ vertices is equivalent to the condition that no two consecutive minimal separators of size $pw(G)$ of P are equal. Hence by Theorem 4, there exists a good path in $\mathcal{PC}(\pi)$ corresponding to this minimal triangulation. □

Thus we will search for good paths in $\mathcal{PC}(\pi)$ to decide whether $ms(G) = pw(G)$ or $ms(G) = pw(G) + 1$. The algorithm is described in the proof of the following theorem.

Theorem 6. *The mixed search number of a permutation graph can be computed in linear time.*

Proof. We describe such an algorithm. By the results of [23], $\mathcal{PC}(\pi)$ can be computed, a topological search on it can be performed, and $pw(G)$ can be computed

in linear time. In fact, the algorithm of Meister [23] computes the size of each minimal separator corresponding to a vertex and each potential maximal clique corresponding to an arc of $\mathcal{PC}(\pi)$ within the linear time bound, and the data structure on which this algorithm is based allows checking the equivalence of two minimal separators in constant time. During the construction of $\mathcal{PC}(\pi)$, minimal separators that are equal are detected and only one copy of the list of vertices in these minimal separators is kept, whereas all minimal separators that contain exactly these vertices are set to point to the same location, all within the linear time bound. Hence to check whether two vertices of $\mathcal{PC}(\pi)$ correspond to two minimal separators that are equal, can be done in constant time by comparing the pointers, after the preprocessing during the construction of $\mathcal{PC}(\pi)$.

We now describe our algorithm, based on the above. First we run the algorithm of [23] to construct $\mathcal{PC}(\pi)$ as described above and to compute $pw(G)$. After this, we delete all arcs corresponding to potential maximal cliques of size larger than $pw(G)+1$, and all vertices corresponding to minimal separators of size larger than $pw(G)$ from $\mathcal{PC}(\pi)$, since these can never be involved in paths corresponding to optimal path decompositions. After this we traverse the graph in a topological order, and delete every arc s_is_j such that $|S_i| = |S_j| = pw(G)$ and $S_i = S_j$. Such arcs can never be part of a good path, and can thus be deleted safely. Let us call $\mathcal{PC}'(\pi)$ the potential maximal cliques graph that we obtain after the described deletions from $\mathcal{PC}(\pi)$. We claim that there is a good path in $\mathcal{PC}(\pi)$ if and only if there is a source-sink path in $\mathcal{PC}'(\pi)$. Clearly, if there is a source-sink path P in $\mathcal{PC}'(\pi)$ then P is a good path in $\mathcal{PC}(\pi)$ since P is a path in $\mathcal{PC}(\pi)$ that does not contain any of the forbidden substructures of a good path. If there is a good path P in $\mathcal{PC}(\pi)$ then this path survives all the deletions described above since it does not contain any of the deleted substructures. Hence P is a source-sink path in $\mathcal{PC}'(\pi)$.

The algorithm, after the deletions, is to simply search for a source-sink path in $\mathcal{PC}'(\pi)$. The algorithm returns such a path if it is found and outputs $ms(G) = pw(G)$. If no such path is found, then the algorithm returns $ms(G) = pw(G)+1$. The correctness of the algorithm follows from the proof of the claim in the previous paragraph and Theorem 5.

For the running time, after computing $\mathcal{PC}(\pi)$ and $pw(G)$ in linear time with Meister's algorithm [23], deletion of arcs and vertices that correspond to too big potential maximal cliques and minimal separators can be done in linear time, too, since we only check sizes. For each arc s_is_j, whether $|S_i| = |S_j| = pw(G)$ can be checked in constant time by the same arguments, and checking whether $S_i = S_j$ can be done in constant time, too, as explained in the first paragraph, comparing the pointers from s_i and s_j. Looking for source-sink paths in $\mathcal{PC}'(\pi)$ takes also clearly linear-time, since this is just simple graph traversal. Hence the total running time is $O(n+m)$. □

4 Edge Search Number of Complete Bipartite Graphs

A *bipartite graph* is a graph whose vertex set can be partitioned into two independent sets. We denote such a graph by $G = (A, B, E)$ where $A \cup B$ is the vertex

set of G, and A and B are independent sets. If G is a connected bipartite graph, then the partition of the vertex set into the two independent sets is unique. A bipartite graph $G = (A, B, E)$ is a *complete bipartite graph* if every vertex of A is adjacent to every vertex of B. Such a graph is denoted by $K_{a,b}$, where $a = |A|$ and $b = |b|$.

It is known that $pw(K_{a,b}) = \min\{a, b\}$ [6], hence the node search number of complete bipartite graphs is completely characterized. By the results of the previous section, their mixed search number can be computed in linear time, since complete bipartite graphs are a subset of permutation graphs. Here, we give a complete characterization of their edge search number, hence completing the knowledge of searching in complete bipartite graphs.

Lemma 3. *If* $\min\{a, b\} \geq 3$ *then* $es(K_{a,b}) = \min\{a, b\} + 2$.

Proof. Let A and B be the two independent sets of $G = K_{a,b}$ with $|A| = a$ and $|B| = b$. Assume without loss of generality that $a \leq b$. By the result mentioned above, $pw(G) = a$. Thus by Lemma 1 we have $a \leq es(G) \leq a + 2$. We will show that $es(G) = a + 2$. Hence we have to show that $es(G) \geq a + 2$.

For this lower bound, assume for a contradiction that there is an edge search program that clears the graph with $a + 1$ searchers without recontamination. If all vertices of A are occupied by searchers initially, then one searcher is left to clear all edges, and since all vertices of B are uncleared and without searchers, there is no way to continue without recontamination. Hence initially at least one vertex v of B is occupied with a searcher. This searcher can only be removed after the clearance of all edges incident to B but one, so the first move cannot be to slide the searcher on v. The same is true for the vertices of A as well, so at most $a - 1$ vertices of A are occupied with searchers initially, and the first move must be to use the last searcher to clear an edge whose both endpoints are occupied by searchers. When all edges incident to v are cleared except one, the searcher on v can be slided to the other endpoint of this remaining edge, and all edges incident to v will be cleared. This can be done only if at most one vertex of A was without searcher. After this, all vertices of A are occupied with searchers, we have one idle searcher, and at least two vertices of B remain without searchers. Hence each vertex of A has at least two uncleared edges incident to it. We can slide the idle searcher from a vertex u of A to a vertex of B different from v and leave it there, and if $b = 3$ we can even slide the searcher on u to the last vertex of B and leave it there, without recontamination. These are the only possible allowed moves at this stage. But after that still we are left with at least two vertices from each side that each have at least two uncleared edges incident to it, and we have no idle searcher available to slide between them. Since none of the searchers can be moved without recontamination, we obtain the desired contradiction, and conclude that the search cannot be completed with at most $a + 1$ searchers. □

For the cases not covered by the above lemma, it can be easily verified that $es(K_{1,1}) = es(K_{1,2}) = 1$, $es(K_{1,b}) = 2$ for $b \geq 3$, $es(K_{2,2}) = 2$, and $es(K_{2,b}) = 3$ for $b \geq 3$.

Hence we can conclude that if a complete bipartite graph is given as a pair of integers, representing the sizes of the two independent sets, its edge search number can be computed in constant time. This is also true for its node search number by the results of [6], and its mixed search number by our next result, which we include for completeness.

Lemma 4. *If* $\max\{a, b\} \geq 3$ *then* $ms(K_{a,b}) = \min\{a, b\} + 1$.

Proof. Let A and B be the two independent sets of $G = K_{a,b}$ with $|A| = a$ and $|B| = b$. Assume without loss of generality that $a \leq b$. By the result mentioned above, $pw(G) = a$. Thus by Lemma 1 we have $a \leq ms(G) \leq a+1$. We will show that $ms(G) = a + 1$. Hence we have to show that $ms(G) \geq a + 1$.

Assume for a contradiction that there is a mixed search program to clear the graph using a searchers without allowing recontamination. If initially all searchers are placed on the vertices of A, each of the vertices is adjacent to at least two uncleared edges. Therefore none of the searchers can be removed or slided without allowing recontamination. Hence initially at least one searcher is placed on a vertex v of B so that at most $a-1$ are placed on the vertices of A. Let $u \in A$ be a vertex without a searcher. All the edges between v and the vertices of $A \setminus \{u\}$ are thus cleared. Note that there are at least two uncleared vertices in B without searchers. (If $a = b$ and all searchers are placed on B the same argument above on A applies on B.) Hence the next step of this search program cannot be to move a searcher from a vertex of A to a vertex of B, because this would recontaminate that vertex of A since every vertex of A is adjacent to at least two uncleared vertices in B. Hence the next move is to move the searcher on v. If it is moved to another vertex of B, v will be recontaminated because of u, which is still uncleared. So to avoid recontamination, the search has to continue by sliding the searcher on v to u along the edge vu. At this moment v and all edges incident to it are cleared, all vertices of A are occupied by searchers, but each vertex of A is adjacent to at least two uncleared edges. Therefore no searcher can be slided or removed without allowing recontamination, which contradicts the existence of the assumed search program. Thus, $ms(G) \geq a+1$. □

For the cases not covered by this lemma, it can be easily verified that $ms(K_{1,1}) = ms(K_{1,2}) = 1$ and $ms(K_{2,2}) = 2$.

5 Concluding Remarks

We have shown that the mixed search number of permutation graphs can be computed in linear time. An interesting further research direction is to study the edge search number of permutation graphs. A result in this direction would complete the knowledge about all three graph searching parameters on this graph class.

References

1. Alpern, S., Gal, S.: The theory of search games and rendezvous. In: International Series in Operations Research & Management Science, vol. 55. Kluwer Academic Publishers, Boston (2003)
2. Bienstock, D.: Graph searching, path-width, tree-width and related problems (a survey). In: DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science, vol. 5, pp. 33–49 (1991)
3. Bienstock, D., Seymour, P.: Monotonicity in graph searching. J. Algorithms 12, 239–245 (1991)
4. Bodlaender, H.L., Kloks, T., Kratsch, D.: Treewidth and pathwidth of permutation graphs. SIAM J. Disc. Math. 8, 606–616 (1995)
5. Bodlaender, H.L., Kloks, T., Kratsch, D., Möhring, R.H.: Treewidth and Minimum Fill-in on d-Trapezoid Graphs. J. Graph Algorithms Appl. 2 (1998)
6. Bodlaender, H.L., Möhring, R.H.: The pathwidth and treewidth of cographs. In: Gilbert, J.R., Karlsson, R. (eds.) SWAT 1990. LNCS, vol. 447, pp. 301–310. Springer, Heidelberg (1990)
7. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using *pq*-tree algorithms. J. Comp. Syst. Sc. 13, 335–379 (1976)
8. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph classes: a survey, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1999)
9. Chou, H., Ko, M., Ho, C., Chen, G.: Node-searching problem on block graphs. Disc. Appl. Math. 156, 55–75 (2008)
10. Corneil, D.G., Lerchs, H., Stewart Burlingham, L.: Complement reducible graphs. Annals Discrete Math 1, 145–162 (1981)
11. Flocchini, P., Huang, M.J., Luccio, F.L.: Contiguous search in the hypercube for capturing an intruder. In: Proceedings of IPDPS 2005. IEEE Computer Society Press, Los Alamitos (2005)
12. Fomin, F., Heggernes, P., Mihai, R.: Mixed search number and linear-width of interval and split graphs. In: Proceedings of WG 2007. LNCS, vol. 4769, pp. 304–315 (2007)
13. Franklin, M., Galil, Z., Yung, M.: Eavesdropping games: a graph-theoretic approach to privacy in distributed systems. J. ACM 47, 225–243 (2000)
14. Gilmore, P.C., Hoffman, A.J.: A characterization of comparability graphs and of interval graphs. Canad. J. Math. 16, 539–548 (1964)
15. Golovach, P.A., Petrov, N.N.: Some generalizations of the problem on the search number of a graph. Vestn. St. Petersbg. Univ., Math. 28(3), 18–22 (1995); translation from Vestn. St-Peterbg. Univ., Ser. I, Mat. Mekh. Astron. 3, 21–27 (1995)
16. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Mathematics, vol. 57. North-Holland, Amsterdam (2004)
17. Gustedt, J.: On the pathwidth of chordal graphs. Disc. Appl. Math. 45, 233–248 (1993)
18. Kirousis, L.M., Papadimitriou, C.H.: Interval graphs and searching. Disc. Math. 55, 181–184 (1985)
19. Kirousis, M., Papadimitriou, C.H.: Searching and pebbling. Theor. Comput. Sci. 47, 205–218 (1986)
20. Kloks, T., Kratsch, D., Spinrad, J.: On treewidth and minimum fill-in of asteroidal triple-free graphs. Theor. Comp. Sc. 175, 309–335 (1997)

21. LaPaugh, A.S.: Recontamination does not help to search a graph. J. ACM 40, 224–245 (1993)
22. Megiddo, N., Hakimi, S.L., Garey, M.R., Johnson, D.S., Papadimitriou, C.H.: The complexity of searching a graph. J. ACM 35, 18–44 (1988)
23. Meister, D.: Computing Treewidth and Minimum Fill-In for Permutation Graphs in Linear Time. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 91–102. Springer, Heidelberg (2005)
24. Parra, A., Scheffler, P.: Characterizations and algorithmic applications of chordal graph embeddings. Disc. Appl. Math. 79, 171–188 (1997)
25. Parsons, T.: Pursuit-evasion in a graph. In: Theory and Applications of Graphs, Springer, Heidelberg (1976)
26. Peng, S.-L., Ko, M.-T., Ho, C.-W., Hsu, T.-s., Tang, C.Y.: Graph searching on some subclasses of chordal graphs. Algorithmica 27, 395–426 (2000)
27. Peng, S.-L., Ho, C.-W., Hsu, T.-s., Ko, M.-T., Tang, C.Y.: Edge and node searching problems on trees. Theor. Comput. Sci. 240, 429–446 (2000)
28. Petrov, N.N.: A problem of pursuit in the absence of information on the pursued. Differentsialnye Uravneniya 18, 1345–1352, 1468 (1982)
29. Skodinis, K.: Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. J. Algorithms 47, 40–59 (2003)
30. Suchan, K., Todinca, I.: Pathwidth of circular-arc graphs. In: Proceedings of WG 2007. LNCS, vol. 4769, pp. 258–269. Springer, Heidelberg (2007)
31. Takahashi, A., Ueno, S., Kajitani, Y.: Mixed searching and proper-path-width. Theor. Comput. Sci. 137, 253–268 (1995)

The 2-Terminal-Set Path Cover Problem and Its Polynomial Solution on Cographs*

Katerina Asdre and Stavros D. Nikolopoulos

Department of Computer Science, University of Ioannina
P.O. Box 1186, GR-45110 Ioannina, Greece
{katerina, stavros}@cs.uoi.gr

Abstract. In this paper we study a generalization of the path cover problem, namely, the 2-terminal-set path cover problem, or 2TPC for short. Given a graph G and two disjoint subsets $\mathcal{T}^1$ and $\mathcal{T}^2$ of $V(G)$, a 2-terminal-set path cover of G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the vertices of $\mathcal{T}^1$ and $\mathcal{T}^2$ are all endpoints of the paths in $\mathcal{P}$ and all the paths with both endpoints in $\mathcal{T}^1 \cup \mathcal{T}^2$ have one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$. The 2TPC problem is to find a 2-terminal-set path cover of G of minimum cardinality; note that, if $\mathcal{T}^1 \cup \mathcal{T}^2$ is empty, the stated problem coincides with the classical path cover problem. The 2TPC problem generalizes some path cover related problems, such as the 1HP and 2HP problems, which have been proved to be NP-complete even for small classes of graphs. We show that the 2TPC problem can be solved in linear time on the class of cographs. The proposed linear-time algorithm is simple, requires linear space, and also enables us to solve the 1HP and 2HP problems on cographs within the same time and space complexity.

Keywords: path cover, fixed-endpoint path cover, perfect graphs, complement reducible graphs, cographs, linear-time algorithms.

1 Introduction

Framework–Motivation. A well studied problem with numerous practical applications in graph theory is to find a minimum number of vertex-disjoint paths of a graph G that cover the vertices of G. This problem, also known as the path cover problem (PC), finds application in the fields of database design, networks, code optimization among many others (see [1,2,16,21]); it is well known that the path cover problem and many of its variants are NP-complete in general graphs [10]. A graph that admits a path cover of size one is referred to as Hamiltonian. Thus, the path cover problem is at least as hard as the Hamiltonian path problem (HP), that is, the problem of deciding whether a graph is Hamiltonian.

* The research Project is co-funded by the European Union - European Social Fund (ESF) & National Sources, in the framework of the program "Pythagoras II" of the 3rd Community Support Framework of the Hellenic Ministry of Education.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 208–220, 2008.

Several variants of the HP problem are also of great interest, among which is the problem of deciding whether a graph admits a Hamiltonian path between two points (2HP). The 2HP problem is the same as the HP problem except that in 2HP two vertices of the input graph G are specified, say, u and v, and we are asked whether G contains a Hamiltonian path beginning with u and ending with v. Similarly, the 1HP problem is to determine whether a graph G admits a Hamiltonian path starting from a specific vertex u of G, and to find one if such a path does exist. Both 1HP and 2HP problems are also NP-complete in general graphs [10].

The path cover problem and several variants of it have numerous algorithmic applications in many fields. Some that have received both theoretical and practical attention are in the content of communication and/or transposition networks [22]. In such problems, we are given a graph (network) G and (Problem A) a set $\mathcal{T}$ of $k = 2\lambda$ vertices of G, and the objective is to determine whether G admits a path cover of size λ that contains paths connecting pairs of vertices of $\mathcal{T}$, that is, G admits λ vertex-disjoint paths with both their endpoints in $\mathcal{T}$ (note that, the endpoints of a path P are the first vertex and the last vertex visited by P), or (Problem B) a set $\mathcal{T}$ of $\lambda = k/2$ pairs of vertices of G (source-sink pairs), and the objective is to determine whether G admits for each pair (a_i, b_i), $1 \leq i \leq \lambda$, a path connecting a_i to b_i such that the set of λ paths forms a path cover.

Another path cover related problem that has received increased attention in recent years is in the context of communication networks. The only efficient way to transmit high volume communication, such as in multimedia applications, is through disjoint paths that are dedicated to pairs of processors. To efficiently utilize the network one needs a simple algorithm that, with minimum overhead, constructs a large number of edge-disjoint paths between pairs of two given sets of requests.

Both problems A and B coincide with the 2HP problem, in the case where $k = 2$. In [9], Damaschke provided a foundation for obtaining polynomial-time algorithms for several problems concerning paths in interval graphs, such as finding Hamiltonian paths and circuits, and partitions into paths. In the same paper, he stated that the complexity status of both 1HP and 2HP problems on interval graphs remains an open question; until now the complexities of 1HP and 2HP keep their difficulties even in the small subclass of split interval graphs – no polynomial algorithm is known.

Motivated by the above issues we state a variant of the path cover problem, namely, the 2-terminal-set path cover problem (2TPC), which generalizes both 1HP and 2HP problems, and also Problem B.

(Problem 2TPC) Let G be a graph and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two disjoint sets of vertices of $V(G)$. A *2-terminal-set path cover* of the graph G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ is a path cover of G such that all vertices in $\mathcal{T}^1 \cup \mathcal{T}^2$ are endpoints of paths in the path cover and all the paths with both endpoints in $\mathcal{T}^1 \cup \mathcal{T}^2$ have one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$; a *minimum 2-terminal-set path cover* of G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ is a 2-terminal-set path cover of G with minimum

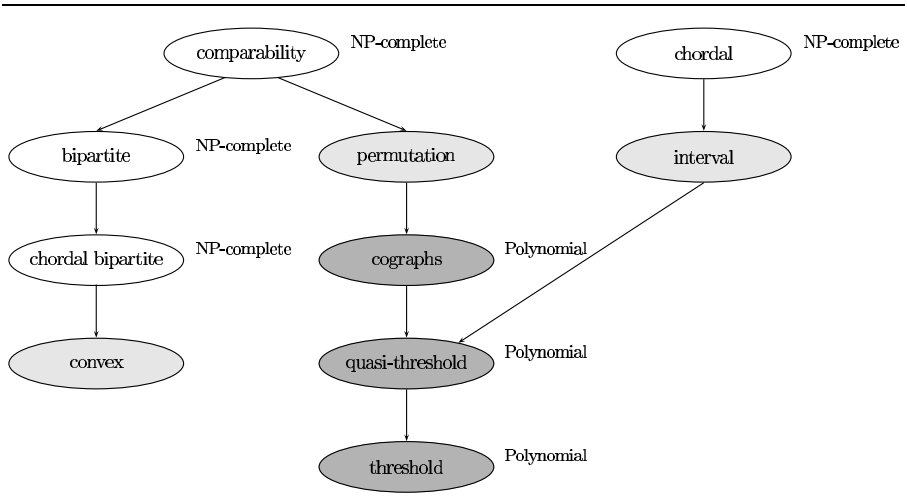


Fig. 1. The complexity status (NP-complete, unknown, polynomial) of the 2TPC problem for some graph subclasses of comparability and chordal graphs. $A \rightarrow B$ indicates that class A contains class B.

cardinality; the *2-terminal-set path cover problem* (2TPC) is to find a minimum 2-terminal-set path cover of the graph G.

Contribution. In this paper, we show that the 2-terminal-set path cover problem (2TPC) has a polynomial-time solution in the class of complement reducible graphs, or cographs [8]. More precisely, we establish a lower bound on the size of a minimum 2-terminal-set path cover of a cograph G on n vertices and m edges. We then define path operations, and prove structural properties for the paths of such a path cover, which enable us to describe a simple algorithm for the 2TPC problem. The proposed algorithm runs in time linear in the size of the input graph G, that is, in $O(n+m)$ time, and requires linear space. Figure 1 shows a diagram of class inclusions for a number of graph classes, subclasses of comparability and chordal graphs, and the current complexity status of the 2TPC problem on these classes; for definitions of the classes shown, see [6,11]. Note that, if the problem is polynomially solvable on interval graphs, then it is also polynomially solvable on convex graphs [18].

The proposed algorithm for the 2TPC problem can also be used to solve the 1HP and 2HP problems on cographs within the same time and space complexity. Moreover, we have designed our algorithm so that it produces a minimum 2-terminal-set path cover of a cograph G that contains a large number of paths with one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$ (we can easily find a graph G and two sets $\mathcal{T}^1$ and $\mathcal{T}^2$ of vertices of $V(G)$ so that G admits two minimum 2-terminal-set path covers with different numbers of paths having one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$; for example, consider the graph G with vertex set $V(G) = \{a, b, c, d\}$, edge set $E(G) = \{ab, bc, ac, cd\}$, and $\mathcal{T}^1 = \{a\}$, $\mathcal{T}^2 = \{b\}$).

Related Work. The class of cographs has been extensively studied and several sequential and/or parallel algorithms for recognition and for classical combinatorial optimization problems have been proposed [8,16,19]. Lin et al. [16] presented an optimal algorithm for the path cover problem on cographs, while Nakano et al. [19] described an optimal parallel algorithm which finds and reports all the paths in a minimum path cover of a cograph in $O(\log n)$ time using $O(n/\log n)$ processors on a PRAM model. Recently, Asdre and Nikolopoulos proposed a linear-time algorithm for the k-fixed-endpoint path cover problem (kPC) on cographs and on proper interval graphs [3,4]. Algorithms for optimization problems on other related classes of graphs have been also described [5,12,13,14,20]. Moreover, algorithms for the path cover problem on other classes of graphs were proposed in [2,15,21].

2 Theoretical Framework

The cographs admit a tree representation unique up to isomorphism. Specifically, we can associate with every cograph G a unique rooted tree $T_{co}(G)$ called the co-tree (or, modular decomposition tree [17]), which we can construct sequentially in linear time [7,8]. The co-tree forms the basis for fast algorithms for problems such as isomorphism, coloring, clique detection, clusters, minimum weight dominating sets [6,7], and also for the path cover problem [16,19].

For convenience and ease of presentation, we binarize the co-tree $T_{co}(G)$ in such a way that each of its internal nodes has exactly two children [16,19]. We shall refer to the binarized version of $T_{co}(G)$ as the modified co-tree of G and will denote it by $T(G)$. Thus, the left and right child of an internal node t of $T(G)$ will be denoted by t_ℓ and t_r, respectively. Let t be an internal node of $T(G)$. Then $G[t]$ is the subgraph of G induced by the subset V_t of the vertex set $V(G)$, which contains all the vertices of G that have as common ancestor in $T(G)$ the node t. For simplicity, we will denote by V_ℓ and V_r the vertex sets $V(G[t_\ell])$ and $V(G[t_r])$, respectively.

Let G be a cograph, $\mathcal{T}^1$ and $\mathcal{T}^2$ be two sets of vertices of $V(G)$ such that $\mathcal{T}^1 \cap \mathcal{T}^2 = \emptyset$, and let $\mathcal{P}_{2\mathcal{T}}(G)$ be a minimum 2-terminal-set path cover of G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ of size $\lambda_{2\mathcal{T}}$; note that the size of $\mathcal{P}_{2\mathcal{T}}(G)$ is the number of paths it contains. The vertices of the sets $\mathcal{T}^1$ and $\mathcal{T}^2$ are called *terminal* vertices, and the sets $\mathcal{T}^1$ and $\mathcal{T}^2$ are called the *terminal sets* of G, while those of $V(G) - (\mathcal{T}^1 \cup \mathcal{T}^2)$ are called *non-terminal or free* vertices. Thus, the set $\mathcal{P}_{2\mathcal{T}}(G)$ contains three types of paths, which we call *terminal*, *semi-terminal*, and *non-terminal or free* paths:

(i) a *terminal path* P_t consists of at least two vertices and both its endpoints, say, u and v, are terminal vertices belonging to different sets, that is, $u \in \mathcal{T}^1$ and $v \in \mathcal{T}^2$;
(ii) a *semi-terminal path* P_s is a path having one endpoint in $\mathcal{T}^1$ or $\mathcal{T}^2$ and the other in $V(G) - (\mathcal{T}^1 \cup \mathcal{T}^2)$; if P_s consists of only one vertex (trivial path), say, u, then $u \in \mathcal{T}^1 \cup \mathcal{T}^2$;

(iii) a *non-terminal or free path* P_f is a path having both its endpoints in $V(G)-(\mathcal{T}^1\cup\mathcal{T}^2)$; if P_f consists of only one vertex, say, u, then $u \in V(G)-(\mathcal{T}^1\cup\mathcal{T}^2)$.

The set of the non-terminal paths in a minimum 2TPC of the graph G is denoted by N, while S and T denote the sets of the semi-terminal and terminal paths, respectively. Furthermore, let S^1 and S^2 denote the sets of the semi-terminal paths such that the terminal vertices belong to $\mathcal{T}^1$ and $\mathcal{T}^2$, respectively. Thus, $|S| = |S^1| + |S^2|$ and the following equation holds.

$$\lambda_{2\mathcal{T}} = |N| + |S| + |T| = |N| + |S^1| + |S^2| + |T| \tag{1}$$

From the definition of the 2-terminal-set path cover problem (2TPC), we can easily conclude that the number of paths in a minimum 2TPC can not be less than the number of the terminal vertices of the terminal set having maximum cardinality. Furthermore, since each semi-terminal path contains one terminal vertex and each terminal path contains two, the number of terminal vertices is equal to $|S|+2|T| = |S^1|+|S^2|+2|T|$. Thus, we have the following proposition, which also holds for general graphs:

Proposition 2.1. *Let G be a cograph and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two disjoint subsets of $V(G)$. Then $|\mathcal{T}^1| = |S^1| + |T|$, $|\mathcal{T}^2| = |S^2| + |T|$ and $\lambda_{2\mathcal{T}} \geq \max\{|\mathcal{T}^1|, |\mathcal{T}^2|\}$.*

Clearly, the size of a 2TPC of a cograph G, as well as the size of a minimum 2TPC of G, is less than or equal to the number of vertices of G, that is, $\lambda_{2\mathcal{T}} \leq |V(G)|$. Let $F(V(G))$ be the set of the free vertices of G; hereafter, $F(V) = F(V(G))$. Furthermore, let $\mathcal{P}$ be a set of paths and let $V_{\mathcal{P}}$ denote the set of vertices belonging to the paths of the set $\mathcal{P}$; hereafter, $F(\mathcal{P}) = F(V_{\mathcal{P}})$. Then, if $\mathcal{T}^1$ and $\mathcal{T}^2$ are two disjoint subsets of $V(G)$, we have $\lambda_{2\mathcal{T}} \leq |F(V)|+|\mathcal{T}^1|+|\mathcal{T}^2|$.

Let t be an internal node of the tree $T(G)$, that is, t is either an S-node or a P-node [17]. Then $\lambda_{2\mathcal{T}}(t)$ denotes the number of paths in a minimum 2TPC of the graph $G[t]$ with respect to $\mathcal{T}_t^1$ and $\mathcal{T}_t^2$, where $\mathcal{T}_t^1$ and $\mathcal{T}_t^2$ are the terminal vertices of $\mathcal{T}^1$ and $\mathcal{T}^2$ of the graph $G[t]$, respectively. Let t_ℓ and t_r be the left and the right child of node t, respectively. We denote by $\mathcal{T}_\ell^1$ and $\mathcal{T}_r^1$ (resp. $\mathcal{T}_\ell^2$ and $\mathcal{T}_r^2$) the terminal vertices of $\mathcal{T}^1$ (resp. $\mathcal{T}^2$) in V_ℓ and V_r, respectively, where $V_\ell = V(G[t_\ell])$ and $V_r = V(G[t_r])$. Let N_ℓ, S_ℓ and T_ℓ be the sets of the non-terminal, semi-terminal and terminal paths in a minimum 2TPC of $G[t_\ell]$, respectively. Similarly, let N_r, S_r and T_r be the sets of the non-terminal, semi-terminal and terminal paths in a minimum 2TPC of $G[t_r]$, respectively. Note that S_ℓ^1 and S_r^1 (resp. S_ℓ^2 and S_r^2) denote the sets of the semi-terminal paths in a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively, containing a terminal vertex of $\mathcal{T}^1$ (resp. $\mathcal{T}^2$). Obviously, Eq. (1) holds for $G[t]$ as well, with t being either an S-node or a P-node, that is,

$$\lambda_{2\mathcal{T}}(t) = |N_t| + |S_t| + |T_t| = |N_t| + |S_t^1| + |S_t^2| + |T_t| \tag{2}$$

where N_t, S_t and T_t are the sets of the non-terminal, the semi-terminal and the terminal paths, respectively, in a minimum 2TPC of $G[t]$, that is in $\mathcal{P}_{2\mathcal{T}}(t)$, and S_t^1 and S_t^2 denote the sets of the semi-terminal paths in $\mathcal{P}_{2\mathcal{T}}(t)$ containing

a terminal vertex of $\mathcal{T}^1$ and $\mathcal{T}^2$, respectively. If t is a P-node, then $\mathcal{P}_{2\mathcal{T}}(t) = \mathcal{P}_{2\mathcal{T}}(t_\ell) \cup \mathcal{P}_{2\mathcal{T}}(t_r)$, where $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ are minimum 2TPCs corresponding to $G[t_\ell]$ and $G[t_r]$, respectively, and $\lambda_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) + \lambda_{2\mathcal{T}}(t_r)$. Furthermore, in the case where t is a P-node, we have

$$
\begin{aligned}
|N_t| &= |N_\ell| + |N_r| \\
|S_t| &= |S_\ell| + |S_r| = |S_\ell^1| + |S_\ell^2| + |S_r^1| + |S_r^2| \\
|T_t| &= |T_\ell| + |T_r|
\end{aligned}
$$

Thus, we focus on computing a minimum 2TPC of the graph $G[t]$ for the case where t is an S-node. Before describing our algorithm, we establish a lower bound on the size $\lambda_{2\mathcal{T}}(t)$ of a minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(t)$ of a graph $G[t]$. More precisely, we prove the following lemma.

Lemma 2.1. *Let t be an internal node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. Then $\lambda_{2\mathcal{T}}(t) \geq \max\{\lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|, \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|, \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}\}$.*

We next define four operations on paths of a minimum 2TPC of the graphs $G[t_\ell]$ and $G[t_r]$, namely *break*, *connect*, *bridge* and *insert* operations; these operations are illustrated in Fig. 2.

Break Operation: Let $P = [p_1, p_2, \ldots, p_k]$ be a path of $\mathcal{P}_{2\mathcal{T}}(t_r)$ or $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ of length k. We say that we *break* the path P in two paths, say, P_1 and P_2, if we delete an arbitrary edge of P, say the edge $p_i p_{i+1}$ $(1 \leq i < k)$, in order to obtain two paths which are $P_1 = [p_1, \ldots, p_i]$ and $P_2 = [p_{i+1}, \ldots, p_k]$. Note that we can break the path P in at most k trivial paths.

Connect Operation: Let P_1 be a non-terminal or a semi-terminal path of $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ (resp. $\mathcal{P}_{2\mathcal{T}}(t_r)$) and let P_2 be a non-terminal or a semi-terminal path of $\mathcal{P}_{2\mathcal{T}}(t_r)$ (resp. $\mathcal{P}_{2\mathcal{T}}(t_\ell)$). We say that we *connect* the path P_1 with the path P_2, if we add an edge which joins two free endpoints of the two paths. Note that if $P_1 \in S_\ell^1$ (resp. $P_1 \in S_r^1$) then, if P_2 is also a semi-terminal path, $P_2 \in S_r^2$ (resp. $P_2 \in S_\ell^2$). Similarly, if $P_1 \in S_\ell^2$ (resp. $P_1 \in S_r^2$) then, if P_2 is also a semi-terminal path, $P_2 \in S_r^1$ (resp. $P_2 \in S_\ell^1$).

Bridge Operation: Let P_1 and P_2 be two paths of the set $N_\ell \cup S_\ell^1 \cup S_\ell^2$ (resp. $N_r \cup S_r^1 \cup S_r^2$) and let P_3 be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *bridge* the two paths P_1 and P_2 using path P_3 if we connect a free endpoint of P_1 with one endpoint of P_3 and a free endpoint of P_2 with the other endpoint of P_3. The result is a path having both endpoints in $G[t_\ell]$ (resp. $G[t_r]$). Note that if $P_1 \in S_\ell^1$ (resp. $P_1 \in S_r^1$) then, if P_2 is also a semi-terminal path, $P_2 \in S_\ell^2$ (resp. $P_2 \in S_r^2$). Similarly, if $P_1 \in S_\ell^2$ (resp. $P_1 \in S_r^2$) then, if P_2 is also a semi-terminal path, $P_2 \in S_\ell^1$ (resp. $P_2 \in S_r^1$).

Insert Operation: Let $P_1 = [t_1, p_1, \ldots, p'_1, t'_1]$ be a terminal path of the set T_ℓ (resp. T_r) and let $P_2 = [p_2, \ldots, p'_2]$ be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *insert* the path P_2 into P_1, if we replace the first edge of

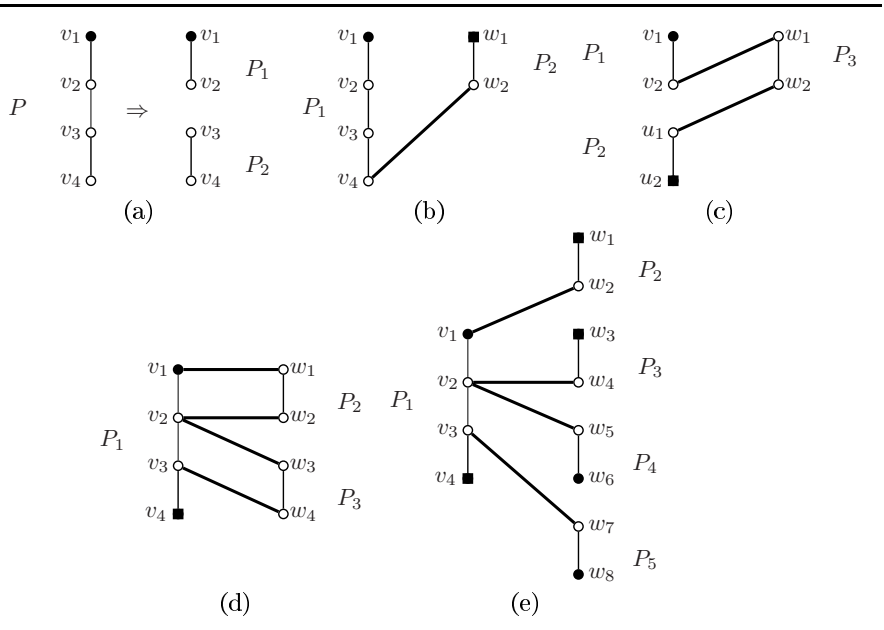


Fig. 2. Illustrating (a) break, (b) connect, (c) bridge, (d) insert, and (e) connect-bridge operations; the vertices of $\mathcal{T}^1$ are denoted by black-circles, while the vertices of $\mathcal{T}^2$ are denoted by black-squares

P_1, that is, the edge t_1p_1, with the path $[t_1, p_2, \ldots, p'_2, p_1]$. Thus, the resulting path is $P_1 = [t_1, p_2, \ldots, p'_2, p_1, \ldots, p'_1, t'_1]$. Note that we can replace every edge of the terminal path so that we can insert at most $|F(\{P_1\})| + 1$ non-terminal paths, where $F(\{P_1\})$ is the set of the free vertices belonging to the path P_1. If the terminal path $P_1 = [t_1, p_1, \ldots, p^\ell_1, p^r_1, \ldots, p'_1, t'_1]$ is constructed by connecting a semi-terminal path of S_ℓ, say, $P_\ell = [t_1, p_1, \ldots, p^\ell_1]$ with a semi-terminal path of S_r, say, $P_r = [p^r_1, \ldots, p'_1, t'_1]$, then it obviously has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$. In this case, if $P_2 \in N_\ell$ (resp. N_r) we can only replace the edges of P_1 that belong to $G[t_r]$ (resp. $G[t_\ell]$). On the other hand, if P_2 has one endpoint, say, p_2, in N_ℓ and the other, say, p'_2, in N_r, we insert P_2 into P_1 as follows: $P_1 = [t_1, p_1, \ldots, p^\ell_1, p'_2, \ldots, p_2, p^r_1, \ldots, p'_1, t'_1]$.

We can also combine the operations connect and bridge to perform a new operation which we call a *connect-bridge* operation; such an operation is depicted in Fig. 2(e) and is defined below.

Connect-Bridge Operation: Let $P_1 = [t_1, p_1, \ldots, p_k, t'_1]$ be a terminal path of the set T_ℓ (resp. T_r), where $t_1 \in \mathcal{T}^2$ and $t'_1 \in \mathcal{T}^1$, and let $P_2, P_3, \ldots, P_{\frac{s+1}{2}}$ be semi-terminal paths of the set S^1_r (resp. S^1_ℓ) and $P_{\frac{s+1}{2}+1}, \ldots, P_s$ be semi-terminal paths of the set S^2_r (resp. S^2_ℓ), where s is odd and $3 \leq s \leq 2k + 3$. We say that

we *connect-bridge* the paths $P_2, P_3, \ldots, P_s$ using vertices of P_1, if we perform the following operations: (i) connect the path P_2 with the path $[t_1]$; (ii) bridge $r = \frac{s-3}{2}$ pairs of different semi-terminal paths using vertices $p_1, p_2, \ldots, p_r$; and (iii) connect the path $[p_{r+1}, \ldots, p_k, t'_1]$ with the last semi-terminal path P_s.

The Connect-Bridge operation produces two paths having one endpoint in $G[t_\ell]$ and the other in $G[t_r]$ and $\frac{s-3}{2}$ paths having both endpoints in $G[t_r]$ (resp. $G[t_\ell]$).

3 The Algorithm

We next present an algorithm for the 2TPC problem on cographs. Our algorithm takes as input a cograph G and two vertex subsets $\mathcal{T}^1$ and $\mathcal{T}^2$, where $\mathcal{T}^1 \cap \mathcal{T}^2 = \emptyset$, and finds the paths of a minimum 2TPC of G in linear time; it works as follows:

Algorithm Minimum_2TPC

1. Construct co-tree $T_{co}(G)$ and make it binary; let $T(G)$ be the resulting tree;
2. Execute subroutine *process*(*root*), where *root* is the root of $T(G)$; the minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(root) = \mathcal{P}_{2\mathcal{T}}(G)$ is the set of paths returned by the subroutine;

where the description of the subroutine *process*() is as follows:

process (node t)

1. `if` t is a leaf
   `then` return($\{u\}$), where u is the vertex associated with the leaf t;
   `else` {t is an internal node with left and right child denoted by t_ℓ and t_r.}
   $\mathcal{P}_{2\mathcal{T}}(t_\ell) \leftarrow process(t_\ell)$;
   $\mathcal{P}_{2\mathcal{T}}(t_r) \leftarrow process(t_r)$;
2. `if` t is a P-node
   `then` return($\mathcal{P}_{2\mathcal{T}}(t_\ell) \cup \mathcal{P}_{2\mathcal{T}}(t_r)$);
3. `if` t is an S-node
   `then if` $|N_\ell| \leq |N_r|$ `then` swap($\mathcal{P}_{2\mathcal{T}}(t_\ell), \mathcal{P}_{2\mathcal{T}}(t_r)$);
   $s^1 = |S^1_\ell| - |S^2_r|$;
   $s^2 = |S^2_\ell| - |S^1_r|$;
   `case 1:` $s^1 \geq 0$ and $s^2 \geq 0$
   call procedure $2TPC\_1$;
   `case 2:` $s^1 < 0$ and $s^2 < 0$
   `if` $|N_r| + \min\{|s^1|, |s^2|\} \leq |F(S^1_\ell \cup S^2_\ell \cup N_\ell)|$
   `then` call procedure $2TPC\_2\_a$;
   `else` call procedure $2TPC\_2\_b$;
   `case 3:` ($s^1 \geq 0$ and $s^2 < 0$) or ($s^1 < 0$ and $s^2 \geq 0$)
   call procedure $2TPC\_3$;

We next describe the subroutine *process*() in the case where t is an S-node of $T(G)$. Note that, if $|N_\ell| \leq |N_r|$, we swap $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$. Thus, we

distinguish the following three cases: (1) $s^1 \geq 0$ and $s^2 \geq 0$, (2) $s^1 < 0$ and $s^2 < 0$, and (3) ($s^1 \geq 0$ and $s^2 < 0$) or ($s^1 < 0$ and $s^2 \geq 0$). We next describe case 1; cases 2 and 3 are similar.

Case 1: $s^1 \geq 0$ and $s^2 \geq 0$

Let SN_r be the set of non-terminal paths obtained by breaking the set $S_r^1 \cup S_r^2 \cup N_r$ into $|N_\ell| - 1 + \min\{s^1, s^2\}$ non-terminal paths; thus, $|SN_r| \leq |F(S_r^1 \cup S_r^2 \cup N_r)|$. In the case where $|N_\ell| - 1 + \min\{s^1, s^2\} \geq F(S_r^1 \cup S_r^2 \cup N_r)$, the paths of SN_r are trivial (recall that $F(S_r^1 \cup S_r^2 \cup N_r)$ is the set of free vertices belonging to the set $S_r^1 \cup S_r^2 \cup N_r$). The paths of SN_r are used to bridge at most $2\min\{s^1, s^2\}$ semi-terminal paths of $S_\ell^1 \cup S_\ell^2$ and, if $|SN_r| - \min\{s^1, s^2\} > 0$, at most $|N_\ell|$ non-terminal paths of N_ℓ. We can construct the paths of a 2TPC using the following procedure:

Procedure 2TPC_1

1. connect the $|S_r^2|$ paths of S_r^2 with $|S_r^2|$ paths of S_ℓ^1, and the $|S_r^1|$ paths of S_r^1 with $|S_r^1|$ paths of S_ℓ^2;
2. bridge $2\min\{s^1, s^2\}$ semi-terminal paths of $S_\ell^1 \cup S_\ell^2$ using $\min\{s^1, s^2\}$ paths of SN_r;
3. bridge the non-terminal paths of N_ℓ using $|N_\ell| - 1$ non-terminal paths of SN_r; this produces non-terminal paths with both endpoints in $G[t_\ell]$, unless $|N_\ell| \leq |F(S_r^1 \cup S_r^2 \cup N_r)| - \min\{s^1, s^2\}$ where we obtain one non-terminal path with one endpoint in $G[t_\ell]$ and the other in $G[t_r]$;
4. if $|N_\ell| \leq |F(S_r^1 \cup S_r^2 \cup N_r)| - \min\{s^1, s^2\}$ insert the non-terminal path obtained in Step 3 into one terminal path which is obtained in Step 1;
5. if $|T_r| = |S_\ell^1| = |S_\ell^2| = 0$ and $|F(S_r^1 \cup S_r^2 \cup N_r)| \geq |N_\ell| + 1$ construct a non-terminal path having both of its endpoints in $G[t_r]$ and insert it into a terminal path of T_ℓ;
6. if $|T_r| = |S_r^1| = |S_r^2| = 0$ and $|F(N_r)| \geq |N_\ell| + \min\{s^1, s^2\}$ construct a non-terminal path having both of its endpoints in $G[t_r]$ and use it to connect two semi-terminal paths of $S_\ell^1 \cup S_\ell^2$;
7. if $s^1 - \min\{\min\{s^1, s^2\}, |F(S_r^1 \cup S_r^2 \cup N_r)|\}$ (resp. $s^2 - \min\{\min\{s^1, s^2\}, |F(S_r^1 \cup S_r^2 \cup N_r)|\}$) is odd and there is at least one free vertex in $S_r^1 \cup S_r^2 \cup N_r$ which is not used in Steps 1–6, or there is a non-terminal path having one endpoint in $G[t_\ell]$ and the other in $G[t_r]$, connect one non-terminal path with one semi-terminal path of S_ℓ^1 (resp. S_ℓ^2);
8. connect-bridge the rest of the semi-terminal paths of $S_\ell^1 \cup S_\ell^2$ (at most $2(|F(T_r)| + |T_r|)$) using vertices of T_r;
9. insert non-terminal paths obtained in Step 3 into the terminal paths of T_r;

Based on the procedure 2TPC_1, we can compute the cardinality of the sets N_t, S_t^1, S_t^2 and T_t, and thus, since $\lambda'_{2\mathcal{T}}(t) = |N_t| + |S_t| + |T_t|$ and $|S_t| = S_t^1 + S_t^2$, the number of paths in the 2TPC constructed by the procedure at node $t \in T(G)$.

In this case, the values of $|N_t|$, $|S_t|$ and $|T_t|$ are the following:

$$\begin{aligned}
|N_t| &= \max\{\mu - \alpha, 0\} \\
|S_t^1| &= \min\{\sigma_\ell^1, \max\{\sigma_\ell^1 - |F(T_r)| - |T_r|, \max\{\sigma_\ell^1 - \sigma_\ell^2, 0\}\}\} \\
|S_t^2| &= \min\{\sigma_\ell^2, \max\{\sigma_\ell^2 - |F(T_r)| - |T_r|, \max\{\sigma_\ell^2 - \sigma_\ell^1, 0\}\}\} \\
|S_t| &= |S_t^1| + |S_t^2| \\
|T_t| &= |S_r^1| + |S_r^2| + \min\{\min\{s^1, s^2\}, |F(S_r^1 \cup S_r^2 \cup N_r)|\} + |T_\ell| + |T_r| + \\
&\quad \frac{\sigma_\ell^1 + \sigma_\ell^2 - |S_t|}{2}
\end{aligned} \tag{3}$$

where

$$\begin{aligned}
\sigma_\ell^1 &= |S_\ell^1| - |S_r^2| - \min\{\min\{s^1, s^2\}, |F(S_r^1 \cup S_r^2 \cup N_r)|\}, \\
\sigma_\ell^2 &= |S_\ell^2| - |S_r^1| - \min\{\min\{s^1, s^2\}, |F(S_r^1 \cup S_r^2 \cup N_r)|\}, \\
\mu &= \max\{|N_\ell| - \pi_r, \max\{1 - \max\{|S_\ell^1|, |S_\ell^2|\}, 0\}\} - \max\{|F(T_r)| + |T_r| - \\
&\quad \min\{\sigma_\ell^1, \sigma_\ell^2\}, 0\} - \min\{\max\{\min\{|N_\ell| - \pi_r, \delta(\sigma_\ell^1), \delta(\sigma_\ell^2)\}, 0\}, \\
&\quad \max\{\min\{F(S_r^1 \cup S_r^2 \cup N_r) - \min\{s^1, s^2\}, 1\}, 0\}\}, \\
\alpha &= \min\{\max\{\min\{\pi_r - |N_\ell|, 1\}, 0\}, \max\{|T_\ell|, 0\}\}, \text{ and} \\
\pi_r &= \max\{|F(S_r^1 \cup S_r^2 \cup N_r)| - \min\{s^1, s^2\}, 0\}.
\end{aligned}$$

In Eq. (3), σ_ℓ^1 (resp. σ_ℓ^2) is the number of semi-terminal paths of S_ℓ^1 (resp. S_ℓ^2) that are not connected or bridged at Steps 1–3. Furthermore, π_r is the number of free vertices in the set $S_r^1 \cup S_r^2 \cup N_r$ that are not used to bridge semi-terminal paths of $S_\ell^1 \cup S_\ell^2$ at Step 3 and δ is a function which is defined as follows: $\delta(x) = 1$, if x is odd, and $\delta(x) = 0$ otherwise. Note that at most $|F(T_r)| + |T_r|$ non-terminal paths can be inserted into the terminal paths of T_r or the terminal paths can connect-bridge at most $2(|F(T_r)| + |T_r|)$ semi-terminal paths.

4 Correctness and Time Complexity

Let G be a cograph, $T(G)$ be the modified co-tree of G, and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two terminal sets of G. Since our algorithm computes a 2TPC $\mathcal{P}'_{2\mathcal{T}}(t)$ of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t)$ for each internal node $t \in T(G)$, we need to prove that the constructed 2TPC $\mathcal{P}'_{2\mathcal{T}}(t)$ is minimum. Obviously, the size $\lambda_{2\mathcal{T}}(t)$ of a minimum 2TPC of $G[t]$ is less than or equal to the size $\lambda'_{2\mathcal{T}}(t)$ of the 2TPC constructed by our algorithm. According to Proposition 2.1, if the size of the 2TPC constructed by our algorithm is $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$, then it is a minimum 2TPC. After performing simple computations we get four specific values for the size $\lambda'_{2\mathcal{T}}(t)$ of the 2TPC constructed by our algorithm, that is, by the 2TPC procedures 1, 2_a, 2_b and 3. More precisely, if t is an internal S-node of $T(G)$, our algorithm returns a 2TPC of size $\lambda'_{2\mathcal{T}}(t)$ equal to either $\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\} + 1$, $\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$, $\lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$, or $\lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|$; see Table 1. Specifically, in the case where $|S_\ell^1| = |S_\ell^2| = |T_r| = |S_r^1| = |S_r^2| = 0$ and $|N_\ell| = |V_r|$ procedure 2TPC_1 returns a 2TPC of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\} + 1$. We prove the following lemma, which shows that if the size of the 2TPC returned by subroutine process(t) for $G[t]$ is $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\} + 1$ (procedure 2TPC_1), then it is a min 2TPC.

Table 1. The size of the 2TPC that our algorithm returns in each case

Procedures	Size of 2-terminal-set PC
Procedure 2TPC_1	$\max\{\|\mathcal{T}_t^1\|, \|\mathcal{T}_t^2\|\} + 1$
All the procedures	$\max\{\|\mathcal{T}_t^1\|, \|\mathcal{T}_t^2\|\}$
Procedures 2TPC_1, 2TPC_2_a and 2TPC_3	$\lambda_{2\mathcal{T}}(t_\ell) - \|F(V_r)\|$
Procedure 2TPC_2_b	$\lambda_{2\mathcal{T}}(t_r) - \|F(V_\ell)\|$

Lemma 4.1. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. If $|S_\ell^1| = |S_\ell^2| = |T_r| = |S_r^1| = |S_r^2| = 0$ and $|N_\ell| = |V_r|$, then the procedure 2TPC_1 returns a minimum 2TPC of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\} + 1$.*

Moreover, if the size of the 2TPC returned by the process(t) is $\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$ (all the procedures), then it is obviously a minimum 2TPC of $G[t]$. We prove that the size $\lambda'_{2\mathcal{T}}(t)$ of the 2TPC $\mathcal{P}'_{2\mathcal{T}}(t)$ that process(t) returns is minimum.

Lemma 4.2. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, resp. If the subroutine process(t) returns a 2TPC of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$, then $\lambda'_{2\mathcal{T}}(t) \geq \max\{\lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|, \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\}$.*

Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, resp. Furthermore, we assume that the conditions $|S_\ell^1| = |S_\ell^2| = |T_r| = |S_r^1| = |S_r^2| = 0$ and $|N_\ell| = |V_r|$ do not hold together. We consider the case where process(t) returns a 2TPC $\mathcal{P}'_{2\mathcal{T}}(t)$ of the graph $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$ (cases 1, 2.a and 3). We prove the following lemma.

Lemma 4.3. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. If the subroutine process(t) returns a 2TPC of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$, then $\lambda'_{2\mathcal{T}}(t) > \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}, \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\}$.*

Similarly we can show that if process(t) returns a 2TPC of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|$ (case 2.b), then $\lambda'_{2\mathcal{T}}(t) > \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}, \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|\}$. Thus, we can prove the following result.

Lemma 4.4. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, resp. Subroutine process(t) returns a 2TPC $\mathcal{P}_{2\mathcal{T}}(t)$ of $G[t]$ of size*

$$\lambda'_{2\mathcal{T}}(t) = \begin{cases} \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\} + 1 & \text{if } |N_\ell| = |V_r| \text{ and} \\ & |S_\ell^1| = |S_\ell^2| = |\mathcal{T}_r^1| = |\mathcal{T}_r^2| = 0, \\ \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}, & \\ \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|, \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\} & \text{otherwise.} \end{cases}$$

Obviously, a minimum 2TPC of the graph $G[t]$ is of size $\lambda_{2\mathcal{T}}(t) \leq \lambda'_{2\mathcal{T}}(t)$. On the other hand, we have proved a lower bound for the size $\lambda_{2\mathcal{T}}(t)$ of a minimum 2TPC of the graph $G[t]$ (see Lemma 2.1). It follows that $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t)$, and, thus, we can state the following result.

Lemma 4.5. *Subroutine process(t) returns a minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(t)$ of the graph $G[t]$, for every internal S-node $t \in T(G)$.*

Since the above result holds for every S-node t of $T(G)$, it also holds for $t = t_{root}$ and $\mathcal{T}_t^1 = \mathcal{T}^1$ and $\mathcal{T}_t^2 = \mathcal{T}^2$. Thus, the following theorem holds:

Theorem 4.1. *Let G be a cograph and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two disjoint subsets of $V(G)$. Let t be the root of the modified co-tree $T(G)$, and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. Algorithm Minimum_2TPC correctly computes a minimum 2TPC of $G = G[t]$ with respect to $\mathcal{T}^1 = \mathcal{T}_t^1$ and $\mathcal{T}^2 = \mathcal{T}_t^2$ of size $\lambda_{2\mathcal{T}} = \lambda_{2\mathcal{T}}(t)$, where*

$$\lambda_{2\mathcal{T}}(t) = \begin{cases} \lambda_{2\mathcal{T}}(t_r) + \lambda_{2\mathcal{T}}(t_\ell) & \text{if } t \text{ is a P-node,} \\ \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\} + 1 & \text{if } t \text{ is an S-node and} \\ & |N_\ell| = |V_r| \text{ and} \\ & |S_\ell^1| = |S_\ell^2| = |\mathcal{T}_r^1| = |\mathcal{T}_r^2| = 0, \\ \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}, & \\ \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|, \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\} & \text{otherwise.} \end{cases}$$

Let G be a cograph on n vertices and m edges, $\mathcal{T}^1$ and $\mathcal{T}^2$ be two terminal sets, and let t be an S-node of $T(G)$. From the description of the algorithm we can easily conclude that a minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(t)$ of $G[t]$ can be constructed in $O(E(G[t]))$ time, since we use at most $|V(G[t_\ell])| \cdot |V(G[t_r])|$ edges to connect the paths of the minimum 2TPCs of the graphs $G[t_\ell]$ and $G[t_r]$; in the case where t is a P-node a minimum 2TPC is constructed in $O(1)$ time. Thus, the time needed by process(t) to compute a minimum 2TPC in the case where t is the root of $T(G)$ is $O(n+m)$; moreover, through the execution of the subroutine no additional space is needed. The construction of the co-tree $T_{co}(G)$ of G needs $O(n+m)$ time and it requires $O(n)$ space [7,8]. Furthermore, the binarization process of the co-tree, that is, the construction of the modified co-tree $T(G)$, takes $O(n)$ time. Hence, we can state the following result.

Theorem 4.2. *Let G be a cograph on n vertices and m edges and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two disjoint subsets of $V(G)$. A minimum 2-terminal-set path cover $\mathcal{P}_{2\mathcal{T}}$ of G can be computed in $O(n+m)$ time and space.*

References

1. Adhar, G.S., Peng, S.: Parallel algorithm for path covering, Hamiltonian path, and Hamiltonian cycle in cographs. In: Int'l Conference on Parallel Processing. Algorithms and Architecture, vol. III, pp. 364–365. Pennsylvania State Univ. Press (1990)
2. Arikati, S.R., Rangan, C.P.: Linear algorithm for optimal path cover problem on interval graphs. Inform. Process. Lett. 35, 149–153 (1990)
3. Asdre, K., Nikolopoulos, S.D.: A linear-time algorithm for the k-fixed-endpoint path cover problem on cographs. Networks 50, 231–240 (2007)
4. Asdre, K., Nikolopoulos, S.D.: A polynomial solution for the k-fixed-endpoint path cover problem on proper interval graphs. In: 18th Int'l Conference on Combinatorial Algorithms (IWOCA 2007), Lake Macquarie, Newcastle, Australia (2007)
5. Asdre, K., Nikolopoulos, S.D., Papadopoulos, C.: An optimal parallel solution for the path cover problem on P_4-sparse graphs. J. Parallel Distrib. Comput. 67, 63–76 (2007)
6. Brandstädt, A., Le, V.B., Spinrad, J.: Graph Classes – A Survey. In: SIAM Monographs in Discrete Mathematics and Applications. SIAM, Philadelphia (1999)
7. Corneil, D.G., Lerchs, H., Stewart, L.: Complement reducible graphs. Discrete. Appl. Math. 3, 163–174 (1981)
8. Corneil, D.G., Perl, Y., Stewart, L.: A linear recognition algorithm for cographs. SIAM J. Comput. 14, 926–984 (1985)
9. Damaschke, P.: Paths in interval graphs and circular arc graphs. Discrete Math. 112, 49–64 (1993)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman, San Francisco (1979)
11. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, 2nd edn. Academic Press, New York (1980); Annals of Discrete Mathematics 57, Elsevier (2004)
12. Hochstättler, W., Tinhofer, G.: Hamiltonicity in graphs with few P_4's. Computing 54, 213–225 (1995)
13. Hsieh, S.Y.: An efficient parallel strategy for the two-fixed-endpoint Hamiltonian path problem on distance-hereditary graphs. J. Parallel Distrib. Comput. 64, 662–685 (2004)
14. Hung, R.W., Chang, M.S.: Linear-time algorithms for the Hamiltonian problems on distance-hereditary graphs. Theoret. Comput. Sci. 341, 411–440 (2005)
15. Hung, R.W., Chang, M.S.: Solving the path cover problem on circular-arc graphs by using an approximation algorithm. Discrete. Appl. Math. 154, 76–105 (2006)
16. Lin, R., Olariu, S., Pruesse, G.: An optimal path cover algorithm for cographs. Comput. Math. Appl. 30, 75–83 (1995)
17. McConnell, R.M., Spinrad, J.: Modular decomposition and transitive orientation. Discrete Math. 201, 189–241 (1999)
18. Müller, H.: Hamiltonian circuits in chordal bipartite graphs. Discrete Math. 156, 291–298 (1996)
19. Nakano, K., Olariu, S., Zomaya, A.Y.: A time-optimal solution for the path cover problem on cographs. Theoret. Comput. Sci. 290, 1541–1556 (2003)
20. Nikolopoulos, S.D.: Parallel algorithms for Hamiltonian problems on quasi-threshold graphs. J. Parallel Distrib. Comput. 64, 48–67 (2004)
21. Srikant, R., Sundaram, R., Singh, K.S., Rangan, C.P.: Optimal path cover problem on block graphs and bipartite permutation graphs. Theoret. Comput. Sci. 115, 351–357 (1993)
22. Suzuki, Y., Kaneko, K., Nakamori, M.: Node-disjoint paths algorithm in a transposition graph. IEICE Trans. Inf. & Syst. E89-D, 2600–2605 (2006)

A Distributed Algorithm to Approximate Node-Weighted Minimum α-Connected (θ,k)-Coverage in Dense Sensor Networks

Yongan Wu, Min Li, Zhiping Cai, and En Zhu

School of Computer, National University of Defense Technology,
410073 Changsha, P.R. China
yongan@nudt.edu.cn

Abstract. The fundamental issue in sensor networks is providing a certain degree of coverage and maintaining connectivity under the energy constraint. In this paper, the connected k-coverage problem is investigated under the probabilistic sensing and communication models, which are more realistic than deterministic models. Furthermore, different weights for nodes are added in order to estimate the real power consumption. Because the problem is NP-hard, a *distributed probabilistic coverage and connectivity maintenance algorithm* (DPCCM) for dense sensor networks is proposed. DPCCM converts task requirement into two parameters by using the consequence of Chebyshev's inequality, then activate sensors based on the properties of weighted ε-net. It is proved that the sensors chosen by DPCCM have (θ,k)-coverage and α-connectivity. And the time and communication complexities are theoretically analyzed. Simulation results show that compared with the distributed randomized k-coverage algorithm, DPCCM significantly maintain coverage in probabilistic model and prolong the network lifetime in some sense.

Keywords: probabilistic model; (θ,k)-coverage; α-connectivity; dense sensor networks.

1 Introduction

Generally speaking, a wireless sensor network (WSN) is composed of a large number of small, autonomous sensors scattered in the hazardous or inaccessible environment. Applications of WSN include forest fire detection, vehicle traffic monitoring, battlefield surveillance, and so on [1-3].

The main goal of WSN is to provide information about a sensing field for an extended period of time. The quality of monitoring provided by WSN is usually measured by coverage. k-Coverage ($k \geq 1$) means that each point in the target area is monitored by at least k sensors. How to select appropriate active sensors to preserve required coverage as well as prolong the network lifetime at the same time is the coverage control problem, which is one of the most fundamental problems in WSN. Connectivity is closely-related to coverage, which ensures that there is at least one communication path between any pair of active sensors. The connected k-coverage problem has been studied extensively for more practical [4].

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 221–232, 2008.

However, there are some shortcomings in the traditional connected k-coverage protocols. First, some connectivity maintenance protocols assume the deterministic communication models for convenience, where node i successfully sends messages to j if j is in the communication range of i. Though they are accurate in wired networks, previous works have shown that communications of sensors are not deterministic but probabilistic [5]. Second, similarly to above, probabilistic sensing models are more practical than the deterministic ones assumed by some k-coverage protocols [6]. Third, traditional connected k-coverage protocols ignore the difference in power available (PA). They activate as few sensors as possible. Although they cost possibly little energy in one task, the protocols are not optimal in a series of tasks because of unbalanced energy. This also results in hotspots of energy consumption, which may cause premature death of sensors and even premature death of entire network.

In order to solve the problems, we propose a new connected k-coverage problem called α-*Connected* (θ, k)-*Coverage Set* ((α, θ, k)-*CCS*) problem, where $\alpha (0 < \alpha < 1)$ shows the connectivity metric, and (θ, k) $(0 < \theta < 1,\ k \in \mathbb{N}^*)$ represents the coverage. α-Connectivity means that any pair of active sensors communicate successfully with probability at least α. And (θ, k)-coverage means that each target point is monitored by at least k sensors with probability at least θ. It is worth of pointing out that θ and k aren't combined into expectation θk. This is because that θk-coverage may infeasible for (θ, k)-coverage. The result is obvious on the supposition that sensors have enough good sensing performance. To the best of our knowledge, this work is the first to address the connected k-overage problem under the probabilistic communication model as well as probabilistic sensing model. Moreover, we take the PA of each sensor into account to activate nodes for energy-consuming balance.

On the other hand, the densely deployment is a common method to avoid blind areas of coverage. In dense sensor networks, the (α, θ, k)-CCS is reduced to a generalization of the connected minimum dominating problem [3]. So we propose a *distributed probabilistic coverage and connectivity maintenance algorithm* (DPCCM) for the node-weighted (α, θ, k)-*CCS* problem in dense WSN because the problem is NP-hard. DPCCM utilizes the properties of weighted ε-net to find "good positions", then expands (θ, k)-flower with α-connectivity by two parameters which can be calculated according to probabilistic model and the request, *i.e.* (α, θ, k). Comparing with the randomized k-coverage algorithm [7], DPCCM significantly maintain coverage and connectivity in probabilistic model and prolong the network lifetime.

The remainder of the paper is organized as follows: Section II reviews the related work in the field. Section III introduces some necessary notations and preliminaries, including the formalization of (α, θ, k)-*CCS* and our approach. The pseudo code and the analysis of the proposed DPCCM are presented in Sections IV. Section V presents the simulation results. The paper concludes in Section VI.

2 Related Work

Because of its importance, the connected k-coverage problem has received significant research attention. Several protocols have been proposed in the literatures. Some of them assume the deterministic models and others assume the probabilistic models.

To the deterministic model, some protocols consider mainly coverage under the condition "the communication range is at least twice the sensing range" [7-9], and others study both coverage and connectivity [4,10,11]. Chakrabarty [8] formulates the k-coverage problem of a set of grid points as an integer linear programming. However, it is known well that localized algorithms (in which simple local node behavior achieves a desired global objective) may be necessary for sensor network coordination. Huang [9] presents a distributed node-scheduling algorithm to turn off redundant sensors. A node decides whether it is redundant only by checking the coverage state of its sensing perimeter. The authors in [7] propose an efficient approximation algorithm to achieve k-coverage in dense sensor networks. They model the problem as a set system for which an optimal hitting set corresponds to an optimal solution for k-coverage. For the connected k-coverage problem, Zhou [10] presents a distributed algorithm, DPA, which works by pruning unnecessary nodes. Wu [11] proposes several local algorithms to construct a k-connected k-dominating set. Yang [4] also presents two distributed algorithms. The first one uses a cluster-based approach to select backbone nodes to form the active node set. The second uses the pruning algorithm based on only 2-hop neighborhood information.

To the probabilistic model, new challenges are introduced in connected k-coverage protocols in sensor networks though they are more realistic. It is the first to address the k-coverage problem under the probabilistic sensing model in [12]. The authors address the problem to activate sensors one by one in a greedy fashion, in which the "contribution" or the "coverage merit" is computed based on the probability of detection of an event by that sensor within its sensing area. The authors in [6] propose a new probabilistic coverage protocol that is fairly general and can be used with different sensing models. However, how to maintain network connectivity is not considered. Hefeeda [13] designs a distributed probabilistic connectivity maintenance protocol that can employ different probabilistic models.

The closest works to ours are [6], [7] and [13]. Unlike DPCCM, node weight, *i.e.* power available is not considered. The algorithms for unweighted case can not be directly applied in weighted one. In addition, associating PCP [6] with PCMP [13] will provide probabilistic coverage and connectivity at the same time, but it only ensures (with probability at least required parameter) that each point in the target area is monitored by at least one sensor. In other words, it is only an approximate algorithm for $(\alpha,\theta,1)$-*CCS*. Therefore, DPCCM for (α,θ,k)-*CCS* is more general.

3 The Node-Weighted (α,θ,k)-CCS Problem and Our Approach

In this section, we formulate the α-connected (θ,k)-coverage set ((α,θ,k)-*CCS*) problem in WSN. Then an overview of our solution is stated with the assumption that sensors are deployed densely and have the same sensing radius R_s and communication radius R_c. Furthermore, localization and time synchronization have been finished, which can be done by many efficient schemes [14,15].

To study connectivity under probabilistic communication model, we represent the network with an undirected weighted simple graph $G=(V,E,c)$ called *communication graph*, where c is a communication probability function $c: V\times V\rightarrow[0,1]$. There exists an edge (i,j) if the distance between i and j is not more than R_c. For an edge (i,j), $c(i,j)$ is the probability of communication between i and j. For a path $p: i\rightarrow j$, $c(i,j)$,

called $c(p)$, equals to $\prod c(e)$, where e is an arbitrary edge in p. Thus, for arbitrary i, $j \in V$, $c(i,j)$ is $1-\prod(1-c(p))$, where p is an arbitrary path between i and j in graph G.

To study coverage under probabilistic sensing model, a sensing probability function $s: V \times T \to [0,1]$ is defined, where T is the target set: if the distance between i and j is not more than R_s, i senses j with probability $s(i,j)$, otherwise $s(i,j)=0$.

Definition 1 (α-Connectivity and (θ,k)-Coverage). Given a communication graph $G=(V,E,c)$ and a sensing probability function s as above, and the target set T. G is said to have α-connectivity if $c(i,j) \geq \alpha$ for arbitrary $i, j \in V$, where $0<\alpha<1$. G is said to have (θ,k)-coverage on T if each element of T is sensed by at least k nodes in V with probability at least θ, where $0<\theta<1$.

Then the α-connected (θ,k)-coverage set problem is formally stated as follows.

Problem 1 (α-Connected (θ,k)-Coverage Set problem, (α,θ,k)-CCS). Given a communication graph $G=(V,E,c)$, a target set T, $0<\alpha<1$, $0<\theta<1$, $k \in \mathbb{N}^*$. Is there a minimum subset V^* of V whose induced subgraph $G[V^*]$ has α-connectivity and (θ,k)-coverage on T.

The above (α,θ,k)-*CCS* is NP-hard, because (1,1,1)-*CCS*, *i.e.* connected cover set problem, as a special case of (α,θ,k)-*CCS* is NP-hard [16].

PA is considered also by node weight in this paper. Generally speaking, the less its PA is, the larger its weight will be, and the smaller the activated probability will be. In this paper, as an example, the weight of node i is defined as $W(i)=\left\lceil a \bullet 2^{-b\left\lceil \frac{PA(i)}{\lambda} \right\rceil} \right\rceil$, where W reflects the relation between PA and activated probability, and the (a,b,λ) are constant parameters based on the type of sensor and environment.

When the target value is continuous variable and deployed sensors are sufficiently dense, area coverage can be approximated by point coverage [4]. That is, it is feasible to select a subset of sensors to cover the rest of sensors. Even now, the problem is still NP-hard because it is reduced to the minimum dominating set problem. We present an approach in dense sensor networks: first find out "good positions" where only a few nodes can cover as many nodes as possible, then activate sensors with small weight around good positions to achieve (θ,k)-coverage and α-connectivity.

The "good positions" problem can be stated that given some weighted disks with the same radius, how to select disks with the minimum total weight to cover all the centers, as shown in Figure 1. In order to reduce the total weight of the nodes around good positions, the weight of disk is defined as follows. According to the theory of geometric disk cover, we adopt a method based on *VC*-dimension and ε-net to find out "good positions". Differing from [7], the weight of node is considered.

Let $\omega: 2^V \to \mathbb{R}$, where $\omega(\varnothing)=0$, $\omega(i)=\left\lfloor \frac{W(i)+\sum_{j\in N(i)} W(j)}{|N(i)|+1} \right\rfloor$, $N(i)=\{j:(i,j)\in E\}$ for $i \in V$ and $\omega(V')=\sum_{i\in V'} \omega(i)$ for $V' \subseteq V$. Let $F \subseteq 2^V$, $|F|=|V|$. Each of F states the nodes covered by the center. Together, the trine (V,F,ω) is a weighted set system.

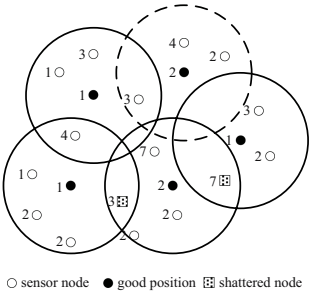


Fig. 1. An example of "good positions" and set shattering: the points are good positions, and the foursquare points are shattered by the four disks with fatter borderlines

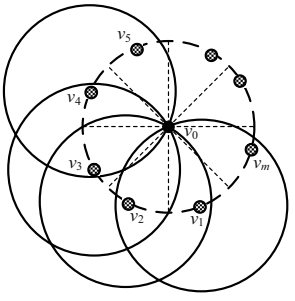


Fig. 2. An example of (θ, k)-flower: v_i senses v_0 with the same probability $p(r)$

Definition 2 (Weighted ε-Net). Given a weighted set system (V, F, ω), $N \subseteq F$ is called a weighted ε-net for (V,F, ω) if $N \cap L \neq \varnothing$ for all $L \in F$ with $\omega(L) \geq \varepsilon\, \omega(V)$.

The weighted ε-net is more general than uniform ε-net, because it is a special case of the former if $\omega(L) = |L|$. We interest in finding small weighted ε-nets. Typically, ε-net finder algorithms are designed for the uniform case. Thus we reduce the weighted case to unweighted one by taking $\lfloor \omega(v) + 1 \rfloor$ copies for $v \in V$, as outlined by [17]. In this paper, we adopt randomly selecting strategy to find ε-nets due to limited computing power and storage space of sensors.

The VC-dimension quantifies how "well behaved" of a set system. VC-dim- $\psi\psi$- is the size of the largest subset of $\psi\psi$that is shattered by $\psi\psi\psi$Figure 1 shows an example of the concept of shattering. The authors in [7] prove the set system composed of the set of points in R^2 and all disks with the same radius for each point has a VC-dimension of 3. From Corollary 3.8 in [- 8], a distributed weighted ε-net finder is designed by randomly selecting biased based on the weight in this paper.

After finding out a weighted ε-net, we simply verify whether it can hit all disks. If it can not, we try another weighted ε-net by *the modified doubling process*. The main idea is to put another weight $\psi(i)$ (initially uniformly) on the node i, and let $\zeta(i) = a2^{-b\omega(i)} + \psi(i)$, where $a2^{-b\omega(i)}$ reflects the relation between node weight and the probability of being the node of a weighted ε-net. If a weighted ε-net doesn't hit some element L of F, we double $\psi(i)$ for all i in L. Then find another weighted ε-net.

With the concept of weighted ε-net, the repeat can be proved only finite times before finding out a hitting set.

Lemma 1. Given a weighted set system (V, F, ω). If there is a hitting set of size c, the modified doubling process as above iterate not more than $6c \log[(n-c)/cq\psi_0 - 1]$ times for $\varepsilon = 1/(2c)$, where $n = |V|$ and $q = \min\{\omega(i): i \in V\}$ and ψ_0 is initially ψ.

Proof. Let H be a hitting set of size c. Let L_j be the element of F that doesn't be hit by a weighted ε-net at the j th iteration, and the weight with subscript j be the one after j iterations. From Definition 2 $\zeta_{j-1}(L_j) < \varepsilon\zeta_{j-1}(V)$. So $\zeta_j(V) = \zeta_j(L_j) + \zeta_j(V - L_j) = \zeta_{j-1}(V) + \psi_{j-1}(L_j)$. Because $\omega(i) > 0$, we have $\zeta_j(V) < \zeta_{j-1}(V) + \zeta_{j-1}(L_j) < (1+\varepsilon)\zeta_{j-1}(V) < (1+\varepsilon)^j \zeta_0(V) < \zeta_0(V)\, e^{j/2c}$. Moreover, since $H \cap L_j \neq \varnothing$, there is at least one node in H whose

$\psi(i)$ has been doubled. That is, if each $h \in H$ has been doubled $d(h)$ times, then $\sum d(h) \geq j$. We have $\psi_j(H) = \psi_0 \sum 2^{d(h)} \geq c\psi_0 2^{j/c}$. Note that $\psi_j(H) < \zeta_j(V)$, we conclude that $c\psi_0 e^{2j/3c} < c\psi_0 2^{j/c} < \zeta_0(V) e^{j/2c} < [c\psi_0 + (n-c)/q] e^{j/2c}$ from which the proof follows. □

The following is how to achieve (θ, k)-coverage and α-connectivity. The authors in [7] introduce the concept of *k-flower* to guarantee the coverage. Similarly, in order to select *(θ, k)-flower* which is a set of k sensors that all intersect at the center point with probability p, our approach is to choose m center nodes with minimal weight at distance r ($r < R_s$) at m sectors $[2\pi i/m, 2\pi(i+1)/m]$ for $0 \leq i \leq m-1$. Note that the m sensors sense the center with the same probability $p(r)$ for given r. Differing from [7], the m and r are alterable parameters, and how to choose is shown as follows.

Theorem 1. Given r and $p(r)$. Selecting $m_{\min} = \min\{m: mp(r)/(mp(r)-k)^2 \leq 1-\theta$ and $m \geq k\}$ nodes as the above strategy yield a (θ, k)-flower.

Proof. Assume that $\{v_1, v_2, \ldots, v_m\}$ is a (θ, k)-flower with radius r, whose center is v_0, as shown in Figure 2. Let X_i be independent random variable attaining the value 1 when v_i senses v_0 and otherwise the value 0. Let $X = \sum X_i$, then $E[X] = mp(r)$, $\sigma^2 = mp(r)$. We have $P[X \geq k] = 1 - P[X < k] = 1 - P[X - mp(r) < k - mp(r)] \geq 1 - P[|X - E[X]| \geq mp(r) - k] = 1 - P[|X - E[X]| \geq (\sigma - k/\sigma)\sigma] \geq 1 - mp(r)/(mp(r)-k)^2$. The last inequality is derived from a consequence of Chebyshev's inequality that states $P[|X - E[X]| \geq k\sigma] \leq 1/k^2$.

According to the definition of (θ, k)-flower, we have $1 - mp(r)/(mp(r)-k)^2 \geq \theta$. □

For example, let $\theta = 0.8$, $k = 10$ and $p(r) = 0.6$, we find $m_{\min} = 34$. For convenience the following m means $m_{\min}$.

The guarantee of α-connectivity is shown as follows.

Lemma 2. Given a triangular mesh grid. If any pair of neighbors can directly communicate with probability at least $\max\{\alpha, 1/[1+(1-\alpha)^{0.5}]\}$, the triangular mesh has a α-connectivity.

Proof. Statements proven by math induction. First, the proposition is true if $|V| = 3$. Assume it is also true when $|V| = K$ for $K \geq 3$. Let G' with $|V(G')| = K + 1$ is a triangular mesh expanded from G, as shown in Figure 3. By Definition 1, we have $c(i, j) \geq \alpha$ for arbitrary $j \in V(G)$. There exist two $i \to j$ paths, where $c(x,i) \geq \alpha$ and $c(y,i) \geq \alpha$ by the assumption, so we have $c(i, j) = 1 - [1 - c(j,x)c(x,i)][1 - c(j,y)c(y,i)] \geq \alpha$. □

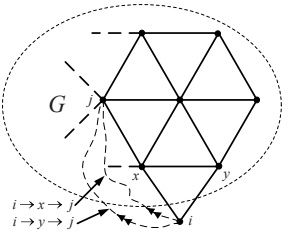


Fig. 3. The triangular mesh expansion

From Theorem 1 and Lemma 2, a sensor can gain appropriate r and m according to its probability model and given (α, θ, k). And the (θ, k)-flower based on $\min\{r, R_s\}$ (for convenience, still named r) and m has α-connectivity and (θ, k)-coverage.

In the following and the simulations, we leave out the communication link with probability less than α for the communication efficiency. This is because that information exchanges between neighbors are very frequent in practical application. Any pair of neighbors should firstly attempt to communicate directly.

4 Distributed Algorithm for Node-Weighted (α,θ,k)-CCS

In the previous section, we propose an approach to node-weighted (α,θ, k)-*CCS*, where the cost of computation is a little, and activating nodes does not rely heavily on global information. Therefore, a distributed algorithm called DPCCM for node-weighted (α,θ,k)-*CCS* is proposed. Its pseudo code is shown in Figure 4.

DPCCM SENDER
(1) *Initialize parameters*
ψ=1; *state*=TEMP; *coverage*=0; *netSize*=1; T=−1; calculate the W; broadcast W to neighbors and wait for B time units; calculate the ω, ζ, r, m; $totalWeight = n*\zeta$;
(2) *Find out "good positions":*
while ($netSize < n$) {
  if ($state ==$ TEMP and $netSize \times \zeta / totalWeight >$ rand()) {
    $state =$ ACTIVE;
    broadcast OK message containing location to neighbors; break; }
  wait for a constant S time units;
  if ($state ==$ TEMP and $1/(n - netSize) >$ rand()) {
    $\psi = 2\psi$; $totalWeight = totalWeight + totalWeight/n$; calculate ζ; }
  $netSize = 2*netSize$; }
(3) *Verify the coverage and form (θ,k)-flower with α-connectivity:*
while (true) {
  if ($state ==$ ACTIVE) {
    broadcast VERIFY message containing location to neighbors;
    wait for YES message; % for a constant R time units
    if ($coverage >= m$) {break;}
    if ($coverage < m$) {
      broadcast FLOWER message containing *coverage* and location
      to neighbors; $coverage = m$; break; } }

DPCCM RECEIVER
if ($msg.type ==$ OK and $state ==$ TEMP and
  space($msg.source$) $< \min\{R_c, R_s\}$) {break;}
if ($msg.type ==$ VERIFY and space($msg.source$) $< r$) {
  $coverage = coverage + 1$;
  if ($state ==$ ACTIVE) {send YES message to $msg.source$;}
  if ($state ==$ TEMP & $coverage >= m$) { $state ==$ SLEEP;} }
if ($msg.type ==$ YES and $state ==$ ACTIVE) { $coverage = coverage + 1$;}
if ($msg.type ==$ FLOWER and $state ==$ TEMP and space($msg.source$) $< r$) {
  calculate back-off timer T ; % construction steps as follows
  if ($T > 0$) { wait until $T == 0$; $state =$ ACTIVE;
    broadcast TURNOFF message containing l to its neighbors; } }
if ($msg.type ==$ TURNOFF and $T > 0$ and $l == l.source$) { $T = -1$;}

Fig. 4. A distributed algorithm for the node weighted *(α,θ,k)-CCS* (DPCCM)

DPCCM works upon receiving a task from the base station. In initialization, node i calculates and broadcasts $W(i)$ to its neighbors. After B time units $\omega(i)$ is calculated, where B is chosen beforehand to receive W of neighbors. Note that the weighted ε-net finder as above randomly selects nodes biased based on the weight. In order to locally estimate the total initial weight, we regard $\zeta(i)$ as the average weight. The estimation is practical since $\omega(i)$ is average and $\psi(i)$ is the same initially. Finally, it calculates the r and m based on Lemma 2 and Theorem 1.

In the process of finding out the "good positions", DPCCM works in rounds of equal S time units, where S is chosen beforehand according to the environment and the task requirement. In each round, some nodes switch randomly to be in ACTIVE state biased based on the weight $\zeta(i)$, and others uniformly double $\psi(i)$ with probability $1/(n-netSize)$. The reason is an under-covered node double $\psi(i)$ in the modified doubling process, the number of which is less than $n-netSize$. From the proof of Lemma 1, it is feasible to estimate the number by $n-netSize$ because it only increases iteration times. Every node with ACTIVE state broadcasts an OK message to its neighbors. When a neighbor is covered by an active node, it breaks the process of finding "good positions". At the end of each round, double *netSize*.

After $S\cdot\lceil \log n \rceil$ time units, every node with ACTIVE state begins to verify its coverage and connectivity. It broadcasts a VERIFY message containing location to neighbors and waits for R time units, where R is sufficient to reduce collision and guarantee that all neighbors can finish the response work. When a node receives a VERIFY message, it firstly compares r with the distance between itself and the message source. If the distance is more than r it rejects the VERIFY message, or else it checks its state. If its *state* = ACTIVE, it replies a YES message to the message source, otherwise it self-increases *coverage* and judges whether its *coverage* reaches k. As soon as a node achieves (θ, k)-coverage, it will change to be in SLEEP state. In R time units, *coverage* of node i self-increase every time receiving a YES message. If its *coverage* is less than k, it will activate some neighbors by broadcasting a FLOWER message to gain (θ, k)-coverage.

The FLOWER message contains location of source and its *coverage*. We prove that, as shown in Theorem 1 and Lemma 2, the centre node of (θ, k)-flower with α-connectivity has at least m active neighbors within radius r. In order to form (θ, k)-flower with α-connectivity, DPCCM chooses nodes with the minimum weight at distance r at $M(i)$ sectors $[2\pi l/M(i), 2\pi(l+1)/M(i)]$ for $0 \le l \le M(i)-1$, where $M(i) = m-coverage$. To make this decision locally, a back-off timer is adopted. The back-off timer $T(j)$ of the receiver j is determined according to its location, $r(i)$ and $M(i)$. The following steps are used in turn to decide $T(j)$: (1) if $d(i,j) < r(i)-\delta$, let $T(j) = -1$, where δ is a small positive constant; (2) if j is in the sector $[2\pi l/M(i), 2\pi(l+1)/M(i)]$, let $T(j) = l\cdot C + C\cdot W(j)/E$, where C is a constant and $E > W(j)$ for arbitrary j. When a sensor times out, the sensor changes its *state* = ACTIVE and broadcasts a TURNOFF message containing l to its neighbors. When a sensor receives a TURNOFF message before the timer expires, it compares l with own: if they are the same, it lets $T(j) = -1$, or else rejects the message.

The algorithm terminates when all sensors are in ACTIVE state or SLEEP state.

As the following, some analyses of DPCCM are shown. First, we prove the correctness of the proposed DPCCM.

Theorem 2. Given a node-weighted sensor network G as above. The active sensors chosen by DPCCM can (θ, k)-cover all nodes in G and have α-connectivity.

Proof: Firstly, we show that the active sensors are guaranteed to hit every sensing disk. In the process of finding out the "good positions", node i doubles $\psi(i)$ with probability $1/(n-netSize)$ until it is activate node or the neighbor of an activated one. Doubling $\psi(i)$ increases the probability to be activated. From Lemma 1, the active node set is a hitting set, otherwise node density is not enough to achieve a hitting set. Then DPCCM activates some nodes to guarantee every active node has m active neighbors within less than radius r. Since the algorithm terminates when all sensors are in ACTIVE state or SLEEP state, both of them satisfy the condition $coverage \geq m$. According to Theorem 1 and Lemma 2, the (θ, k)-flower based on the above r and m has α-connectivity and (θ, k)-coverage. □

The next theorem provides time complexity of DPCCM. We carry out our analysis in terms of the input parameters B, C, R and S, which are discussed in Figure 4. We assume that a message transferred between two neighbors takes one time unit, and so does continuous local computation. And we reduce the communication collision by waiting for some time.

Theorem 3. DPCCM terminates in at most $(m \cdot C + R + 5)n + S \cdot \lceil \log n \rceil + B + 2$, *i.e.* $O(n)$ time units, where m is determined by (α, θ, k) based on Lemma 2 and Theorem 1.

Proof. According to hypothesis, every node completes the initialization in $B+2$ time units. In the process of finding "good positions", the algorithm iterates for $\lceil \log n \rceil$ steps. Since each iteration works in rounds of S time units, the algorithm costs $S \cdot \lceil \log n \rceil$ time units. Within the following processes till the termination of algorithm, there exist three types of state change: ACTIVE→break, TEMP→ACTIVE→break, and TEMP →SLEEP. To the first, an active node broadcasts a VERIFY message to its neighbors and waits for R time units. Within R time units, either a TEMP node increases its coverage or an ACTIVE node replies a YES message. So it costs R+3 time units. To the second, it is certain that the node receives a FLOWER message from an active node. After it receives the message it will wait for T time units to be activated. From the construction of T, we have $T \leq C(l+1) \leq CM \leq Cm$. So it costs at most $mC+2$ time units. Then it costs $R+3$ time units from ACTIVE to break. To the last, the node receives at least k VERIFY messages and it costs k time units. On the other hand, the above three types of state change are repeated continuously until the algorithm terminates. Since every node is corresponding to one type and only, the total time is $B+2+S \cdot \lceil \log n \rceil + (R+3)\tau_1 + (mC + R + 5)\tau_2 + k\tau_3$, where τ_i is the number of the i th case as above. Note that $m \geq k$ and $\tau_1 + \tau_2 + \tau_3 = n$, the proof can be concluded. □

In the following theorem, we provide the communication complexity of DPCCM algorithm.

Theorem 4. The number of messages broadcasted or sent in the DPCCM is at most $6n$, *i.e.* $O(n)$.

Proof. In the initialization, every node broadcasts its W to its neighbors, so the number of messages is n. In the process of finding "good positions", every active node broadcasts an OK message. The number of active node is less than n, so is the number of OK messages. From the following processes till the termination of algorithm,

analysis is similar to the proof of Theorem 3. One "ACTIVE→break" node broadcasts at most two messages: VERIFY and FLOWER. And one "TEMP →ACTIVE→break" node broadcasts at most three messages: TURNOFF, VERIFY and FLOWER. And one "TEMP→SLEEP" node does not broadcast. On the other hand, the YES messages are only sent by active but not break nodes, the number of which is less than n. So the proof can be concluded. □

The approximation factor of DPCCM algorithm is underway. Because it is difficult to analyze theoretically the approximation factor, we test experimentally the performance of DPCCM. The result is shown in Fig.7. In fact, it is exactly our aim to prolong the network lifetime on the requirement of connectivity and coverage. So the experiment result can support the performance of DPCCM in some sense.

5 Simulation

This section presents results from our simulation. The proposed DPCCM algorithm and the randomized k-coverage algorithm, named DRKC [7], were simulated in Prowler, a probabilistic sensor network simulator [19]. To assure the network is dense without coverage hole, 300 sensors are deployed as a grid of points and 300 sensors are randomly placed in a restricted 10×10 area. Some simulation parameters are shown here: sensing range is 1 and so is communication, the initial energy of each sensor is 5000, transmission, reception and idle are 5, 1, and 1, completeness a task consumes 300. The sensing mode is adapted from the exponential model, and the communication model is set to the log-normal shadowing model. The evaluation metrics include the percentage of active sensors with various (α, θ, k), coverage, and the network lifetime.

First we analyze the percentage of active sensors when (α, θ, k) is changed. We vary the requested coverage k between 1 and 8, sensing probability between 50% and 90%, connected probability between 50% and 90%. When one of them is varied, all other parameters are fixed, as shown in Figure 5. The Figure 5(a) and 5(c) shows that the percentage of active sensors increases fast while required sensing probability or required coverage increase. We think that it may be caused by r and m from Theorem 1 and Lemma 2, especially m increases rapidly. However, as is shown in Figure 5(b), the percentage increases slowly with higher connected probability. The result indicates that a triangle mesh is sufficient to ensure probabilistic connectivity.

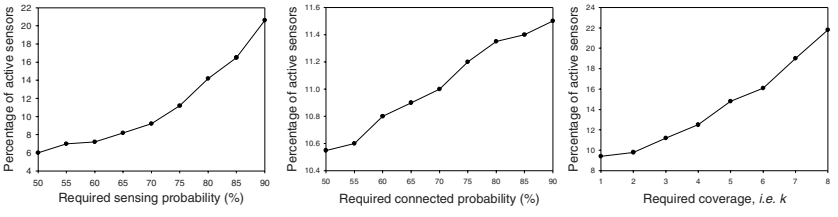


Fig. 5. Analyzing effect on percentage of active sensors when (α, θ, k) is changed: (a) vary $\alpha \in [0.5,0.9]$ while fix $\theta = 0.75$, $k = 3$; (b) vary $\theta \in [0.5,0.9]$ while fix $\alpha = 0.75$, $k = 3$; (c) vary $k \in [1,8]$ while fix $\alpha = 0.75$, $\theta = 0.75$

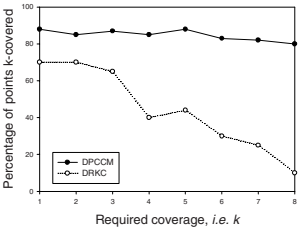


Fig. 6. Comparing the percentage of points k-covered with DRKC

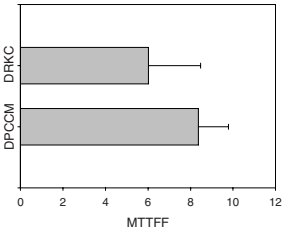


Fig. 7. The MTTFF under DPCCM and DRKC

The coverage by DPCCM and DRKC are compared. The achieved coverage at some random sampling points in the target area has been collected statistically. We fix $\theta = 0.70$ and $\alpha = 0.80$. As shown in Figure 6, DPCCM is significantly better than DRKC. This is important because, under the probabilistic sensing model, the active sensors chosen for deterministic model are not k-covered really.

Finally, we study the MTTFF (*mean time to first failure*) of any given task under DPCCM and DRKC, which can indicate the network lifetime in some application. We randomly gain k from 1 to 8 in each task. As soon as a task can not been completed, we calculate the MTTFF in this experiment. After repeating 50 times, the result is shown in Figure 7. Compared with DRKC the MTTFF under DPCCM has been prolonged about 21%. This is because that DRKC activate as few sensors as possible based on required coverage and isn't always optimal in a series of tasks because of unbalanced energy-consuming among nodes.

6 Conclusion

In this paper, we consider connected k-coverage problems under probabilistic sensing models and probabilistic communication models, which are more realistic than deterministic models. We represent the problems with the α-connected (θ,k)-coverage set problem and formulize it as (α,θ,k)-*CCS*. Moreover, in order to satisfy various coverages and realize energy-consuming balance, we also take power available into consideration with node weight. Because node-weighted (α,θ, k)-*CCS* is NP-hard, a distributed approximate algorithm, named DPCCM, is proposed for dense sensor networks. DPCCM utilizes the properties of VC-dimension and weighted ε-net to find "good positions", then expands (θ, k)-flower with α-connectivity by r and m. The two parameters can be calculated according to the tasks and performance indexes of sensors. We prove the correctness of DPCCM and theoretically analyze the time complexity and communication complexity. We also implement our algorithm in Prowler and compare it against the randomized k-coverage algorithms.

Acknowledgments. This research was supported by the National Natural Science Foundation of China under Grant Nos.60603062 and the Natural Science Foundation of Hu'nan Province of China under Grant No.06JJ3035.

References

1. Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next century challenges: Scalable coordination in sensor networks. In: 5th ACM International Conference on Mobile Computing and Networking (MOBICOM 1999), pp. 263–270. ACM Press, Seattle (1999)
2. Cerpa, A., Elson, J., Hamilton, M., Zhao, J., Estrin, D., Girod, L.: Habitat monitoring: Application driver for wireless communications technology. In: Proc. of ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, pp. 3–5. ACM Press, Costa Rica (2001)
3. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: A survey. Computer Networks 38(4), 393–422 (2002)
4. Yang, S., Dai, F., Cardei, M., Wu, J., Patterson, F.: On connected multiple point coverage in wireless sensor networks. Journal of Wireless Information Networks 13(4), 289–301 (2006)
5. Hekmat, R., Van Mieghem, P.: Connectivity in wireless ad-hoc networks with a log-normal radio model. Mobile Networks and Applications 11(3), 351–360 (2006)
6. Hefeeda, M., Ahmadi, H.: A probabilistic coverage protocol for wireless sensor networks. In: 15th IEEE International Conference on Network Protocols (ICNP 2007), pp. 41–50. IEEE Press, Beijing (2007)
7. Hefeeda, M., Bagheri, M.: Randomized k-coverage algorithms for dense sensor networks. In: 26th IEEE International Conference on Computer Communications (INFOCOM 2007), pp. 2376–2380. IEEE Press, Anchorage (2007)
8. Chakrabarty, K., Iyengar, S., Qi, H., Cho, E.: Grid coverage for surveillance and target location in distributed sensor networks. IEEE Transactions on Computers 51(12), 1448–1453 (2002)
9. Huang, C., Tseng, Y.: The coverage problem in a wireless sensor network. In: 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA 2003), pp. 115–121. ACM Press, San Diego (2003)
10. Zhou, Z., Das, S., Gupta, H.: Connected k-coverage problem in sensor networks. In: 13th International Conference on Computer Communications and Networks (ICCCN 2004), pp. 373–378. IEEE Press, Chicago (2004)
11. Wu, J., Dai, F.: On constructing k-connected k-dominating set in wireless networks. In: 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), pp. 81a–81a. IEEE Press, Denver (2005)
12. Lu, J., Bao, L., Suda, T.: Coverage-Aware Sensor Engagement in Dense Sensor Networks. In: Yang, L.T., Amamiya, M., Liu, Z., Guo, M., Rammig, F.J. (eds.) EUC 2005. LNCS, vol. 3824, pp. 639–650. Springer, Heidelberg (2005)
13. Hefeeda, M., Ahmadi, H.: Network connectivity under probabilistic communication models in wireless sensor networks. In: 4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2007), pp. 1–9. IEEE Press, Pisa (2007)
14. He, T., Huang, C., Lum, B., Stankovic, J., Adelzaher, T.: Range-free localization schemes for large scale sensor networks. In: 9th ACM International Conference on Mobile Computing and Networking (MOBICOM 2003), pp. 81–95. ACM Press, San Diego (2003)
15. Ganeriwal, S., Kumar, R., Srivastava, M.B.: Network-Wide Time Synchronization in Sensor Networks. NESL Technical Report (2003)
16. Gupta, H., Zhou, Z., Das, S.R., Gu, Q.: Connected sensor cover: Self-Organization of sensor networks for efficient query execution. In: 4th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2003), pp. 189–200. ACM Press, New York (2003)
17. Brönnimann, H., Goodrich, M.: Almost optimal set covers in finite VC-dimension. Discrete&Computational Geometry 14(4), 463–479 (1995)
18. Haussler, D., Welzl, E.: Epsilon-nets and simplex range queries. Discrete and Computational Geometry 2(1), 127–151 (1987)
19. Simulator can be downloaded from, http://www.isis.vanderbilt.edu/Projects/nest/prowler/

Optimal Surface Flattening

Danny Z. Chen[1,⋆] and Ewa Misiołek[2]

[1] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA
chen@cse.nd.edu

[2] Mathematics Department, Saint Mary's College, Notre Dame, IN 46556, USA
misiolek@saintmarys.edu

Abstract. The problem of optimal surface flattening in 3-D finds many applications in engineering and manufacturing. However, previous algorithms for this problem are all heuristics without any quality guarantee and the computational complexity of the problem was not well understood. In this paper, we prove that the optimal surface flattening problem is NP-hard. Further, we show that the problem admits a PTAS and can be solved by a $(1+\epsilon)$-approximation algorithm in $O(n \log n)$ time for any constant $\epsilon > 0$, where n is the input size of the problem.

1 Introduction

We consider the problem of "flattening" a surface S in $\mathbb{R}^3$. Surface flattening is an important problem that finds applications in computer graphics and surface reconstruction as well as in engineering and manufacturing [3] (particularly in aircraft, vehicle, or garment design). In the latter applications, the surface of a 3-D object must be assembled from a flat piece of material. To find the shape of the flat piece of material for the assembly of the surface, we start with a 3-D model of the surface S and search for an optimal way to flatten it by cutting S. The cutting must start from the boundary and continue toward the interior of S in such a way that (1) the total length of the cutting paths is minimized and (2) if the flat equivalent S' of S in $\mathbb{R}^2$ is glued back along the cuts, the reconstructed surface requires a minimum "stretch" to achieve the original shape of S. The amount of needed stretch, or the deviation of S around a point p from being flat, is measured by the *Gaussian curvature* at p. The Gaussian curvature at a point p of S is the product of the principal curvatures of S at p, i.e., roughly speaking, the product of the maximum and minimum curvatures at the point p among all the curves on the surface passing through p. The larger the absolute value of the Gaussian curvature at p is, the farther S is from being flat at p. Thus, to reduce the stretch, we cut the surface S along points at which the absolute values of the Gaussian curvatures are too large (e.g., with respect to a given threshold value). On the other hand, if the Gaussian curvatures at all points on S are 0, then the

⋆ The research of this author was supported in part by the National Science Foundation under Grant CCF-0515203.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 233–244, 2008.

surface is flat and requires no cuts to flatten it (and no stretch to "reconstruct" the original shape from the flat version).

From the application view point, the surface to be flattened is represented by a triangular mesh $S = (N, M)$, where N is a given set of n points and M is a set of $m = O(n)$ segments connecting the points in N. We assume that the Gaussian curvature at each of the n points is given. (Note that the computation of Gaussian curvature using a discrete representation of the surface is not possible, but it can be approximated, for example, by using a discrete method [9,12].) Each point at which the absolute value of the Gaussian curvature is greater than a given threshold value is required to lie on a cutting path. Clearly, we can use an undirected graph $G = (V, E)$ to represent the mesh surface $S = (N, M)$, by letting $V = N$ and $E = M$. We assume in this paper that such a graph G for S is planar and G admits a planar embedding such that the boundary of S corresponds to the border of the outer face of the planar embedding of G.

Given this setting, the optimal surface flattening problem is defined as follows.

Surface Flattening Problem (SF). *Given a triangular mesh $S = (N, M)$ representing a surface in $\mathbb{R}^3$, $N_A \subseteq N$, which is the set of points with the absolute values of the Gaussian curvatures above a threshold value, and $N_B \subseteq N$, $N_B \neq \emptyset$, which is the set of points on the boundary of the surface S, find a set of cutting paths of the shortest total length along the edges of S that connect every point in N_A to at least one point in N_B.*

Without loss of generality, we assume $N_A \neq \emptyset$ (otherwise, the solution is trivial). Note that in the above definition of the SF problem, the essential requirement is that in the solution, there is a path connecting each point in N_A to some point in N_B. The paths for two different points in N_A can cross or (partially) overlap. Therefore, the union of the paths in a solution should form a certain subgraph in the graph G defined by (N, M), such that the total sum of the edge lengths in this subgraph is minimized.

Since surface flattening is a very important problem in a number of industrial applications, the literature on surface flattening is vast. However, to the best of our knowledge, all previous algorithms for this problem are heuristics without any quality guarantee of their solutions, and no thorough analysis of the computational complexity of the problem has been given. In most cases, the known algorithms seek to find a solution based on an application-specific metric. The various approaches for solving the problem include greedy algorithms that search for seemingly best possible insertions of cuts or darts; for example, Parida and Mudur [11] found cuts during successive flattenings of the mesh triangles, and Aono *et al.* [1,2] inserted darts into a flat cloth to adjust the shape to the surface. Other methods include optimization of various energy functions [8,10,13] and, more recently, algorithms searching for individual shortest cuts passing through points with high absolute values of Gaussian curvatures [12,14]. Sheffer's algorithm [12] finds shortest paths that connect nodes with high absolute values of Gaussian curvatures with the nodes on the boundary of the surface using a minimal spanning tree. Unfortunately, this method does not work well for surfaces

with widely distributed curvatures. Wang *et al.* [14] avoided this problem by first computing a boundary geodesic map and then repeatedly finding a shortest path from a selected node to the surface boundary using this map. Neither of these algorithms guarantees an optimal quality solution. Azariadis *et al.* [3], recognizing the large number of optimization criteria and methods for surface flattening, considered the problem of quality control from the view point of applications. They evaluated many existing surface flattening methods by using "intuitively-acceptable" visualization techniques (they also gave a brief overview of the many surface flattening algorithms). To the best of our knowledge, there are no previous results providing theoretical analysis of the computational complexity of the problem or quality guarantee of the solutions.

In this paper, we prove that the optimal surface flattening problem as specified in [12,14], i.e., the problem of computing cutting paths of the minimum total length along the mesh edges that pass through points with high absolute values of Gaussian curvatures, is NP-hard. This implies that finding an optimal solution for the problem in deterministic polynomial time is unlikely unless P = NP. Furthermore, we show that the surface flattening problem admits a PTAS (polynomial-time approximation scheme), that is, it can be reduced to the problem of computing an optimal Steiner tree on a planar graph, and, as such, can be solved by a $(1+\epsilon)$-approximation algorithm due to Borradaile, Klein, and Mathieu [5] in $O(n \log n)$ time, where $\epsilon > 0$ is any constant. This appears to be a theoretically "best possible" solution for the problem unless P = NP.

2 The NP-Hardness of the Optimal Surface Flattening Problem

In order to show that the optimal SF problem is NP-hard, we first prove the NP-completeness of the following decision version of the problem.

Surface Flattening Decision Problem (SF-D). *Given a triangular mesh $S = (N, M)$ representing a surface in $\mathbb{R}^3$, $N_A \subseteq N$ which is a set of points on S with the absolute values of the Gaussian curvatures above a threshold value, $N_B \subseteq N$ which is a set of points on the boundary of S, and an integer k, does there exist a set of cutting paths along the edges of S that connect every point in N_A to at least one point in N_B whose total length is less than or equal to k?*

To show that the SF-D problem is NP-complete, we must show that (1) the problem is in NP, i.e., it can be solved in polynomial time by a nondeterministic Turing machine (or a given solution can be verified in polynomial time), and (2) the problem is NP-hard (a known NP-complete problem can be transformed in polynomial time to the SF-D problem). Since verifying the first condition is easy, we omit it here. The second part will involve a two-step reduction from a well-known NP-complete problem, the Rectilinear Steiner Tree Problem [6].

Given a set A of points in the plane, a *rectilinear Steiner tree* (RST) for A is a tree connecting all points in A using line segments that are either horizontal or vertical. As opposed to a *spanning tree*, the nodes of an RST (i.e., the endpoints

of the segments of the RST) may include some *Steiner points*, that is, points in the plane that do not belong to A but are needed for the desired connection of the RST. A minimum RST is a tree whose total length of line segments is minimized. The RST problem, stated as follows, has been proved to be NP-complete [6].

Rectilinear Steiner Tree Problem (RSTP). *Given a finite set A of points in the plane and an integer $k > 0$, does there exist a rectilinear Steiner tree for A with a total length no bigger than k?*

In fact, even the special case of the RST problem in which the coordinates of the points in A are all integers was shown to be NP-complete in [6]. In the rest of this paper, whenever we refer to any variations of the (geometric) RST problem, we assume that the coordinates of the points in A are all integers.

For our proof, we start with transforming the RSTP to a slightly different problem, called the *boxed* rectilinear Steiner tree problem (or the boxed RSTP), and proving that the boxed RSTP is also NP-complete. In the boxed RSTP, we choose a certain "box" B_A containing all the points in the set A (as shown below), and require that the sought rectilinear Steiner tree connect all the points of A as well as *at least one point* on the box B_A. Note that for the sought rectilinear Steiner tree to connect all the points in A and at least one point of the box B_A, it is sufficient to consider only a finite set of points on the box B_A, denoted by B. The box B_A and the point set B on B_A are constructed as follows. Create a rectilinear grid R_A using the vertical and horizontal lines on the plane passing through all the points in A. Note that the coordinates of all vertices of R_A are integers. Let $d = 1 + \max\{d_h, d_v\}$, where d_h is the distance between the leftmost and the rightmost vertical lines of the grid R_A, $x = x_l$ and $x = x_r$, and d_v is the distance between the top and the bottom horizontal lines of R_A, $y = y_t$ and $y = y_b$. Clearly, d is an integer value. Further, note that the rectilinear (or L_1) distance between any two points in A is strictly less than $2d$. The box B_A is defined by the two vertical lines $x = x_1 = x_l - 2d$ and $x = x_2 = x_r + 2d$ and the two horizontal lines $y = y_1 = y_b - 2d$ and $y = y_2 = y_t + 2d$. That is, B_A consists of four line segments $((x_1, y_1), (x_2, y_1))$, $((x_2, y_1), (x_2, y_2))$, $((x_1, y_2), (x_2, y_2))$, and $((x_1, y_1), (x_1, y_2))$. The points in the set B are those on the intersection of the grid R_A and the four segments of B_A. See Fig. 1 for an example. Let R_B denote the finite portion of the grid R_A contained inside the box B_A, together with the box B_A as its boundary. Clearly, the coordinates of all vertices of R_B are also integers; furthermore, the lengths of edges of R_B are all integers as well.

Boxed Rectilinear Steiner Tree Problem (BRSTP). *Given a finite set A of points in the plane and an integer $k > 0$, and suppose the box B_A containing A and the point set B on B_A have been determined as above, does there exist a rectilinear Steiner tree for A that also includes at least one point in B with a total length less than or equal to k?*

Note that we can restrict a rectilinear Steiner tree T, as a solution to RSTP or BRSTP, to the grid R_A or R_B in the sense that any Steiner point of the tree T not on a grid vertex can be moved to a grid vertex without changing the

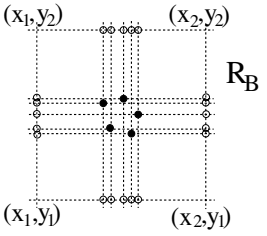


Fig. 1. The points in A are marked as solid dots, and the points in B are marked as empty dots

total length of the resulting rectilinear Steiner tree T'. We have the following statement on A and the grid R_B whose vertices all have integer coordinates only.

Lemma 1. *The boxed rectilinear Steiner tree problem (BRSTP) is NP-complete.*

Proof. Our transformation is from the rectilinear Steiner tree problem (RSTP) (with the points in A all having integer coordinates) to the boxed rectilinear Steiner tree problem (BRSTP). The ideas of the proof are not complicated but the details are a little tedious. Due to the space limit, the detailed proof is omitted here and can be found in the full version of the paper. □

We now show the NP-completeness of the surface flattening decision problem (SF-D) by using a transformation from the NP-complete boxed rectilinear Steiner tree problem on R_B whose vertices all have integer coordinates only. The instance of the SF-D problem produced by the transformation, however, may have nodes with non-integer coordinates.

Theorem 2. *The surface flattening decision problem (SF-D) is NP-complete.*

Proof. Our idea for the proof, based on a transformation from the integer version of BRSTP to SF-D, is fairly straightforward, yet the details of the transformation are a little tedious.

Using the grid R_B on the plane (i.e., the box B_A together with the portion of R_A enclosed in and bounded by B_A), we construct, in polynomial time, a triangular mesh surface $S = (N, M)$ in $\mathbb{R}^3$ such that the solutions to BRSTP on R_B and to SF-D on S are equivalent.

To define S in $\mathbb{R}^3$, we need to specify the x-, y-, and z-coordinates of all the points in N as well as the set M of segments (including their lengths) connecting the points of N. To begin, we include in S all vertices and segments of R_B, initially setting the z-coordinates of all these vertices to 0. This creates a rectangular mesh on the xy-plane (with $z = 0$). To obtain from this planar rectangular mesh a triangular mesh in $\mathbb{R}^3$, we add to the rectangular mesh more vertices and edges: Each of the newly added vertices lies at the intersection of the two diagonals of every cell of the rectangular mesh, and the new edges are the "diagonal" segments connecting each of these diagonal intersection points with its four cell vertices. We denote the set of points thus added by N_D ($N_D \subset N$). Also, we let

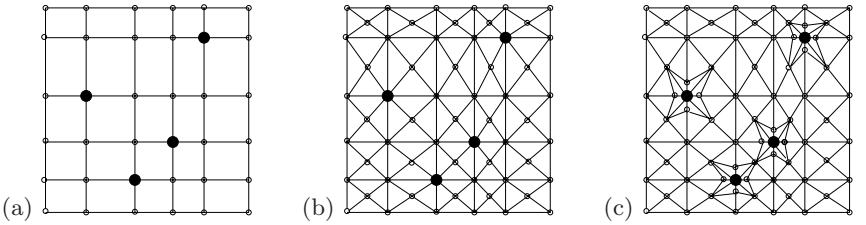


Fig. 2. Illustrating the steps for constructing the mesh surface S (S is vertically projected onto the xy-plane in these figures). (a) $S = R_B$ (the points of N_A are marked by solid dots). (b) The addition of points in N_D lying at the intersections of the diagonals of all the cells of R_B and the addition of the diagonal segments. (c) The addition of points in N_K and the addition of segments between the points in N_D and the neighboring points in N_K.

N_A and N_B denote the subsets of points in N that correspond to the points of A and B on R_B, respectively. Figure 2(a) shows the initial rectangular mesh of S ($= R_B$) in which the points of N_A are marked by large solid dots; Figure 2(b) shows the modified mesh with the points of N_D and the "diagonal" segments.

The next step is to elevate (i.e., increase the z-coordinates of) all points of N_A and N_D in $\mathbb{R}^3$. The necessity of elevating the points of N_A is obvious: To ensure their absolute Gaussian curvatures to be sufficiently large. We elevate the points of N_D to ensure that no points of N_D belong to any shortest path along the edges of S between any two points of S that correspond to two arbitrary vertices of the initial grid R_B (so that any "good" cutting path on S will follow only the edges of S that correspond to those of R_B). Without loss of generality, we assume that the length of the shortest edge of the original grid R_B is one unit. Let l be the length of the longest edge of R_B, and c be the number of edges of R_B. Then we set the z-coordinates of all points in N_D to be $2l + 1$ and the z-coordinates of all points in N_A to be $1/(9c)$.

Let g_0 be the absolute value of the Gaussian curvature at the intersection point p of the diagonals of a 1×1 square such that the z-coordinate of p is $2l + 1$ and all the vertices of the square are on the xy-plane. Since the length of the shortest edge of R_B is 1, the absolute value of the Gaussian curvature at any point of N_D is no bigger than g_0. We set the threshold curvature value for the SF-D problem on S to be g_0. To ensure a high absolute value of the Gaussian curvature at each point $p_a \in N_A$ on S, we build a "steep" structure around p_a on S, as follows. We create four points around every point $p_a \in N_A$, and denote the set of all the points thus created by N_K. Specifically, for each $p_a \in N_A$, we place points $p_i^a \in N_K$, $i = 1, 2, 3, 4$, on each of the four segments of the original rectangular mesh R_B adjacent to p_a, such that the z-coordinate of each p_i^a is 0 and p_i^a is at a distance d_0 from the vertical projection point of p_a on the xy-plane. The fixed value d_0 is chosen to be sufficiently small to satisfy the following two conditions: (1) $d_0 \leq 1/(9c)$, and (2) the absolute value of the Gaussian curvature at any $p_a \in N_A$ is larger than the threshold value g_0. (The value d_0 can be calculated from the fixed values g_0, l, and c.) The points in

N_K, all on the xy-plane, are added to the set N to make their corresponding points $p_a \in N_A$ locally "steep" (thus attaining high absolute values of Gaussian curvatures at p_a), and they split each edge of R_B adjacent to every $p_a \in N_A$ into two segments in M. Lastly, we add to M the segments connecting each point $p_d \in N_D$ with any point $p_i^a \in N_K$ lying on the boundary of the cell of R_B containing p_d.

This completes the construction of S. See Fig. 2(c) for an example of the resulting mesh surface S (as projected vertically onto the xy-plane). Clearly, the construction of S takes polynomial time since S consists of $O(|A|^2)$ vertices and $O(|A|^2)$ segments.

It remains to show that BRSTP has a solution T_{BP} of a total length less than or equal to k on R_B if and only if SF-D has a solution C_{SF} of a total length less than or equal to $k + 1/2$ on S, for any given integer $k > 0$.

Before we show the equivalence of these two solutions, recall that the reason for adding the points of N_D to N is to create a triangular mesh for the surface flattening problem. However, we need to prevent any "good" cutting paths of S from going through any points of N_D so that the cutting paths on S follow only those segments of S corresponding to the edges of R_B (as required by a BRSTP solution on R_B). By elevating the points of N_D high enough to make their adjacent segments longer than any of the edges of R_B and by making the heights of all the points of N_A very small, we make sure that no points of N_D belong to any shortest path along the segments of S connecting any two points in $N - N_D$. This is because a path between any two points in $N - N_D$ through a "diagonal peak point" $p_d \in N_D$ of any cell of R_B is longer than a path along the border of that cell. Thus, "good" cutting paths connecting points in $N - N_D$ and along the segments of S use only segments that correspond to the edges of R_B. Without loss of generality, we assume that any solution C_{SF} for the SF-D problem on S, that is, the union of a set of cutting paths for N_A on S, uses only segments of S that correspond to the edges of R_B (otherwise, we can always replace any cutting path in C_{SF} passing through a point in N_D by another cutting path using only the segments along the borders of the cells of R_B without increasing the total length of the resulting SF-D solution).

Now, suppose T_{BP} is a solution to BRSTP on R_B of a total length less than or equal to k. Note that since the lengths of all edges of R_B are integers, the total length $|T_{BP}|$ of T_{BP} must be an integer as well. Let C_{SF} consist of all the segments of S whose vertical projections onto the xy-plane lie entirely on the edges of T_{BP}. Then C_{SF} is a feasible solution to SF-D since every point in N_A is connected to a point in N_B along the segments of $S \cap C_{SF}$ (this follows from the fact that T_{BP} is an RST for A and connects to at least one point of B). Note that the segments of C_{SF} correspond to the edges of T_{BP} on R_B, plus connections to a little "tip" at each point $p_a \in N_A$. Every such little tip at $p_a \in N_A$ adds a total length less than $4 \times 1/(9c) < 1/(2c)$ to the total length of T_{BP} for the SF-D solution C_{SF}. Thus, a total length less than $|A| \times 1/(2c) \leq c \times 1/(2c) = 1/2$ due to the "tips" over all the points of N_A is added to the total length of T_{BP} for

the SF-D solution C_{SF}, implying that the total length $|C_{SF}|$ of C_{SF} is no bigger than $|T_{BP}| + 1/2 \leq k + 1/2$.

Conversely, suppose C_{SF} is a solution to SF-D on S of a total length less than or equal to $k + 1/2$. Let T_{BP} be the set of edges on R_B corresponding to the segments of C_{SF}. Since, unlike C_{SF}, T_{BP} does not contain the "tips" at the points of N_A, its total length $|T_{BP}|$ is strictly smaller than $|C_{SF}| \leq k + 1/2$. Further, since the total length of T_{BP} must be an integer, the largest possible integer value for $|T_{BP}|$ that is less than $k + 1/2$ is k. Besides, as argued in the previous paragraph, the total contribution of all the "tips" to the total length of C_{SF} is strictly less that $1/2$. Thus, $|T_{BP}| \leq k$ holds. If T_{BP} forms a tree, then we are done, since it contains all the points of A and at least one point of B. However, note that C_{SF} is the union of cutting paths along the segments of S corresponding to the edges of R_B, and such a cutting solution, although connecting each point of N_A to some point of N_B, need not form a tree structure. That is, T_{BP} may not be a desired RST for BRSTP on R_B. Actually, T_{BP} may consist of multiple connected components, and each such component may contain some cycles. Hence, our remaining task is to convert T_{BP} to an RST for A on R_B that touches at least one point in B, *without increasing* the total length of the resulting T_{BP}.

Observe that since C_{SF} is a feasible cutting solution to SF-D on S, every connected component of T_{BP} must contain at least one point of B and some points of A (those components containing no points of A can be safely removed from any further consideration). We first remove from T_{BP} all edges that lie entirely on the boundary of the box B_A (this removal does not yet remove the end vertices of those edges). If any point of $B \cap T_{BP}$ becomes an isolated vertex of T_{BP} after this edge-removal, then remove that point from T_{BP} as well. Clearly, this removal process does not affect the feasibility of the corresponding cutting solution to SF-D on S (i.e., if the corresponding segments and endpoints are removed from C_{SF}), and can only decrease the total length of T_{BP}. But, it can create more connected components of T_{BP}, each of which still contains at least one point of B (but, each such point of B is now of degree exactly one in T_{BP}). Note that for any two points of A, the rectilinear (or L_1) distance between them is strictly less than $2d$. We "merge" any two distinct connected components C_1 and C_2 of T_{BP} into one component, as follows: Remove from T_{BP} all the points of B and their adjacent edges that are contained in C_1 (this decreases the total length of T_{BP} by at least $2d$), and connect an arbitrary remaining vertex of C_1 with an arbitrary vertex of C_2 by a shortest path lying on R_B (this increases the total length of T_{BP} by less than $2d$). The resulting T_{BP}, with at least one less connected component, has a smaller total length than its previous version. Further, T_{BP} still contains all the points of A, and each of its connected components contains at least one point of B. We continue this merge process until T_{BP} has exactly one connected component. At this point, T_{BP} is a connected subgraph on R_B that contains all the points of A and at least one point of B. But, T_{BP} may not yet be a tree. Thus, we remove from T_{BP} any edge (but not its end vertices) without disconnecting T_{BP} (this can only further

decrease the total length of T_{BP}). We continue this edge-removal process until T_{BP} becomes a tree, which is a sought RST for A touching at least one point of B on R_B, and its total length is no bigger than k. Hence, the resulting T_{BP} is a solution to BRSTP for A on R_B with $|T_{BP}| \leq k$.

Finally, it is easy to see that given a solution C_{SF} to SF-D on S, a corresponding solution T_{BP} to BRSTP on R_B can be obtained in polynomial time in terms of the input size $|A|$. □

Because the decision version of the surface flattening problem is NP-complete, the optimization version of this problem is NP-hard.

3 A $(1 + \epsilon)$-Approximation SF Algorithm

Since the optimal surface flattening problem (SF) is NP-hard, it is unlikely that a deterministic polynomial time algorithm for optimally solving this problem is possible unless P = NP. Thus, we present an efficient method for finding a provably good approximate solution for the problem.

Our approach is to solve the SF problem using a graph representation. We model the mesh surface $S = (N, M)$ using an undirected weighted graph $G = (V, E)$, where $V = N$ and $E = M$, i.e., the vertices of G correspond to the point set of S and the edges of G correspond to the line segments of S. We let $V_A \subseteq V$ be the subset of vertices corresponding to the points of N_A whose absolute values of the Gaussian curvatures are greater than the threshold value. The weight of an edge $e = (v, w)$ in E is equal to the length of the segment between the mesh points represented by vertices v and w. Since S is a bounded surface in $\mathbb{R}^3$, the graph G is planar and a planar embedding of G can be easily specified by S such that the boundary of S corresponds to the border of the outer face of the planar embedding of G. Clearly, the one-to-one correspondence between the mesh points on the boundary of S and the vertices on the outer face of G defines the vertex subset $V_B \subseteq V$ as corresponding to $N_B \subseteq N$.

Given the graph representation G of the surface S, observe that the union of the optimal cutting paths connecting some boundary points of S with all the points of N_A along the segments of S corresponds to a set of trees, or a forest, F, in G whose roots all lie on the outer face. It is required that the total length of these optimal cutting paths (or trees) be minimized. The reason that the union of the optimal cutting paths for S must form a forest is as follows. Suppose the union, F, in G contains a cycle. Then we can remove from F any edge (but not its end vertices) on the cycle, still retaining a feasible cutting solution for S; but, the solution thus obtained has a smaller total length than that of F, a contradiction to the assumption that F corresponds to the union of the optimal cutting paths for S.

The optimal surface flattening problem defined on the graph model G is as follows.

Terminal Cutting Problem on a Planar Graph (TCPG). *Given an embedded undirected weighted planar graph $G = (V, E)$ representing a mesh surface*

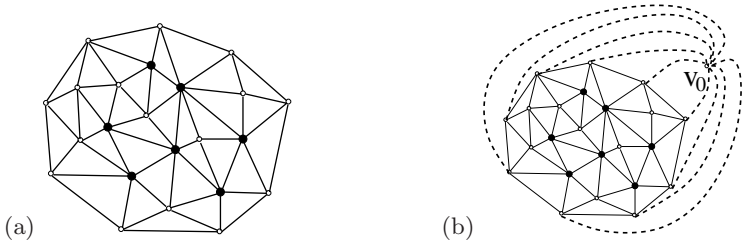


Fig. 3. (a) A planar graph G representing the surface S (the vertices of V_A are denoted by solid dots, and the vertices of V_B are on the outer face of G). (b) The planar graph G' augmented by adding the supernode v_0 and the edges connecting v_0 with all the vertices of V_B.

S in $\mathbb{R}^3$, $V_A \subseteq V$, which is a set of terminals (corresponding to the points of A on S whose absolute values of the Gaussian curvatures are above a threshold value), and $V_B \subseteq V$, which is the set of vertices on the outer face of G (corresponding to the boundary nodes of S), find a subgraph F of G with the minimum total sum of edge weights connecting every vertex of V_A to at least one vertex of V_B.

We solve the TCPG problem by transforming it to the optimal Steiner tree problem on planar graphs, which is known to be NP-hard [6], and applying an $O(n \log n)$ time $(1+\epsilon)$-approximation algorithm for the optimal Steiner tree problem on planar graphs [5] to finish the job.

Optimal Steiner Tree Problem on a Planar Graph (OSTPG). *Given an undirected planar graph $H = (V_H, E_H)$ with nonnegative edge weights and a set $T \subseteq V_H$ of terminals, find a tree F' in G with the minimum total sum of edge weights that contains all terminals of T.*

The key to the transformation from TCPG to OSTPG is to make certain changes to the graph G while preserving the planarity of the resulting graph G' and to specify which of the vertices in G' are to be terminals in T for OSTPG. Clearly, T should include all the vertices of V_A. But, since TCPG requires that each vertex of V_A be connected with some vertex on the outer face of G, a feasible solution to OSTPG needed by TCPG must include connections to some vertices of V_B. To satisfy this requirement, we add to G a new vertex v_0, which is also a terminal in T and is called a *supernode*, and add edges that connect v_0 with all the vertices of V_B such that the weights of these added edges are all zero. To preserve the planarity of the resulting embedded graph, v_0 and its edges are all placed in the interior of the outer face of G. Let G' be the new graph thus obtained from G. Figure 3 illustrates the transformation from G to G'. Clearly, G' is planar and has a set of terminals that includes v_0. Furthermore, G' has $O(|V|)$ vertices and $O(|E|) = O(|V|)$ edges. The next lemma shows the equivalence of a solution to TCPG on G and a corresponding solution to OSTPG on G'.

Before we proceed to the next lemma, observe that a non-optimal solution to TCPG is a subgraph of G that may consist of one or more connected components,

each of which contains some vertices of V_A and at least one vertex of V_B and may even contain some cycles (i.e., it may not be a tree). However, when this case occurs, we can, easily in linear time (say, based on depth-first search), remove any edges (but not their end vertices) from each non-tree component without disconnecting it until the component becomes a tree. This edge-removal process does not affect the feasibility of the resulting TCPG solution for G, and the total sum of edge weights of the resulting TCPG solution can only decrease. Thus, without loss of generality, we assume for the rest of this paper that any solution to TCPG consists of $l \geq 1$ connected components in G, each of which is a tree.

Lemma 3. *Let $G = (V, E)$ be an undirected, weighted embedded planar graph representing a mesh surface S, $V_A \subseteq V$ be the subset of vertices representing the set A of points on S whose absolute Gaussian curvatures are above a threshold value, and $V_B = \{v_{b_1}, \ldots, v_{b_k}\} \subseteq V$ be the subset of vertices on the outer face of G representing the nodes on the boundary of S. Furthermore, let $G' = (V \cup \{v_0\}, E \cup \{(v_0, v_{b_1}), \ldots, (v_0, v_{b_k})\})$, and $T = V_A \cup \{v_0\}$ be a set of terminals in G'. Then a solution to TCPG on G can be obtained from a solution to OSTPG on G' (with the same sum of edge weights), and* vice versa. *Moreover, the transformation takes $O(n)$ time, where $n = |V|$.*

Proof. The proof of the equivalence of the solutions to OSTPG and TCPG is not difficult to show. Due to the space limit, the detailed proof is omitted here and can be found in the full version of the paper. □

Using the algorithm for OSTPG by Borradaile, Klein, and Mathieu [5], we obtain a $(1+\epsilon)$-approximate solution to TCPG (and thus to SF) in $O(2^{\text{poly}(1/\epsilon)} n \log n)$ time, for any constant $\epsilon > 0$. Note that for any constant $\epsilon > 0$, $2^{\text{poly}(1/\epsilon)}$ is a (possibly large) constant as well. Hence the running time of our $(1+\epsilon)$-approximation SF algorithm is $O(n \log n)$.

Theorem 4. *The surface flattening problem can be solved by a $(1+\epsilon)$- approximation algorithm in $O(2^{\text{poly}(1/\epsilon)} n \log n)$ time, for any constant $\epsilon > 0$.*

Proof. This follows immediately from Lemma 3 and the PTAS result in [5]. □

For any constant $\epsilon > 0$, we have obtained in $O(n \log n)$ time an approximate solution (i.e., a cutting of the surface S) whose total length is no more than $1+\epsilon$ times the total length of the optimal solution. Due to its NP-hardness, this appears to be a theoretically "best possible" approximation solution for the SF problem.

References

1. Aono, M., Breen, D.E., Wozny, M.J.: Modeling methods for the design of 3D broadcloth composite parts. Computer-Aided Design 33(13), 989–1007 (2001)
2. Aona, M., Denti, P., Breen, D.E., Wozny, M.J.: Fitting a woven cloth model to a curved surface: Dart insertion. IEEE Computer Graphics and Applications 16(5), 60–70 (1996)

3. Azariadis, P.N., Sapidis, N.S.: Planar development of free-form surfaces: Quality evaluation and visual inspection. Computing 72(1-2), 13–27 (2004)
4. Borradaile, G., Kenyon-Mathieu, C., Klein, P.N.: A polynomial-time approximation scheme for Steiner tree in planar graphs. In: Proc. of 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1285–1294 (2007)
5. Borradaile, G., Klein, P.N., Mathieu, C.: Steiner tree in planar graphs: An $O(n \log n)$ approximation scheme with singly exponential dependence on epsilon. In: Proc. of 10th International Workshop on Algorithms and Data Structures, pp. 276–287 (2007)
6. Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem is NP-complete. SIAM Journal on Applied Mathematics 32(4), 826–834 (1977)
7. Karp, R.: On the computational complexity of combinatorial problems. Networks 5, 45–68 (1975)
8. Kim, S.M., Kang, T.J.: Garment pattern generation from body scan data. Computer-Aided Design 35(7), 611–618 (2003)
9. Kobbelt, L.P., Bischoff, S., Botsch, M., Kähler, K., Rössl, C., Schneider, R., Vorsatz, J.: Geometric modeling based on polygonal meshes. In: Eurographics 2000 Tutorial (2000)
10. McCartney, J., Hinds, B.K., Seow, B.L.: The flattening of triangulated surfaces incorporating darts and gussets. Computer-Aided Design 31(4), 249–260 (1999)
11. Parida, L., Mudur, S.P.: Constraint-satisfying planar development of complex surfaces. Computer-Aided Design 25(4), 225–232 (1993)
12. Sheffer, A.: Spanning tree seams for reducing parameterization distortion of triangulated surface. In: Proc. of International Conference on Shape Modeling and Applications, pp. 61–68 (2002)
13. Wang, C.L., Smith, S.F., Yuen, M.F.: Surface flattening based on energy model. Computer-Aided Design 34(11), 823–833 (2002)
14. Wang, C.L., Wang, Y., Tang, K., Yuen, M.F.: Reduce the stretch in surface flattening by finding cutting paths to the surface boundary. Computer-Aided Design 36(8), 665–677 (2004)

Visiting a Polygon on the Optimal Way to a Query Point*

Ramtin Khosravi[1] and Mohammad Ghodsi[2,3]

[1] School of Electrical and Computer Engineering, University of Tehran, P.O. Box 14395-515, Tehran, Iran
[2] Department of Computer Engineering, Sharif University of Technology, P.O. Box: 11365-9517, Tehran, Iran
[3] School of Computer Science, Institute for Studies in Theoretical Physics and Mathematics, P.O. Box: 19395-5746, Tehran, Iran
rkhosravi@ece.ut.ac.ir, ghodsi@sharif.ir

Abstract. We study a constrained version of the shortest path problem in polygonal domains, in which the path must visit a given target polygon. We provide an efficient algorithm for this problem based on the wavefront propagation method and also present a method to construct a subdivision of the domain to efficiently answer queries to retrieve the constrained shortest paths from a single-source to the query point.

1 Introduction

In this paper, we study the problem of finding a shortest path between two points inside a polygonal domain P (a simple polygon with a number of polygonal holes inside it) while the path is constrained to visit (i.e. has non-empty intersection with) a given *target polygon* T inside the free space. We call such a path a *shortest T-visiting path*. Our goal is to construct the *shortest T-visiting path map* $\mathrm{SPM}_T(s, P)$, a decomposition of the free space into a number of regions such that the combinatorial structure of the shortest T-visiting path from the given source point s to any point in a region is the same. This way, we will be able to find the length of the shortest T-visiting path between s and any query point in logarithmic time and report the actual path with an additional cost proportional to the complexity of the path. If the number of vertices in P and T be n and m respectively, and $N = m + n$, we preprocess the input (s, P, T) in $O(N \log N)$ time to construct a subdivision of the same space, so that the queries can be answered in $O(\log N)$ time.

Our method is based on the *continuous Dijkstra* paradigm to compute shortest paths in polygonal domains [4,10]. The main idea of our algorithm is to propagate a wavefront from s to visit T. Parts of T reached this way work as pseudo-sources for finding shortest T-visiting paths to points of the free space. We let those wavelets that first visit T propagate further and also propagate back a

* This work has been supported by a grant from IPM School of CS (No. CS1382-2-02).

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 245–251, 2008.

reflected version of those wavelets to cover points of the free space that shortest T-visiting paths to them visits the boundary of T and reflect back. This idea has been previously introduced in [7] by the authors.

A similar problem has been studied by the authors for the case of simple polygons [8] resulting in a linear algorithm for convex target polygons, as well as a method to construct the shortest T-visiting path map. The method presented here can be used to solve that problem for non-convex polygons. Extending the problem to multiple target polygons makes the problem similar to TSP with neighborhoods [2,3] which is NP-hard. Dror et al. [1] have presented an algorithm for the problem of finding the shortest path that visits k given convex polygons in a pre-specified order. Also, they have shown that the problem is NP-hard for the case of non-convex polygons.

We first introduce the concept of *reflective subdivision* in Sect. 2 which determines the structure of the shortest T-visiting paths without any obstacles in the plane. Then we extend this concept to the general case of polygonal domains in Sect. 3.

2 The Reflective Subdivision

To study the properties of the constrained shortest paths, we start by a simple case in which there is no obstacles in the plane. We are given a (possibly non-convex) *target polygon* T with m vertices and a point s outside T. We define $G(s)$ as the set of points where shortest T-visiting paths from s have their first intersections with T and call it the *gate* of s to T, or the gate of s for short. If T is a convex polygon, $G(s)$ is a connected chain on the boundary of T [1]. In general, $G(s)$ is not connected, but since we assumed there is no obstacles in the plane, there is at most one segment in $G(s)$ on any edge of T (which of course may be the entire edge).

Computing $G(s)$ can be easily done using algorithms for computing visibility polygon of a point in simple polygons [9,5], since if we consider T as an obstacle, $G(s)$ is the part of the boundary of T visible from s.

For an arbitrary point x, consider a shortest T-visiting path from s to x. If the segment $\overline{sx}$ intersects T, this segment is the desired path. The set of such points x makes a connected region called the *pass-through region*. If $\overline{sx}$ does not intersect T, the path consists of two segments $\overline{sc}$ and $\overline{cx}$ where c is a point on the boundary of T. We call c the *contact point*.

If c is an interior point of an edge e of T, the angle between $\overline{sc}$ and e is the same as the angle between e and $\overline{cx}$. We call the part of $G(s)$ on the interior of e an *edge-reflector*. If c is a vertex v of T, we call v a *vertex-reflector*. It easy to see that a vertex in $G(s)$ is a vertex-reflector if the angle made by the incident edges inside T is less than π. We define the *root* of a reflector r as r itself if it is a vertex-reflector, and s reflected about r if it is an edge-reflector.

Since the pass-through region can be easily computed and the corresponding shortest T-visiting paths are straight segments, we limit our attention to its complement, called D.

The *reflective subdivision* $\mathrm{RS}(s,T)$, or $\mathrm{RS}(s)$ for short, is the decomposition of D into faces such that the contact point of every point in a face is the same vertex-reflector r, or belongs to the same edge-reflector r. We call such a face a *reflective region* of r (Fig. 1).

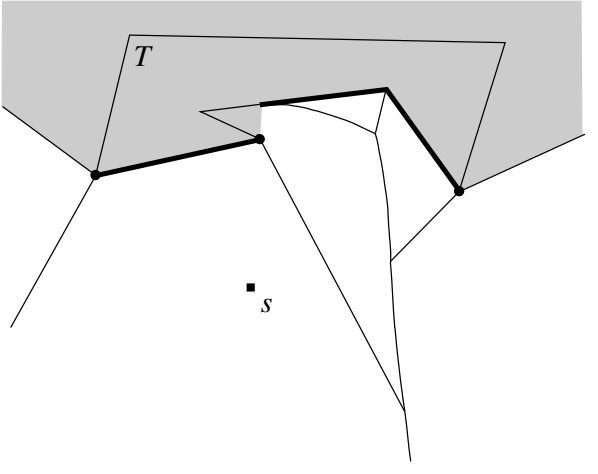


Fig. 1. The reflective subdivision $\mathrm{RS}(s,T)$: The shaded area is the pass-through region. The edge-reflectors are shown in thick segments and the vertex-reflectors are shown in black circles.

Some edges of $\mathrm{RS}(s)$ are from the boundary of D. Other edges separate reflective regions of different reflectors. In general, the edge separating two regions of reflectors r_1 and r_2 with roots a and b respectively, is defined by the bisector curve of a and b. This curve is a hyperbolic curve in general and is the locus of points x such that $w(a) + |\overline{ax}| = w(b) + |\overline{bx}|$ where $w(a)$ is $|\overline{sa}|$ if a is a vertex-reflector and zero if it is an edge-reflector. The following lemma establishes a linear bound on the size of this decomposition.

Lemma 1. *For a target polygon T with m vertices, the complexity of* $\mathrm{RS}(s,T)$ *is* $O(m)$.

Proof. We prove that for a reflector r, there is at most one reflective region. First observe that the points on a reflector r belong to its own reflective regions. Now consider a point x in a reflective region of r and assume c is the contact point of x. We can easily check that every point y on $\overline{cx}$ belongs to the same reflective region.

Now consider a case in which there is a reflector r with root a that has two reflective regions f_1 and f_2. Since these two regions are distinct, there exists a ray R emanated from a in the space between f_1 and f_2 such that every point on R from the intersection of R and r away from a belongs to the reflective regions of reflectors other than r. Let x be the intersection of R and r. Then, the length

of the T-visiting path between s and x through r is the same as the length of such a path through another reflector namely r'. So, x lies on the bisector curve of the roots of r and r'. If this curve is a straight line, part of either f_1 or f_2 will be in the half-plane geodetically closer to r' which is impossible. The case that the curve is a not a straight line and has two intersections with r is not acceptable since part of r will reside in the reflective region of r'. So, there is at most one face in $\mathrm{RS}(s)$ corresponding to a reflector r, hence the number of faces is $O(m)$. The vertices of this subdivision are of these kinds: vertices of T, endpoints of edge-reflectors, and intersections between bisectors. The number of the first two kinds is $O(m)$. A vertex of the third kind borders at least three faces, hence the total number of vertices is $O(m)$. □

We can compute $\mathrm{RS}(s)$ in $O(m \log m)$ time and $O(m)$ space using a simple sweep process. For a point $x \in D$, define $\delta(s, x)$ to be the length of the shortest T-visiting path from s to x. We sweep D based on the increasing value of δ. The sweep structure is a wavefront consisting of circular arcs (wavelets) centered at the roots of the reflectors. Initially, there will be a wavelet corresponding to each reflector. The *release time* for a vertex-reflector with root a is the length of $\overline{sa}$. At any instant during sweep, we say two bisectors are adjacent if they bound the same wavelet.

The only event in the sweep process occurs when the two bisectors separating the region of r from its two adjacent regions intersect. At this time the wavelet sweeping the region of r disappears and its two neighbors become adjacent. Since the intersections only occur between two adjacent bisectors, when processing an event, we can compute the times at which the newly created bisector intersects its two adjacent bisectors. It is easy to see that processing all $O(m)$ events can be done in $O(m \log m)$ time and $O(m)$ space.

3 Polygonal Domains

Let P be a polygonal domain having n vertices. A T-visiting path is a path in the free space having non-empty intersection with the target polygon T which we assume to have m vertices. We define the gate of s as before as the set of points where the shortest T-visiting paths from s have their first intersections with T. Again, $G(s)$ consists of a number of segments on the boundary of T.

Consider a maximally connected set of points on an edge e of T such that the last vertex on the shortest paths from s to them is the same vertex v of P. We call such a segment an edge-reflector and define its root as v reflected about e. Like before, a vertex of T in $G(s)$ is called a vertex-reflector and its root is the vertex itself. The edge-reflectors are a subset of the segments made on the boundary of T when intersected by $\mathrm{SPM}(s, P)$. In general, there can be $O(mn)$ such segments, but the following lemma shows that only $O(m + n)$ of these segments are edge-reflectors.

Lemma 2. *For a polygonal domain and a target polygon having n and m vertices respectively, there are $O(m + n)$ edge-reflectors.*

Proof. Let f be a cell of SPM(s) with root r. By adding three kinds of segments, we can decompose f into triangle-like regions (Fig. 2):

1. Segments connecting r to the intersection points between the boundaries of T and f,
2. segments connecting r to the vertices of f, and
3. segments each connecting r to some point on the boundary of f passing through a vertex of T inside f.

Since f is star-shaped with kernel r, all these segments are inside f and connect r to some point on the boundary of f. The regions obtained this way are either triangles, or bounded by two segments incident to r and a hyperbolic curve. Each region intersects a number of edges of T (possibly zero), but there is no vertices of T *inside* a region. Thus, the intersection of a region with T makes a number of segments with their end-points lying on the two boundary segments incident to r. Since the segments does not intersect inside the region, they can be ordered according to the increasing distance from r. It easy to check that only the nearest segment to r is a part of an edge-reflector. Since the shortest T-visiting paths to points on other segments already intersect it. Since there are at most $O(m+n)$ triangle-like regions in total, the number of edge-reflectors is bounded by the same order. □

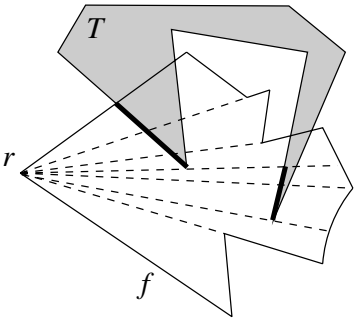


Fig. 2. Proof of lemma 2: The thick segments are parts of $G(s)$

To compute the set of reflectors, we can use the algorithms for constructing the shortest path map of a polygonal domain such as the algorithm of Hershberger and Suri [4] or that of Mitchell [10]. To do this, we consider T as an obstacle and define the polygonal domain $P' = P - T$. If we construct SPM(s, P'), the boundary of T is partitioned into a number of segments. Some of these segments are edge-reflectors. Consider a segment that is made by the SPM cell with root r. If r is a vertex of T, then the segment cannot be a part of $G(s)$. Assuming r is a vertex of P, we locate r in the two shortest path trees SPT(s, P) and SPT(s, P'). If the path from s to r is the same in both trees, considering T as an obstacle has no effect in the shortest path to the points in the segment under consideration, so it is an edge-reflector. So, computing the edge-reflectors can be

done in $O((m+n)\log(m+n))$ time and the same space. This computation also produces a list of vertex reflectors.

We define the pass-through region as before. Let D be the free space with the pass-through region removed. $\mathrm{RS}(s,T)$ is the partition of D into regions according to the reflector that is first visited along shortest T-visiting paths from s. A similar argument as the one in lemma 1 proves there are $O(m)$ regions in the subdivision and its complexity is $O(m+n)$.

Computing the shortest T-visiting path map $\mathrm{SPM}_T(s,P)$ can be done using wavefront propagation method. This map has two parts: one corresponding to the pass-through region, and another for D. The first part is $\mathrm{SPM}(s,P)$ restricted to the pass-through region. For the second part, we have multiple sources which are the roots of the reflectors. Each source has a specified release-time. For vertex-reflectors, the release time is the geodesic distance from s to that vertex, and for the edge-reflectors, it is the geodesic distance from s to the last vertex v on the shortest paths from s to points on the edge-reflector, plus d which is the distance from v to the reflector segment. To cover points in D, we use a wavefront propagation algorithm to "reflect back" those parts of the original wavefront started from s that have visited T. Note that the initial wavelets are to be computed carefully, since some sources may lie outside D. For an edge-reflector, if v is the last vertex on the shortest path from s to points on the reflector, the initial wavelet is centered at the $\bar{v}$ which is v reflected about the edge-reflector, and the radius is d.

Both algorithms of [4] and [10] are capable of handling multiple source with specified release-times. If we use the first algorithm (that of Hershberger and Suri) which is worst-case optimal, we obtain $O((m+n)\log(m+n))$ time and space bounds to construct $\mathrm{SPM}_T(s,P)$. Since computing the reflectors can be done using the same algorithm, the order remains the same for the entire computation. Hence we have our main result as follows.

Theorem 1. *For a polygonal domain P and a target polygon T inside P with N vertices in total, and a source point s, we can compute the shortest T-visiting path map $\mathrm{SPM}_T(s,P)$ in $O(N\log N)$ time and space.*

4 Conclusion

We showed how one can use the wavefront propagation method to partition the free space in a polygonal domain according to the combinatorial structure of shortest paths from a given source point s to the points in the free space that has non-empty intersection with a target polygon T. We showed how to compute this subdivision having an algorithm for wavefront propagation capable of handling multiple sources with specified release-times. The best known method so far ([4]) solves this problem in $O((m+n)\log(m+n))$ time and space.

We leave an open problem that is whether one can use the methods based on searching the visibility graph to find the shortest T-visiting path between two points. This is particularly important, since the best known algorithm using this method by Kapoor et al. [6], solves the shortest path problem in polygonal

domains in $O(n + h^2 \log n)$ which is only linear in n, while being quadratic in the number, h, of holes.

References

1. Dror, M., Efrat, A., Lubiw, A., Mitchell, J.S.B.: Touring a sequence of polygons. In: Proc. 35th ACM Sympos. Theory Comput (2003)
2. Dumitrescu, A., Mitchell, J.S.B.: Approximation algorithms for TSP with neighborhoods in the plane. In: Symposium on Discrete Algorithms, pp. 38–46 (2001)
3. Gudmundsson, J., Levcopoulos, C.: A Fast Approximation Algorithm for TSP with Neighborhoods and Red-Blue Separation. In: Asano, T., Imai, H., Lee, D.T., Nakano, S.-i., Tokuyama, T. (eds.) COCOON 1999. LNCS, vol. 1627, pp. 473–482. Springer, Heidelberg (1999)
4. Hershberger, J., Suri, S.: An optimal algorithm for Euclidean shortest paths in the plane. SIAM J. Comput. 28(6), 2215–2256 (1999)
5. Joe, B., Simpson, R.B.: Correction to Lee's visibility polygon algorithm. BIT 27, 458–473 (1987)
6. Kapoor, S., Maheshwari, S.N., Mitchell, J.S.: An efficient algorithm for euclidean shortest paths among polygonal obstacles in the plane. Discrete Comput. Geom. 18, 377–383 (1997)
7. Khosravi, R., Ghodsi, M.: Shortest paths in polygonal domains with polygon-meet constraints. In: Proc. 19th European Workshop Comput. Geom., pp. 137–142 (2003)
8. Khosravi, R., Ghodsi, M.: Shortest paths in simple polygons with polygon-meet constraints. Inform. Process. Lett. 91, 171–176 (2004)
9. Lee, D.T.: Visibility of a simple polygon. Comput. Vision Graph. Image Process 22, 207–221 (1983)
10. Mitchell, J.S.B.: Shortest paths among obstacles in the plane. Internat. J. Comput. Geom. Appl. 6, 309–332 (1996)

Constraint Abstraction in Verification of Security Protocols*

Ti Zhou[1], Zhoujun Li[2], Mengjun Li[1], and Huowang Chen[1]

[1] School of Computer Science,National University of Defence Technology, Changsha, 410073, China
[2] School of Computer, BeiHang University, Beijing, 100083, China
lizj@buaa.edu.cn

Abstract. This paper incorporates time constraints in the Horn logic model, and this extended model can verify Wide-Mouthed-Frog protocol quickly. It discusses relations between the constraint system and Horn model, abstracts the constraint system, and gives the proofs of some propositions and theorems. We also give the algorithm about how to compute the abstract constraint, and analyze its complexity. As a case study we discuss the verification of Wide-Mouthed-Frog protocol whose attack can be found quickly in new model. Therefore, the method in this paper is very effective in verification of time sensitive security protocols. In the future, we will use this method to verify some complex protocols, such as Kerberos protocol etc.

Keywords: time sensitive; security protocol; logic model; formal verification; constraint system.

1 Introduction

Network security protocols are difficult to design and debug. Some attacks are found after protocols have used many years [1]. Many flaws cannot be found by hands. Therefore, the formal verification of security protocols is one of the key fields of protocol design [2]. However, there is no effective method to model and verify the validity of messages and security properties when the session key is expired. In many cases, the analysis was carried out at the symbolic Dolev-Yao level [3,4] which cannot verify protocols with time stamps. [5] gives a Horn logic model with constraints to verify time sensitive protocols. The method is easy to understand, but constraints will be too much in the process of resolution. In this paper, we will discuss how to abstract constraints and reduce the complexity.

Many approaches focus on the study of protocols that use time stamps [6,7,8,9,10,11,12,13]. [6] analyzes protocols with timing information based on a discrete

* Supported by the National Natural Science Foundation of China under Grant No. 60473057, 90604007, 60703075, 90718017, the National High Technology Research and Development Program of China No. 2007AA010301, and the Research Fund for the Doctoral Program of Higher Education No. 20070006055.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 252–263, 2008.

time model. [7] uses a model of discrete time with an upper bound on the time window. It can identify a timed-authentication attack on Wide-Mouthed-Frog protocol. [6,7] have to define a finite integer set as the range of time stamps, so time stamps are discrete, and Casper/FDR can just check a finite number of sessions. In our method, the range of time stamps is in a continuous interval of real number without no the upper bound, and this method can verify protocols with an unbounded number of sessions. Last but not least, constraints assure there is few false attacks generated by time stamps. [8] makes a semi-automated analysis on a Timed CSP model of Wide-Mouthed-Frog protocol. It uses the PVS proving system to discharge proof obligations to find an invariant property. [9] presents a real-time process algebra for the analysis of time-dependent properties. It focuses on compositional results and shows how to model timeouts. Theory of timewise refinement [10] allows refinement to be translated between the untimed and timed models, thus enabling verifications to be carried out at their most appropriate level of abstraction and then combined, if necessary, from different models. [11] proposes a method for design and analysis of security protocols that are aware of timing issues. It models security protocols using timed automata, and uses UPPAAL to simulate, debug and verify protocols. The verification method in [12] is based on the combination of constraint solving techniques and first-order term manipulation with Sicstus Prolog. A global clock is assumed to verify the Wide-Mouthed-Frog protocol. [13] presents a symbolic decision procedure for time sensitive cryptographic protocols with time stamps. It uses logic formulae to describe symbolic constraints. However, it does not give an automatic method to determine how to choose time variables to obtain an attack.

Based on the Horn logic model, [14,15,16] present an effective verification method which fits for verifying interleaving runs of the security protocol's infinite sessions and terminates for many security protocols. In [17,18], an extended Horn logic model for security protocols is proposed, and the modified-version verification method to construct counter-examples automatically is presented. To verify time sensitive security protocols, [5] extends this approach to verify protocols with time stamps effectively in a continuous domain.

This paper presents that the theory of abstract constraint system. We establish some novel theorems to guarantee the correctness of the abstraction process of constraint system. By abstracting constraint system, the verification complexity is reduced. Many constraints can be deleted. The number of resolution of logic rules will decline due to the higher abstract level. As an example, we consider the timed version of the Wide-Mouthed-Frog protocol (WMF for short). The verification of the abstract constraint system is effective.

The paper is organized as follows. In Section 2, we describe the protocol model and the resolution briefly. In Section 3, we characterize constraints in resolution and solve the abstract constraint system. Section 4 shows the result about the verification under abstract constraint system. In Section 5, we give the summarization and the future work.

2 Protocol Model

2.1 Protocol Preliminaries

The Wide-Mouthed-Frog protocol [1,2] is one of the classic time sensitive protocols. If time sensitive security protocols can be verified, we can detect the attack of the Wide-Mouthed-Frog protocol(WMF for short). The formal description of WMF is given in Table 1, and its attack is described in Table 2.

Table 1. The formal description of Wide-Mouthed-Frog protocol

$$\begin{aligned} &\text{A}\rightarrow\text{S} : A, \{B, K, T_1\}Kas \\ &\text{S}\rightarrow\text{B} : \{A, K, T_2\}Kbs \\ &\text{B}\rightarrow\text{A} : \{Secret\}K \end{aligned}$$

Table 2. The attack of WMF

$$\begin{aligned} &\text{I(B)}\rightarrow\text{S} : B, \{A, K, T_2\}Kbs \\ &\text{S}\rightarrow\text{I(A)} : \{B, K, T_3\}Kas \\ &\text{I(A)}\rightarrow\text{S} : A, \{B, K, T_3\}Kas \\ &\text{S}\rightarrow\text{B} : \{A, K, T_4\}Kbs \\ &\text{B}\rightarrow\text{I(A)} : \{Secret\}K \end{aligned}$$

Table 1 denotes a protocol in which A (Alice) sends a message $A, \{B, K, T_1\}Kas$ to S (Server), and then, S sends another message $\{B, K, T_3\}Kas$ to B (Bob) who then sends $\{Secret\}K$ to A. We denote a mischievous principal by I. The notation $I(A)$ denotes that the principal I impersonates A. In the first message of WMF, A sends to S the message whose components are a principal identifier B together with an encrypted tuple $\{A, K, T_2\}Kbs$. K is a pseudo-random session key, T_2 is a time stamp and Kbs is a shared key between B and S. This protocol uses T_1, T_2 to assure that respective messages are unexpired. However, the attacker can repeat these actions in Table 2, and then, the server S will transmit an encrypted message with a fresh time stamp to B. For the session key K will be revealed after a period of time, and the session between A and B has been finished. However, when S sends the session request $\{A, K, T_4\}Kbs$ to B, B misunderstands that A wants to have a new session and the session key is K. In fact, the attacker replays this message and masquerades A to begin a new session with B, and then, the attacker I can use the old session key K to steal secret data which B thought to send to A.

2.2 Horn Logic Syntax

Constraint is a linear equality or inequality whose variables are used to describe time. A logic rule is the form of $F_1 \wedge \cdots \wedge F_n \rightarrow F : C$ where C is constraint. New terms are introduced for time variable, such as time stamps. The Horn logic uses these predicates: *attacker*, *begin*, and *end*. $attacker(M)$ means that the attacker may have M. $begin(M, N)$ means that the event **begin** has been executed with a parameter corresponding to M and environment N. $end(M, N)$ means that **end** has been executed in session list N with a parameter corresponding to M. [5] defines a global clock **now** which is a special place-holder, and if the constraint is **true**, then the rule will ignore this constraint. In other words, $H_1 \wedge \cdots \wedge H_n \rightarrow F$ means $H_1 \wedge \cdots \wedge H_n \rightarrow F : true$. The clock **now** represents the current time. The sender overprints time stamps referring to **now**. When the receiver receives this

message, he will check time stamps with the current time. If time stamps have not expired, he will believe the message is still valid. The parameters representing network delays can be assigned in the beginning.

2.3 The Model for the Attacker

The intruder's Dolev-Yao model [3, 4], which is very popular in analysis and verification of security protocols, describes that the intruder can control the network wholly in perfect encryption assumption. However, after a long time, some low-entropy keys are easy to be vulnerable. Generally speaking, the session key between agents is easy to be leaked by accident when they have expired. The attack of WMF protocol belongs to this case. Therefore, the Dolev-Yao model with constraints needs to add the rule $\mathbf{DY}_4$ which means session keys can be leaked after a long time. The Dolev-Yao model is described in Horn model with constraints as follows:

$\mathbf{DY}_1$ $\rightarrow attacker(M) : true$, where M belongs to the public knowledge set S;
$\mathbf{DY}_2$ $attacker(x_1) \wedge \cdots \wedge attacker(x_n) \rightarrow attacker(f(x_1, \cdots, x_n)) : true$,
where f is an n-ary constructor;
$\mathbf{DY}_3$ $attacker(M_1) \wedge \cdots \wedge attacker(M_n) \rightarrow attacker(M) : true$,
where g is a destructor and there is a reduction $g(M_1, \cdots, M_n) = M$;
$\mathbf{DY}_4$ $\rightarrow attacker(k[t]) : \Delta + t \leq \mathbf{now}$

In the rule $\mathbf{DY}_4$, the lifetime of the session key k is Δ, where Δ is a big time constant which describes the safe period of session keys, and $k[t]$ means that k is generated at the time clock t, and session keys can only be leaked by accident when they have expired.

2.4 The Model of the Honest Roles

Every honest role A will do some action after she receives certain messages. When A sends a message M, a rule R will be added to the model of the honest roles. The head of R is an atomic $attacker(M)$. Its body can be empty or conjunction of some predicates, such as $attacker(M_1) \wedge \cdots \wedge attacker(M_n)$, where $M_1, \cdots, M_n$ have been received before A sends M.

To deal with time stamps, by adding constraints to honest roles, we should obey the following principles:

- When a rule is added into a role, it is necessary to include a time constraint $\mathbf{now} - d \leq T \leq \mathbf{now}$ for the message which has a time stamp T in hypothesis;
- When agent needs to send a message which contains a time stamp t to show the current time, the generated rule needs to contain a time constraint $t = \mathbf{now}$;
- The new generated session key is represented in the form of *name*, and the parameter is a time stamp when it's generated.

Example 1 (The model of Wide-Mouthed-Frog protocol)

A:① $\rightarrow attacker(A) : true$
② $\rightarrow attacker(\{B, K[T_1], T_1\}Kas) : T_1 = \mathbf{now}$

S:③ $attacker(xA) \wedge attacker(\{xB, xK, xT_1\}Kxas) \rightarrow attacker(\{xA, xK, T_2\}Kxbs) : \mathbf{now} - d \leq xT_1 \leq \mathbf{now}, T_2 = \mathbf{now}$

B:④ $attacker(\{xA, xK, xT_2\}Kbs) \rightarrow attacker(\{Secret\}xK) : \mathbf{now} - d \leq xT_2 \leq \mathbf{now}$

These rules represent actions of role Alice (marked A), Server (marked S), and Bob (marked B) respectively. If there are facts of the body of a rule in the network and the constraint system can be satisfiable, the corresponding role will send the message (as the action in the head of the rule).

2.5 Resolution

Let $R = \mathcal{H} \rightarrow F : C$, $GetRule(R) = \mathcal{H} \rightarrow F$, $GetCons(R) = C$. Suppose that σ is a unifier, and σ' is the maximal sub-constraint of σ involving only time terms. If L is a constraint or a set of constraints, $\sigma|_L$ is the maximal sub-constraint of σ involving only the variables in L.

Definition 1 (Resolution). *Let* $R_1 = H_{11} \wedge H_{12} \wedge \cdots \wedge H_{1n} \rightarrow F : L_1$ *and* $R_2 = H_{21} \wedge H_{22} \wedge \cdots \wedge H_{2m} \rightarrow C : L_2$ *be two logic rules,* $H_{2i} = attacker(Mes), H_{2i} = attacker(Mes')$*, or* $F = end(M, Mes), H_{2i} = end(M, Mes'), 1 \leq i \leq m$*, such that* Mes *can be unified with* Mes'*, and* $\theta = mgu(Mes, Mes')$ *is the most general unifier of* Mes *and* Mes'*. Let* $L = (L_1 \cup L_2)\theta' \cup \{t_1\theta' \leq t_2\theta' | t_1 = \max\{t | t \in dom(\theta'|_{L_1})\}$*, and* $t_2 = \max\{t | t \in dom(\theta'|_{L_2})\}\}$*, and if* L *is satisfiable, then the resolution* $R_1 \bullet R_2$ *between* R_1 *and* R_2 *is* $(H_{21} \wedge \cdots \wedge H_{2(i-1)} \wedge (H_{11} \wedge \cdots \wedge H_{1n}) \wedge H_{2(i+1)} \wedge \cdots \wedge H_{2m})\theta \rightarrow C_2\theta : \exists x_1 \cdots x_n L$*, where* $\{x_1, \cdots, x_n\} = fv(GetRule(R_1 \bullet R_2))$ *we say* $F' = selectedAtom(R_2)$ *is the selected atom of* R_2*, and* $\theta = sub(R_1, R_2)$ *is called the substitution of the resolution* $R_1 \bullet R_2$*.*

Let R_1 and R_2 have any other variables in common. R_1 provides the head of it (it sends a message). R_2 provides a fact in the body (it receives a message). So the latest time in R_1 is earlier than that in R_2. Therefore, a new constraint should be added to the final constraints to represent this relation.

3 Constraint Abstraction

We now show how to abstract the constraints in Horn logic model to reduce the verification complexity. We will discuss constraint abstraction under the satisfiability of constraint system, which is determined by the algorithm in [19]. This abstraction will delete time variables which cannot keep the freshness of messages, so the number of constraints are declined.

Proposition 1. *Every time variable will not be greater than the current time* **now** *in the constraint system of a rule.*

Proof. By section 2.5, the rule with constraints can be resolved with another one, only if constraints are satisfied. In Horn logic model, the agent will not send the head of a rule until the messages in the body of the rule are received. This means that the sender will do some action according to history messages.

So the rule represents the current action in Horn logic model with constraints, and the current time **now** is greater than other time variables. Therefore, every rule contains a constraint $t \leq \mathbf{now}$ where t is a time variable in this rule. □

Definition 2. *Let S (resp. S') be the greatest lower bound (resp. the least upper bound) of time variable t with* **now** *and constants. We define $t \in Q = [S, S']$, $\min(Q) = S$, and $\max(Q) = S'$.*

Definition 3. *Suppose $r : C_1, s : C$ are two logic rules. r is a left (resp. right) resolution fixpoint on s if $r : C_2 = r : C_1 \bullet s : C$ (resp. $r : C_2 = s : C \bullet r : C_1$). Let $t_1, \cdots, t_n$ be time stamps in r, and let the range of t_j be $Q_{ij} \subseteq \mathbb{R}[\mathbf{now}]$ in $C_i (i = 1, 2, 1 \leq j \leq n)$.*

Definition 4. *Let $r : C = r_1 : C_1 \bullet r_2 : C_2$. If $r \neq r_1$ and $r \neq r_2$, then C is the initial constraint of r, otherwise, C is the fixpoint constraint of r.*

Firstly, we compute $r_3 = r_1 \bullet r_2$, if $r_3 = r_1$, then the fixpoint constraint of r_1 is used to compute the constraint of r_3, and if there is no fixpoint constraint of r_1, then the initial constraint of r_1 is used to compute it.

In the following discussion, Q (resp. Q') represents the range of time stamp t before (resp. after) the left or right resolution fixpoint R. Let now_R represent the current time **now** in the rule R. We use Q_j to represent the range of t_j. Introducing time stamps will redefine the freshness of message.

Definition 5 (fresh). *For each message M in rule $R{:}C$, the message M is* ***weak fresh*** *if the constraints corresponding to $var(C) \cap var(M)$ are satisfiable. For each time stamp $t \in M$, $\mathbf{now} - t \leq \Delta$ is also true, then the message M is* ***fresh****.*

Proposition 2. *Let $Q_i = [0, \mathbf{now}]$, then t_i will not be constrained.*

Proof. $Q_i = [0, \mathbf{now}]$, that means t_i can be any value from the start of the protocol to the current time. So there is no constraints which can limit the range of t_i. Therefore, t_i will not be constrained. □

In the verification of time sensitive security protocols, it is necessary to discuss the runs when the session key is expired. Therefore, when $\max(Q_i) - \min(Q_i) \geq \Delta$, the protocol cannot use t_i to make sure the freshness of message, that is, after a period Δ, the attacker can make the message sent at t_i valid in a finite number of steps. By the definition of **fresh**, there is a value $a \in Q_i$ s.t. $\mathbf{now} - t_i \leq \Delta$ is false when $t_i = a$, the constraints about t_i will not be in use. So we gain the following theorem.

Theorem 1 (Freshness). *If the range of t_i in a message M is $Q_i \supseteq [\mathbf{now} - \Delta, \mathbf{now}]$, then t_i cannot preserve the freshness of M.*

Proof. For $Q_i \supseteq [\mathbf{now} - \Delta, \mathbf{now}]$, $\max(Q_i) - \min(Q_i) \geq \Delta$. There is a value $a \in Q_i$ s.t. $\mathbf{now} - t_i \leq \Delta$ is false when $t_i = a$. By the definition of **fresh**, the message M is not fresh. So t_i cannot preserve the freshness of M. □

The theorem above presents that the freshness of M is vulnerable. When there is an attack, the time stamp t_i doesn't keep the freshness of messages, so t_i is not constrained. In verification, authentication and security are very important to security protocols, and the freshness of message needn't to be satisfied, though the attack of freshness usually lead to the attack of security. Therefore, it is necessary to abstract current constraint system.

Corollary 1. *If* $\max(Q_i) - \min(Q_i) \geq \Delta$*, then the constraints containing* t_i *can be deleted.*

Because Δ is very big(the delay in network and the lifetime of message can be omitted according to the lifetime of session key), t_i can be any value in a huge range, that is, t_i is not constrained. Note that inequality will not always be held on if there is a variable in it, so these constraints couldn't be deleted.

Strategy 1 (Verification Strategy). *When verifying time sensitive security protocols, these strategies are used:*

(a) $\boldsymbol{DY}_4$ *in the attacker model isn't concerned, that is, verifying the model when session keys are not expired;*

(b) $\boldsymbol{DY}_4$ *in the attacker model should be concerned, that is, verifying the model when session keys are expired;*

If $R_1 : C = R_1 : C_1 \bullet R_2 : C_2$, then $R_1 : C_1$ is used to resolve with other rules in the step (a) but not in the step (b), and $R_1 : C$ is used in the step (b) but not in the step (a). In fact, the step (a) describes runs when session keys are not expired and the attack in this step is one that session key is not revealed. The step (b) describes runs when session keys are revealed, so the changes of time stamps caused by message exchange should be payed attention to in the last step. For example, message is thought to be valid though it is expired. So $R_1 : C$ is used as the result of the resolution between R_1 and R_2.

Definition 6 (trivial constraint). *A is a trivial constraint, if A is true or substitution.*

Proposition 3. *The form of Q is* $[\mathbf{now} - d, \mathbf{now}]$ *in the step (a).*

Proof. In the Horn logic model, the non-trivial constraint is added as follows:

1. There is only the last rule which will add a non-trivial constraint in attacker model. However, this rule will not resolve in step (a), so the attacker model will not lead to non-trivial constraints in step (a).
2. In the honest model, there are two cases which will add constraints into system:
   – to overprint the time stamp T in message M, a trivial constraint $T = \mathbf{now}$ is added.
   – to check if the time stamp T is in the valid range $[\mathbf{now} - d, \mathbf{now}]$, $\mathbf{now} - d \leq T \leq \mathbf{now}$ is added into constraint system. So the range of T is $Q = [\mathbf{now} - d, \mathbf{now}]$.
3. In resolution, the result only adds a new constraint $\mathbf{now}_{R_1} \leq \mathbf{now}_{R_2}$ where R_1 provides the head. This constraint doesn't change the upper bound of Q, and the unification just generates some equalities, so the form of range will not be changed.

Therefore, the form of Q is $[\mathbf{now} - d, \mathbf{now}]$ in the step (a). □

Proposition 4. *Suppose* $R : C = (R_1 : C_1) \bullet (R_2 : C_2)$. *If* $\mathbf{now}_{R_1}$ *can unify with a time stamp* t_2 *in* R_2, *then, in constraint system of* R, *the lower bounds of time stamps from* R_1 *will extend a same interzone.*

Proof. Suppose the range of t_2 is $[\mathbf{now} - d, \mathbf{now}]$ in C_2, then $\mathbf{now}_{R_2} - d' \leq \mathbf{now}_{R_1} \leq \mathbf{now}_{R_2}$. For all $t_1 \in C_1$ and $\mathbf{now}_{R_1} - d \leq t_1 \leq \mathbf{now}_{R_1}$, there is $\mathbf{now}_{R_2} - d' - d \leq \mathbf{now}_{R_1} - d \leq t_1 \leq \mathbf{now}_{R_1} \leq \mathbf{now}_{R_2}$, so the low bound of t_1 extends d'. □

Proposition 5. *Suppose* $R : C = (R_1 : C_1) \bullet (R_2 : C_2)$, *if there is no time variable in the variable set of unification when* R_1 *resolves with* R_2, *then the constraints from* R_1 *could be deleted from* C.

Proof. Because of no unification between time stamps $t_1 \in var(R_1)$ and $t_2 \in var(R_2)$, time variables in R_1 have no relation with ones in R_2 except that $\max(Q)$ is changed by $\mathbf{now}_{R_1} \leq \mathbf{now}_{R_2}$. The greatest lower bound of time variables from R_1 is independent of $\mathbf{now}_R$ and have no influence with time constraints from R_2. Therefore, in the later rules, these constraints will always be satisfiable. So these constraints from R_1 could be deleted from the result of resolution, and the correctness of verification will not be changed. □

Proposition 6. *Suppose* $R_1 : C = (R_1 : C_1) \bullet (R_2 : C_2)$, *if there are* $t_1 \in var(R_1)$ *and* $t_2 \in var(R_2)$ *such that they can unify, and* $t_1 \neq \mathbf{now}_{R_1}$, *then there is no range contracted when* $R_1 : C$ *resolves with* $R_2 : C_2$ *in this left resolution fixpoint.*

Proof. Suppose $t_1, t_3 \in R_1$ and $\mathbf{now}_{R_1} - d_1 \leq t_1 \leq \mathbf{now}_{R_1}, \mathbf{now}_{R_1} - d_3 \leq t_3 \leq \mathbf{now}_{R_1}$, and $t_2 \in R_2$ such that $\mathbf{now}_{R_2} - d \leq t_2 \leq \mathbf{now}_{R_2}$, then $t_1 = t_2$ by the condition. The cases is discussed as follows:

(1) there is no constraint $t_1 - d' \leq t3$, then $\mathbf{now}_{R_2} - d \leq t_2 = t_1 \leq \mathbf{now}_{R_2}$, so $\mathbf{now}_{R_2} - d - d_3 \leq \mathbf{now}_{R_1} - d_3 \leq t_3 \leq \mathbf{now}_{R_2}$. Therefore the range of t_3 is extended a period d.
(2) there exists $t_1 - d' \leq t_3$, then the range of t_3 is decided by the relation between t_1 and t_3 since the changes of ranges of time variables in R_1 are brought by the change of the range of t_1.
$\mathbf{now}_{R_2} - d - d' \leq t_1 - d' \leq t_3$, but it is unknown who is greater than another one between $d + d'$ and d_3. However, when the result resolves with R_2, $\mathbf{now}_{R_2} - d - d' \leq t_1 - d' \leq t_3$ holds. So the range of t_3 will not be changed in this iterative left resolution fixpoint. □

Definition 7. *Suppose* $R_1 : C = (R_1 : C_1) \bullet (R_2 : C_2)$, *if the range of* $t_1 \in var(R_1)$ *in the process of* $(R_1 : C) \bullet (R_2 : C_2)$ *is unchanged, then* t_1 *is called stable in the resolution process* $(R_1 : C_1) \bullet (R_2 : C_2)$, *else it is called unstable.*

Proposition 7. *If the range of* t *extends a length* d *in the resolution process with the left(or right) resolution fixpoint* R, *from* $Q = [L, \mathbf{now}]$ *to* $Q' = [L - d, \mathbf{now}]$, *and* t *is unstable, then its range will extend the same length in the iterative resolution.*

Proof. We will give the proof about the left resolution fixpoint, and it is analogous to prove the other one.
By the proofs of propositions 4∼6, the process in which the range of t extends to $[\mathbf{now}-d-d', \mathbf{now}]$ from $[\mathbf{now}-d, \mathbf{now}]$ is independent with the possible value of t, and the relative positions among time variables in the constraint system of R_1 are unchanged. So the extended length will be the same in the next fixpoint resolution. □

No matter how time changes, the hypothesis of R_1 will lead to its result validly, that is, time stamps in R_1 couldn't keep the freshness of messages. So there is a theorem as follows:

Theorem 2. *If there is a time variable in the resolution fixpoint such that its range is extended and it is unstable, then the freshness of message in the protocol is flawed.*

Proof. Suppose the extended length is d in the process of fixpoint resolution, by the proposition 7, it will extend d in every iterative resolution. Since Δ is a finite value, there is a natural number n such that $n \times d > \Delta$. After n-iteration, $Q \supseteq [\mathbf{now}-\Delta, \mathbf{now}]$. By theorem 1, t couldn't keep the freshness of messages. □

Corollary 2. *If there is a time stamp t whose range is extended in fixpoint resolution, and it is unstable, then the constraints containing t can be deleted.*

Suppose $R : C = (R_1 : C_1) \bullet (R_2 : C_2)$, and θ is the most general unification in this resolution. By the discussion in this section, we have the following algorithm. Suppose the range of t in C_1 is $[t_{q1}, t_{q2}]$, and that in C is $[t_{p1}, t_{p2}]$. Let θ be the most general unifier of R_1 and R_2 and $var(R_1) \cap var(R_2) = \emptyset$.

Algorithm 1. *function abs_const(R:C, $R_1 : C_1$, $R_2 : C_2$, θ)*
(1) *if* $var(C_1) \cap \mathbf{dom}(\theta) = \emptyset$,
(2) *then* $C = C_2\theta$;
(3) *else*
(3.1) *for all* $t \in var(R_1)$ *do*
(3.1.2) *if* $t = \mathbf{now}_{R_1}, \mathbf{now}_{R_1}\theta = t_2\theta, t_2 \in R_2, \mathbf{now}_{R_2} - d' \leq t_2 \leq \mathbf{now}_{R_2}$ *then*
(3.1.3) *for each time variable* $t \in var(C_1)$ *do*
(3.1.3.1) *let* $t_{p1} = t_{q1} - d'$
(3.1.4) *if* $t = t_1 \in var(R_1), t_2 \in var(R_2), t_1 \neq now_{R_1}, t_1\theta = t_2\theta$, $\mathbf{now}_{R_2} - d \leq t_2 \leq \mathbf{now}_{R_2}$ *then*
(3.1.5) *for all* $t3 \in var(R_1), t3 \neq t_1$ *do*
(3.1.5.1) *if* $C_1 \Rightarrow (t_1 - d' \leq t3)$ *then* $t3_{p1} = t3_{q1} - d$.
(3.1.5.2) *else* $t3_{p1} = \max\{\mathbf{now}_{R_2} - (d + d'), \mathbf{now}_{R_2} - d_3\}$
(3.2) *let* $R : C_3 = R : C \bullet R_2 : C_2$
(3.3) *for all* $t \in C$ *do*
(3.3.1) *if the range of* t *in* $C_3 \neq$ *the range of* t *in* C *then*
(3.3.2) *mark(t)*
(3.4) $C \leftarrow C$ *without* t *marked by step (3.3).*
(4) *return R:C*

In step (3.3.2), $mark(t)$ is used to mark t deleted. In this algorithm, we delete all time variables which will not work in constraint, and extend some time variables' ranges. The algorithm is written according to the theory in this paper, so it is correct. Suppose $n_1 = |var(C_1)|, n_2 = |var(C_2)|, n = |var(C)|$. Step (1) costs $O(n_1 \times (n_1 + n_2))$. Step (2) costs $O(n_2)$. Step (3.1) needs to execute n_1 times, so do step (3.1.3) and step (3.1.5). Actually, the most cost is in step (3.1.5.1). The time cost in $C_1 \Rightarrow (t_1 - d' \leq t_3)$ is very high. It depends on the number m of inequalities in C_1. Step (3.2), (3.3.1), (3.3.2) and (3.4) cost $O(1)$, so the loop from (3.3) to (3.3.2) will cost $O(n)$. So the worst cost in this algorithm is $O(n_1^2 \times f(m))$, where $f(m)$ is the size of the cost in $C_1 \Rightarrow (t_1 - d' \leq t_3)$.

4 Verification of WMF Protocol in Abstracted Horn Model

The attacker can hold the session key after it is expired. Because of a decrypt rule(destruct rule) $decrypt(\{Secret\}_{xK}, xK) = Secret$ by $\mathbf{DY}_3$, there is a following attack sequence(the first column is the No. of the rule, the second is the rule, the last notes where the rule is generated from):

No.	Rule	Notes
A_1:	$attacker(\{xB, xK, xT_1\}Kxas) \rightarrow attacker(\{xA, xK, T_2\}Kxbs) :$ $\mathbf{now} - d \leq xT_1 < \mathbf{now}, T_2 = \mathbf{now}$	①③
A_2:	$attacker(\{xB, xK, xT_1\}Kxas) \rightarrow attacker(\{xA, xK, xT_2\}Kxbs) :$ $xT2 = \mathbf{now}$	A_1A_1
A_3:	$attacker(\{xB, xK, xT_1\}Kxas) \rightarrow attacker(\{Secret\}XK) : true$	$A_2$④
A_4:	$\rightarrow attacker(\{Secret\}K[T_1]) \rightarrow attacker(Secret) : true$	$A_3$②
A_5:	$attacker(K[T_1]) \rightarrow attacker(Secret) : true$	$A_4(1)$
A_6:	$\rightarrow attacker(Secret) : \mathbf{now} > \Delta + T_1$	DY_4A_5

A_2 is thought to have constraints $t_1 - d \leq xT_1 \leq t_1, \mathbf{now} - d \leq t_1 \leq \mathbf{now}$. However, because the range of xT_1 is extended and it is not stable, by corollary 2, the constraints containing xT_1 can be deleted. In rule A_6, the attacker gains the secret data $Secret$, and the constraints system is satisfiable, so the WMF protocol does not hold secret. The reason is that messages are fresh in any time by exchanging messages with the attacker.

In the Horn logic model without abstract constraint, we have a flaw sequence as follows.

(1) $attacker(\{xB, XK, XT1\}Kxas) \rightarrow attacker(\{xA, XK, T2\}Kxbs) : now - d \leq XT1 < now, T2 = now$ ①③

(2) $attacker(\{xB, XK, XT1\}Kxas) \rightarrow attacker(\{xA, XK, XT2\}Kxbs) : t_1 - d \leq XT1 < t_1, now - d \leq t_1 < now, XT2 = now$ (1)(1)

... .

(n+1) $attacker(\{xB, XK, XT1\}Kxas) \rightarrow attacker(\{xA, XK, XT2\}Kxbs) : t_n - d \leq XT1 < t_n, t_{n-1} - d \leq t_n < t_{n-1}, \cdots, t_1 - d \leq t_2 < t_1, now - d \leq t_1 < now, XT2 = now$ (1)(n)

This proceeding will not terminate automatically. Although the attack can be found with the attendance of human, it is very boring to the fact that, for any D, there exists an n(n=[D/d]) such that there is a set of the solution of constraints in the step n+1. The constraints in this proceeding are more complex than the abstract constraints, and the termination cannot be decided automatically. Therefore, the verification in abstract constraint is more effective than the model without abstract.

The verification method in [12] is based on the combination of constraint solving technology and first order term manipulation, and it uses Sicstus Prolog to solve constraints. This method need to compute symbolic reachability graph whose complex is higher than ours. [20] gives the result of verifying WMF protocol in the method of [12]: the back research procedure terminates in 3.3s after 9 steps computing a graph with 14 nodes. In our model, by abstracting constraints system, the verification also terminates and the number of the attack sequence is 11. Horn logic model is very effective in verification and it needs small time to find the result [17], so the Horn model with time constraints will be suitable for verification of complex protocols with time stamps. Table 3 gives the compare with related work.

Table 3. Comparison of results with related work

Verification Method	Termination	No. of Nodes/Rules
MSR	yes	14
Horn Logic with constraint	no	-
Horn Logic with abstract constraint	yes	11

5 Conclusion

The verification in [14, 15, 16] is very effective, and can verify infinite runs of protocols, but it cannot verify time sensitive security protocols for there is no time information in that model. [5] adds time constraints into Horn logic model, but it will not terminate automatically. We research the constraint system, and give a method of constraints abstraction. And then, the Horn logic with abstract constraint can verify WMF protocol very quickly and effectively. In this paper, we discuss relations between the constraint system and Horn logic model, abstract the constraint system, and give the proofs of some propositions and theorems. In the experiment, the attack of WMF protocol can be found in new model very quickly. Therefore, the method in this paper is a very effective one in verification of time sensitive security protocols.

[18] gives an automatic verification tool SPVT on Horn model which can verify security protocols and construct counter-examples very effective. The future work is to modify the tool SPVT with the method in this paper to verify time sensitive security protocols and construct counter-examples automatically. Complex protocols will be verified by this tool.

References

1. Anderson, R.J., Needham, R.M.: Programming Satan's computer. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 426–440. Springer, Heidelberg (1995)
2. Clark, J.A., Jacob, J.L.: A survey of authentication protocol literature. Technical Report 1.0 (1997)
3. Dolev, D., Yao, A.C.: On the security of public key protocols. Technical report, Stanford, CA, USA (1981)
4. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–208 (1983)
5. Zhou, T., Li, M., Li, Z., Chen, H.: Verification of time sensitive security protocols based on the extended Horn logic model. Chinese Journal of Computer Research and Development 43(suppl.2), 534–540 (2006)
6. Lowe, G.: Casper: A compiler for the analysis of security protocols. In: 10th IEEE Computer Security Foundations Workshop (CSFW-10), pp. 18–30 (1997)
7. Lowe, G.: A hierarchy of authentication specifications. In: 10th IEEE Computer Security Foundations Workshop (CSFW-10), pp. 31–44 (1997)
8. Evans, N., Schneider, S.: Analysing time dependent security properties in CSP using PVS. In: ESORICS, pp. 222–237 (2000)
9. Gorrieri, R., Locatelli, E., Martinelli, F.: A Simple Language for Real-Time Cryptographic Protocol Analysis. In: Degano, P. (ed.) ESOP 2003 and ETAPS 2003. LNCS, vol. 2618, pp. 114–128. Springer, Heidelberg (2003)
10. Gorrieri, R., Martinelli, F.: A simple framework for real-time cryptographic protocol analysis with compositional proof rules. Sci. Comput. Program. 50(1-3), 23–49 (2004)
11. Corin, R., Etalle, S., Hartel, P.H., Mader, A.: Timed model checking of security protocols. In: FMSE 2004: Proceedings of the 2004 ACM workshop on Formal methods in security engineering, pp. 23–32. ACM Press, New York (2004)
12. Delzanno, G., Ganty, P.: Automatic Verification of Time Sensitive Cryptographic Protocols. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 342–356. Springer, Heidelberg (2004)
13. Bozga, L., Ene, C., Lakhnech, Y.: A Symbolic Decision Procedure for Cryptographic Protocols with Time Stamps. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 177–192. Springer, Heidelberg (2004)
14. Abadi, M., Blanchet, B.: Analyzing security protocols with secrecy types and logic programs. In: Symposium on Principles of Programming Languages, pp. 33–44 (2002)
15. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: 14th IEEE Computer Security Foundations Workshop (CSFW-14), pp. 82–96 (2001)
16. Blanchet, B.: From Secrecy to Authenticity in Security Protocols. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 342–359. Springer, Heidelberg (2002)
17. Li, M., Li, Z., Chen, H.: Security protocol's extended Horn logic model and its verification method. Chinese Journal of Computers 29(9), 1667–1678 (2006)
18. Li, M., Li, Z., Chen, H.: Spvt: An efficient verification tool for security protocol. Chinese Journal of Software 17(4), 898–906 (2006)
19. Li, Z., Zhou, T., Li, M., Chen, H.: Constraints Solution for Time Sensitive Security Protocols. In: Preparata, F.P., Fang, Q. (eds.) FAW 2007. LNCS, vol. 4613, pp. 191–203. Springer, Heidelberg (2007)
20. Delzanno, G.: Automatic of secrecy and authentication in unbound models of cryptographic protocols with fresh nonce generation and time-stamps(draft) (2003)

Fast Convergence of Variable-Structure Congestion Control Protocol with Explicit Precise Feedback

Huixiang Zhang, Guanzhong Dai, Lei Yao, and Hairui Zhou

College of Automatic, Northwestern Polytechnical University, Xi'an, China
hxiang.zhang@gmail.com

Abstract. Traditional TCP has significant limitations such as unclear congestion implication, low utilization in high bandwidth delay product networks, unstable throughput and limited fairness. In order to overcome such limitations, research on design and development of more effective congestion control algorithms, especially in the high bandwidth delay product networks, is very active. Variable-structure congestion Control Protocol (VCP) uses two ECN bits to deliver the bottleneck link utilization region to end systems, and achieves high utilization, low persistent queue length, negligible packet loss rate and reasonable fairness. Owing to the utilization of large multiplicative decrease factor, VCP flows need very long time to finish fairness convergence. To address this problem, a new method called VCP-Fast Convergence (VCP-FC) is proposed in this paper. VCP-FC uses more bits to deliver precise network load factor back to end systems. The end system calculates the fairness bandwidth based on the variance rates of the load factor and throughput, and then quickly adjusts the congestion window to approach the fairness bandwidth. VCP-FC shortens the fairness convergence time effectively, and meanwhile improves the efficiency and fairness of VCP. At last, the performance of VCP-FC is evaluated using ns2 simulations.

1 Introduction

It is well-known that the Addictive Increase Multiplicative Decrease (AIMD)[1] congestion control algorithm employed by traditional TCP[2] doesn't perform well in high bandwidth delay networks. The research on design and development of more effective congestion control algorithms, especially in the high bandwidth delay product networks, is very active. One direction is pure end-to-end improvement, such as High-speed TCP[3], LTCP[4], Fast TCP[5] and Scalable TCP[6]. These algorithms increase congestion window aggressively and decrease conservatively to improve the network utilization. Another approach is to utilize explicit feedback from internet routers, such as VCP[7],RCP[8],ACP[9],XCP[10]. These algorithms redesign the internet to achieve high utilization, low persistent queue length, negligible packet loss rate and max-min fairness.

Among the algorithms utilizing explicit feedback, Variable-structure congestion Control Protocol (VCP) is a simple and low complexity protocol that modifies mainly in end systems and deploys easier than XCP. VCP is able to achieve

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 264–275, 2008.

comparable performance to XCP but converges significantly slower to the fair bandwidth than XCP. When a new flow joins the network of high utilization, the existing flows can't decrease occupied bandwidth fast owing to VCP large MD factor β (0.875), the new flow converges to the fair allocation very slowly.

There are several relative researches on improving convergence speed to fairness in the literature. [11] improves the convergence speed of High-Speed TCP to the fair bandwidth. The proposed mechanism checks the congestion window size just before a loss event. If the size continuously declines, the window is assessed to be on a downward trend. Once the difference between the maximum and minimum values of the checked size during the current downward trend exceeds a threshold, the congestion window decrease parameter is set larger than usual. Thus, flows with a larger window size than fair can decrease their window more aggressively to improve the convergence times. But the condition described above is an very special case in the fast changing network environment, so the mechanism isn't an general solution. [12] presented a congestion control algorithm based on the traditional TCP. The algorithm explicitly calculates the fair share and converges to it in two congestion cycles in a distributed fashion. But owing to the limitation of TCP, only synchronous flows is analyzed in [12].

To improve VCP convergence speed to fairness, a new method called VCP-Fast Convergence (VCP-FC) is proposed in this paper. VCP-FC use more bits to deliver precise network load factor back to end systems. End systems estimate the fair bandwidth based on the variance rates of the load factor and throughput. If current bandwidth deviates from the fair bandwidth, the flow rapidly adjusts its congestion window to the fair bandwidth, which improves the convergence speed to fairness. If current bandwidth approaches the fair bandwidth, the flow apply lager MD factor (0.9) than VCP so as to smooth the flow's throughput and meanwhile improves the network utilization. The additional benefit of using precise feedback is to improve VCP convergence speed to efficiency in MI stage.

The rest of the paper is organized as follows: Section 2 briefly reviews VCP and analysis its convergence behaviors. Section 3 elaborates VCP-FC. Section 4 uses ns2 simulations to evaluate the performance of VCP-FC. Finally, conclusions and future works are provided in Section 5.

2 Variable-Structure Congestion Control Protocol (VCP)

The VCP router calculates the load factor ρ periodically:

$$\rho = \frac{\lambda + \kappa q}{\gamma C t_\rho} \tag{1}$$

Here t_ρ is the calculation interval. Owing to 75% $\sim$ 90% of flows have RTTs less than 200 ms [13], VCP set t_ρ=200 ms. λ is the amount of input traffic during the last interval t_ρ. q is the persistent queue length during the last interval t_ρ. κ controls how fast the persistent queue drains and set $\kappa = 0.5$. γ is the target utilization and set $\gamma = 0.98$. C is the link bandwidth.

In every interval t_ρ, the link utilization is classified into three regions based on the load factor ρ. If $0 \leq \rho < 80\%$, the link utilization is classified as low-load region; if $80 \leq \rho < 100\%$, the link utilization is classified as high-load region, if $\rho \geq 100\%$, the link utilization is classified as overload region. VCP routers encode the utilization regions into two ECN bits in the IP header of each data packet. This information is then sent back by the receiver to the sender via ACK packets. Depending on the utilization regions, the sender applies different congestion response. If in low-load region, the sender increases its sending rate using MI to improve the link utilization quickly; if in high-load region, the sender increases its sending rate using AI to improve the link utilization slowly; if in overload region, the sender decreases its sending rate using MD immediately. The respective response functions are as follows:

$$MI : cwnd(t + rtt) = cwnd(t) \times (1 + \varepsilon) \tag{2}$$

$$AI : cwnd(t + rtt) = cwnd(t) + \alpha \tag{3}$$

$$MD : cwnd(t + rtt) = cwnd(t) \times \beta \tag{4}$$

Where $\varepsilon = 0.0625$, $\alpha = 1$, $\beta = 0.875$. To offset the impact of the RTT heterogeneity, VCP scales ε and α using equation (5)(6) respectively according to their RTTs. And further, in order to allocate the bandwidth in fairness, VCP uses equation (7) adding an additional scaling factor to the AI algorithm:

$$\varepsilon_s = (1 + \varepsilon)^{\frac{rtt}{t_\rho}} - 1 \tag{5}$$

$$\alpha_s = \alpha \frac{rtt}{t_\rho} \tag{6}$$

$$\alpha_{rate} = \alpha_s \frac{rtt}{t_\rho} = \alpha (\frac{rtt}{t_\rho})^2 \tag{7}$$

The behavior of VCP convergence could be divided into two stages. Stage one is convergence to efficiency. VCP flows quickly take the available bandwidth using MI. The link utilization ramps up to 80% quickly, which shows VCP has high efficiency. Stage two is convergence to fairness. VCP flows achieve to the fair bandwidth using AIMD. VCP doesn't guarantee fairness in stage one. Owing to the scaling of α_s, the same time started flows but of different RTTs will converge to the same congestion window in stage one, which also means that flows converge to unfair bandwidth. Only in stage two these flows of different sending rate converge to fairness. Owing to the scaling of α_s, flows of different RTTs increase their congestion window equally every t_ρ interval. Owing to the scaling of α_{rate}, flows of different RTTs increase their bandwidth equally every t_ρ interval. To prevent the system from oscillating between MI and MD, VCP set the MD factor $\beta = 0.875$. As only the MD function can affect the convergence time to fairness, the choice of 0.875 is the root cause of slow convergence to fairness.

The behavior that VCP flows converge to fairness slowly manifest in two aspects. Firstly, the same time started flows but of different RTTs need long

time to converge to the fairness bandwidth. Figure 1 shows the convergence of congestion window of two VCP flows with different RTTs (20 ms and 100 ms respectively). Two flows start to send packets at $t = 0$. The bottleneck bandwidth is 100Mbps. In stage one, i.e. convergence to efficiency, two flows converge to the identical congestion window value. Then the flows converge to the fairness bandwidth using AIMD in stage two. The total convergence time is about 300 seconds. Secondly, when the link utilization is in high-load region, i.e. the load factor is between 80% and 100%, one new flow joins and starts sending packets; the new flow needs long time to converge to the fairness bandwidth. As shown in Fig.2, one flow starts firstly and achieves the stable state, then the other new flow starts to send packets at $t = 100s$. The two flows have identical RTT of 80 ms. The bottleneck bandwidth is 100Mbps. The new flow needs about 300 seconds to converge to the fairness bandwidth.

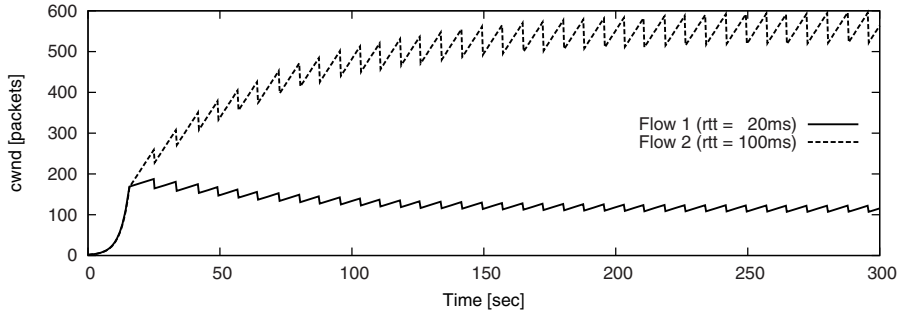


Fig. 1. Two VCP flows of different RTTs start simultaneously

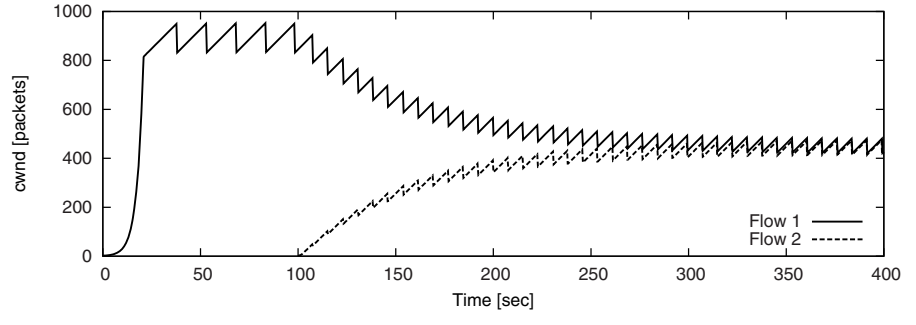


Fig. 2. Two VCP flows of identical RTTs start sequentially

3 VCP Fast Convergence (VCP-FC)

VCP-FC keeps the algorithms of VCP router unchanged and just uses ten bits to deliver the load factor back to end systems. The load factor ρ is represented using ten bits, that is to say the quantized load factor has the precision of 0.001. With

the quantized load factor, end systems can improve the convergence speed to efficiency and fairness. The following subsections will describe them separately.

3.1 Convergence to Efficiency

In the stage of convergence to efficiency, $0 \leq \rho < 80\%$. Using the precise load factor, VCP-FC can adjust the congestion window more quickly and accurately. VCP-FC substitutes the VCP MI response function with equation (8):

$$cwnd(t + rtt) = cwnd(t) + \mu(1 - \rho(t))cwnd(t) \tag{8}$$

Where $\rho(t)$ is the load factor at time of t. μ controls how fast the convergence to efficiency; in order to guarantee the robustness, we choose $\mu = 0.5$. VCP-FC scales $\rho(t)$ as VCP does in equation (5) and just replaces ε with $\rho(t)$. At the end of convergence to efficiency, simultaneously started flows can reach the same value of congestion window. Using the new MI response function, the convergence speed to efficiency is improved effectively.

Suppose there is a single bottleneck with bandwidth of C shared by multiple flows. The flows start to send packets simultaneously. Assuming the flows have identical RTTs and start from the unit aggregate rates r(0)=1. The flows converge to efficiency using MI. For VCP flows, [7] proofs that the aggregate rate after n rounds of MI is $r(n) = r(0)(1+\varepsilon)^n$, where $\varepsilon = 0.0625$; then at the end of convergence to efficiency, VCP flows need $\frac{\log(0.8C)}{\log(1+\varepsilon)}$ rounds of MI. For VCP-FC flows, the aggregate rate after n rounds of MI is $r(n) = r(0)\prod_{i=1}^{n}(1 + \mu(1 - \rho_i))$, where $0 \leq \rho_i < 0.8$ and $\mu = 0.5$, so $\mu(1 - \rho_i) > 0.1 > \varepsilon$. Apparently the number of MI rounds VCP-FC flows needed is less than $\frac{\log(0.8C)}{\log(1+0.1)}$. Compared with VCP flows, VCP-FC flows converge to efficiency faster than VCP flows.

3.2 Convergence to Fairness

In the stage of convergence to fairness, $\rho \geq 80\%$. VCP flows converge to fairness using AIMD. Additive increase doesn't affect the fairness among flows. The convergence speed to fairness is determined by the MD factor, i.e. β. The smaller the MD factor, the faster approaching the fairness; but the oscillation is higher too. To prevent the system oscillation between MI and MD, VCP set $\beta = 0.875$. To decrease the MD factor to improve the convergence speed to fairness isn't a good solution.

VCP-FC delivers the load factor back to end systems. Thus, the fairness bandwidth is able to estimate using the variance rates of the load factor and throughput in end systems. Suppose there is a single bottleneck with bandwidth of C shared by multiple flows. Consider an situation which there are N flows existing and no flows join or leave during a period of ΔT. In the stage of convergence to fairness, all flows update their congestion window using AIMD. When the bottleneck link is in high-load region, the aggregate incremental amount of bandwidth during ΔT is : $\sum_i^n (r_i(t+\Delta T) - r_i(t))$. Owing to additive increase, all flows increase their bandwidth by the same amount, denote as Δr, so we have:

$$\sum_{i}^{n}(r_i(t+\Delta T)-r_i(t))=N\Delta r \tag{9}$$

Assuming $\Delta T > t_\rho$, the incremental amount of the bottleneck utilization is $\Delta u = u(t+\Delta T)-u(t)$, we have:

$$\Delta u=\frac{\sum_i^n(r_i(t+\Delta T)-r_i(t))}{C}=\frac{N\Delta r}{C} \tag{10}$$

Thus, We can calculate the fairness bandwidth F as follow:

$$F=\frac{C}{N}=\frac{\Delta r}{\Delta u} \tag{11}$$

And further we can obtain the fairness congestion window W as follow:

$$W=F\bullet rtt=\frac{\Delta w}{\Delta u} \tag{12}$$

The VCP is able to achieve very low persistent queue length, thus the change of utilization Δu is approximately substituted with the change of load factor $\Delta\rho$ in end systems.

Suppose the network reaches congestion at some point, due to addictive increase. Then all flows will decrease their bandwidth multiplicative, and then resume addictive increase until the network congests again. We choose the interval between sequent congestion points as ΔT to estimate the fairness bandwidth more accurately. Figure 3 elaborates this interval. There are $m > 1$ rounds of AI and one round of MD in every ΔT interval. The bottleneck utilization is changing every t_ρ interval. Thus, in each ΔT interval we can obtain n pairs of value denote as (u_i, w_i), where u_i represent the load factor every t_ρ interval; w_i represent the value of congestion window when u_i is feed back to the end system. At the time of $T+\Delta T$, end systems calculate the fairness bandwidth using equation (13). End systems take (u_n, w_n) as the base value and calculate n-1 values of the fairness congestion window. Then end systems calculate the average of these values, so we obtain the fairness congestion window W_f in this ΔT interval:

$$W_f=\frac{1}{n-1}\sum_{i=1}^{n-1}\frac{w_n-w_i}{u_n-u_i} \tag{13}$$

The end systems estimate one new W_f every ΔT interval, then smooth the value as follow:

$$W_e^f=(1-\theta)last\_W_e^f+\theta W_f \tag{14}$$

In order to track the quickly changed network environment we choose larger value of θ and set $\theta = 0.4$.

End systems compare current congestion window with W_e^f, if $|cwnd-W_e^f| < \eta W_e^f$, where $\eta = 0.1$, that means the two variable is approximately equally, flows approach the fairness approximately. Then we can use larger MD factor to

smooth the flow throughput and set $\beta = 0.9$; otherwise we use following equation to update the congestion window:

$$cwnd_{new} = (1 - \omega)cwnd + \omega W_e^f \tag{15}$$

where $\omega = 0.2$. In some situation, we can't obtain enough samples, if $n < 5$, VCP-FC set $\beta = 0.875$ as original VCP.

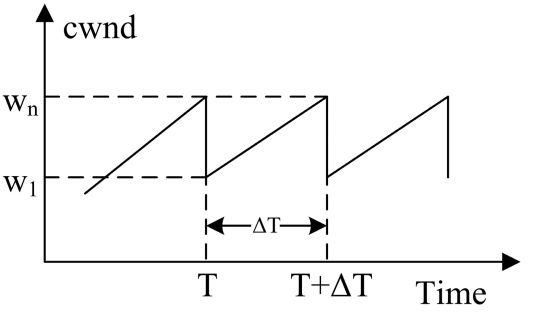


Fig. 3. Interval of ΔT

4 Simulations and Results

We incorporated our algorithm into VCP and validated its performance on NS-2[14]. The performance of VCP-FC is compare with VCP and XCP. We use the single congested link topology shown in Fig.4, where S_i is sending packets to D_i. We evaluate the convergence time to efficiency firstly, and then evaluate the convergence time to fairness when varied fairness bandwidth and round-trip times separately. At last, we study the performance of VCP-FC in an RTT heterogeneity environment and in an dynamic environment. We use FTP as the application layer data generator in all simulations. The data packet size is set to 1KBytes. The parameters of VCP and XCP is set according to the authors' recommendations in [7] and [10] separately.

Simulation results for convergence to efficiency: The bottleneck bandwidth varies from 2Mbps to 1Gbps. Only a single flow starts to send packets at $t = 0$, and its RTT=80 ms. The convergence time to efficiency is defined as how much time needed the bottleneck utilization reaches 80%. The result is show in Fig.5, and the x-axis is in logarithmic scale. From Fig.5, XCP converges to efficiency fastest. The convergence time to efficiency of VCP-FC is shorter than VCP regardless of the bandwidth. When the bandwidth varied from 2Mbps to 1Gbps, the convergence time to efficiency of VCP-FC increases slower than VCP. The result shows the VCP-FC flows converge faster to efficiency than VCP.

Simulation results for converge to fairness with varied fairness bandwidth: One flow starts to send packets at $t = 0$ and reaches the stable situation. Then the other flow starts to send packets. Two flows have identical RTT

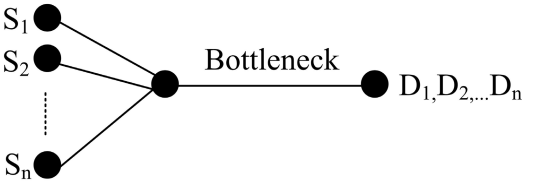


Fig. 4. A single bottleneck topology

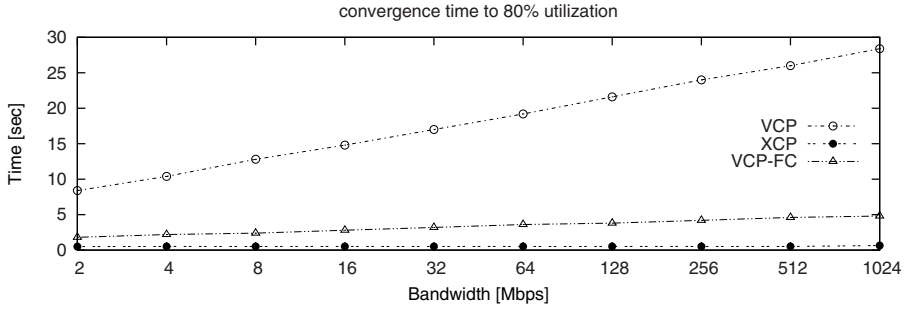


Fig. 5. Convergence time to efficiency versus bandwidth

of 80ms. The bottleneck bandwidth varies from 2Mbps to 1Gbps, which means the fairness bandwidth varies from 1Mbps to 512Mbps. The convergence time to fairness of the second flow is measured using the metric of $\delta - fair$ convergence time proposed in [15]. The metric is defined as the time taken for the second flow converges to $\frac{1-\delta}{2}$ of the link bandwidth. Here we set $\delta = 0.1$. As shown in Fig.6, XCP converges to the fairness bandwidth very fast and hardly affect by the fairness bandwidth. And the convergence time to fairness of VCP-FC is almost the same as VCP when the fairness bandwidth is less than 2Mbps. When the bandwidth increased, the increment of convergence time of VCP-FC is much less than VCP. The result shows VCP-FC significantly improve the convergence speed to fairness in high bandwidth environment, but still consume more time than XCP.

Simulation results for converge to fairness with varied RTT: The bottleneck bandwidth is fixed at 45Mbps. One flow starts to send packets at $t = 0$ and reaches the stable situation. Then the other flow starts to send packets. The RTT of the two flows is varied from 20ms to 200ms. The convergence time to fairness of the second flow is measured using $\delta - fair$ convergence time as the preceding simulation. As shown in Fig.7, the impact of RTT is rather slight to convergence time of VCP-FC, VCP and XCP. As the RTT grows, the convergence time increases very slowly. The convergence time of VCP-FC is smaller than VCP regardless of RTT, and XCP converges fastest to fairness.

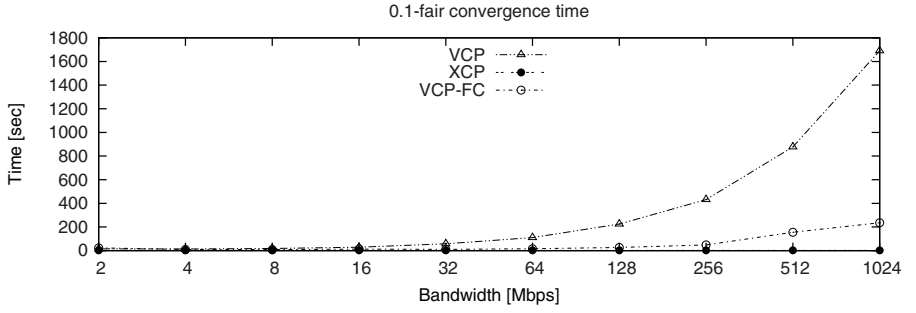


Fig. 6. Convergence time to fairness versus bandwidth

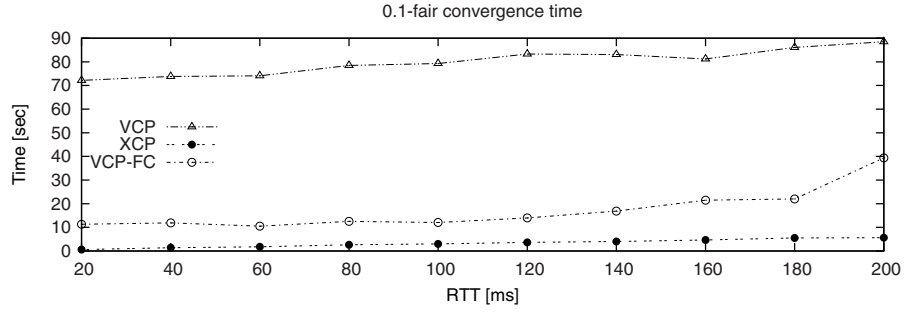


Fig. 7. Convergence time to fairness versus RTT

Simulation results for RTT heterogeneity environment: VCP flows can achieve max-min fairness in some extent. VCP-FC preserves the good property of VCP. We have 20 flows sharing the bottleneck link, and the bottleneck bandwidth is fixed at 200Mbps. We perform four sets of simulations: (a) the same RTT of 20ms; (b) small RTT difference from 20ms to 96ms; (c) large RTT difference from 20ms to 153ms; (d) huge RTT difference from 20ms to 229ms. We measured flows throughput in equilibrium. As shown in Fig.8, VCP-FC is able to allocate bandwidth fairly among competing flows, as long as their RTTs are not significantly different. With the RTT heterogeneity increases, the fairness of VCP-FC is degrade.

Simulation results for dynamic environment: In an dynamic environment, flows usually join or leave the network in an unpredictable manner. When flows leave, the available bandwidth is increased. VCP is able to take the available bandwidth by MI action quickly, and so does VCP-FC. when flows join, the contention of flows become intensive. The existing flows should decrease their bandwidth to make room for new flows. Here we focus on the effect of increased contention. We have 10 flows sharing a 400Mbps bottleneck. At t=60s, 110s, 160s, there are 10 flows join the network respectively. All flows have identical RTT of 80ms. The simulation last for 210s. Thus, the duration is divided into

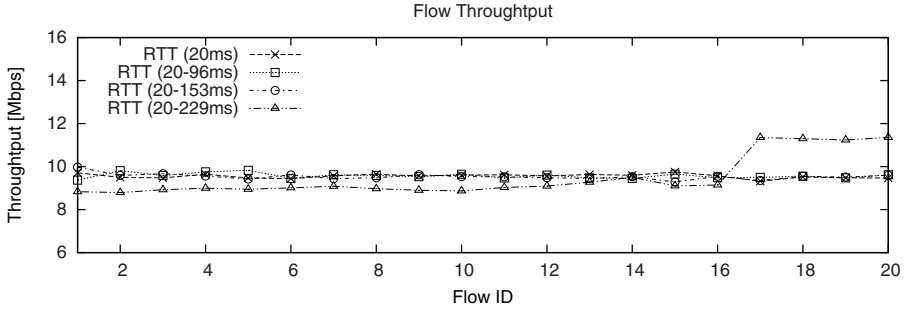
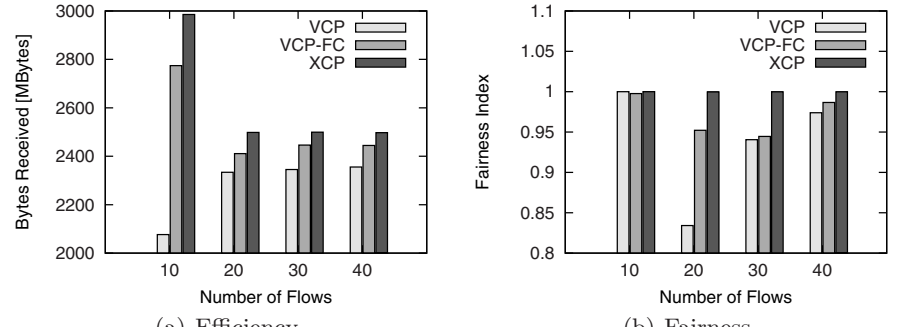


Fig. 8. Flow throughput in an RTT heterogeneity environment

four parts, which is 0-60s, 60-110s, 110-160s, 160-210s. We measured the efficiency and fairness in each part. The efficiency is measured using the goodput which is the bytes received in receivers. The fairness is measured using Fairness Index presented in [16]: $F(x) = (\sum x_i)^2/n(\sum {x_i}^2)$, where x_i is the goodput achieved by each flow. As show in Fig.9, XCP outperforms VCP and VCP-FC both in efficiency and fairness when graduated contention increased. In the first part of the duration, the efficiency of VCP-FC is much better than VCP, which also means VCP-FC converge to efficiency faster than VCP. In all parts of the duration, VCP-FC outperforms VCP in efficiency and fairness.



(a) Efficiency (b) Fairness

Fig. 9. Performance for graduated contention increase

5 Conclusion

This paper has proposed a new mechanism called VCP-FC for improving the convergence speed of VCP to efficiency and fairness. VCP-FC uses ten bits to deliver the load factor back to end systems. With the support from routers, VCP-FC can estimate the fairness bandwidth every congestion cycle. Simulation

results show that VCP-FC is valid in stationary and dynamic environment, and also effective in the RTT heterogeneity environment.

VCP-FC improves the efficiency and fairness of VCP effectively, but still not as good as XCP. The weak point of VCP-FC manifest mainly in two aspects. Firstly, the convergence time to fairness is affected by the fairness bandwidth, which is determined by the AIMD algorithms employed by VCP-FC. Secondly, the router calculates the load factor every t_ρ interval ($t_\rho = 200ms$). If the RTT of flows is beyond 200ms, the fairness of VCP and VCP-FC becomes worse. In such large RTT variance environment, the validity of VCP-FC needs further research.

References

1. Chiu, D., Jain, R.: Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. Computer Networks and ISDN Systems 17(1), 1–14 (1989)
2. Jacobson, V.: Congestion Avoidance and Control. ACM SIGCOMM Computer Communication Review 25(1), 157–187 (1995)
3. Floyd, S.: RFC3649: HighSpeed TCP for Large Congestion Windows. Internet RFCs (2003)
4. Bhandarkar, S., Jain, S., Reddy, A.: Improving TCP Performance in High Bandwidth High RTT Links Using Layered Congestion Control. In: The 3rd International Workshop on Protocols for FAST Long-Distance Networks (2005)
5. Wei, D., Jin, C., Low, S., Hegde, S.: FAST TCP: Motivation, Architecture, Algorithms, Performance. IEEE/ACM Trans. Networking 16(6), 1246–1259 (2006)
6. Kelly, T.: Scalable TCP: Improving Performance in Highspeed Wide Area Networks. ACM SIGCOMM Computer Communication Review 32(2), 83–91 (2003)
7. Xia, Y., Subramanian, L., Stoica, I., Kalyanaraman, S.: One More Bit Is Enough. In: The 2005 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 37–48. ACM Press, New York (2005)
8. Dukkipati, N., Kobayashi, M., Rui, Z., McKeown, N.: Processor Sharing Flows in the Internet. In: The 13th International Workshop on Quality of service, pp. 286–297. Springer, Berlin (2005)
9. Lestas, M., Pitsillides, A., Ioannou, P., Hadjipollas, G.: Adaptive congestion protocol: A congestion control protocol with learning capability. Computer Networks 51(13), 3773–3798 (2007)
10. Katabi, D., Handley, M., Rohrs, C.: Congestion Control for High Bandwidth-Delay Product Networks. In: The 2002 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 89–102. ACM Press, New York (2002)
11. Nabeshima, M., Yata, K.: Improving the convergence time of highspeed TCP. In: The 12th IEEE International Conference on Networks, pp. 19–23. IEEE press, New York (2004)
12. Attie, P., Lahanas, A., Tsaoussidis, V.: Beyond AIMD: Explicit Fair-share Calculation. In: The 8th IEEE International Symposium on Computers and Communications, pp. 727–734. IEEE press, Washington (2003)
13. Jiang, H., Dovrolis, C.: Passive Estimation of TCP Round-Trip Times. ACM Computer Communications Review 32(3), 75–88 (2002)

14. The network simulator ns-2.30, http://www.isi.edu/nsnam/ns
15. Bansal, D., Balakrishnan, H., Floyd, S., Shenker, S.: Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms. In: The 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 263–274. ACM Press, New York (2001)
16. Jain, R., Chiu, D., Hawe, W.: A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Systems. Technical Report TR-301, Digital Equipment Corporation (1984)

Applying a New Grid-Based Elitist-Reserving Strategy to EMO Archive Algorithms

Jiongliang Xie, Jinhua Zheng*, Biao Luo, and Miqing Li

Institute of Information Engineering, Xiangtan University,
Xiangtan, Hunan, China 411105
bright1228@sohu.com, jhzheng@xtu.edu.cn
biao.Luo@Hotmail.Com, limit1008@126.Com

Abstract. Grid-based measure is an often-used strategy by some MOEAs to maintain the diversity of the solution sets. The well known ε-MOEA, based on the ε-dominance concept, is essentially based on grid-strategy too. Though often gaining an appropriate tradeoff between the aspects of the performance, the ε-MOEA has its inherent vice and behaves unacceptably sometimes. That is, when the PF_{true}'s slope to one dimension changes a lot along the coordinate, the algorithm loses many extreme or representative individuals, that has obvious influence on the diversity of the solution sets. In order to solve this problem, a new δ-dominance concept and the *suppositional optimum point* concept are defined. Then we proposed a new grid-based elitist-reserving strategy and applied it in an EMO archive algorithm (δ-MOEA). The experimental results illustrated δ-MOEA's good performance, which is much better especially for the diversity than NSGA-II and ε-MOEA.

Keywords: Grid, Archive set, ε-dominance, δ-dominance, *suppositional optimum point*, Grid-based Elitist-reserving Strategy, δ-MOEA.

1 Introduction

MOEAs(Multi-Objective Evolutionary Algorithms) have the ability to detect interesting solution candidates for multi-objective optimization problems[1][2], that enables the decision maker to filter efficient solutions and to discover trade-offs between opposing objectives among these solutions.

In practice, the decision maker wishes to evaluate only a limited number of Pareto-optimal solutions. This is due to the limited amount of time for examining the applicability of the solutions to be realized in practice. Hence, how to gain a solution set with good distribution is an important pursuing goal for the MOEA designers. Typically the satisfying solution set should include extreme individuals as well as the ones that are located in important parts of the solution space, where balanced trade-offs can be found.

More than several methods based on grid or hyper-volume measure are used by MOEAs as selection strategy to maintain diversity[3]. The well-known ε-MOEA[5]

* Corresponding author.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 276–287, 2008.

proposed by Deb et is essentially based on the grid measure too. When deciding whether a new generated individual to be reserved in the archive or not, ε-MOEA doesn't employ the general Pareto domination concept[6] but uses the ε- dominance concept instead. Because of the weaker domination relationship than the generally used one, ε-dominance may make the domination relationship between two individuals come into being, though they have no similar relation according to the Pareto domination concept. Furthermore, ε-MOEA just allows only one individual occupying each grid, hence the algorithm converges quickly and the archive set has good diversity. However, the ε-dominance has its inherent vice that when the true PF of the problem has quite discrepant value of slope in different portion of one dimension, some extreme or important representative individuals are lost. Though one can relieve this losing-phenomenon by adjusting the value of ε, the region used to having moderate number of solutions may contain too many individuals, so it can't solve this matter fundamentally.

In order to solve this problem, we defined a new δ-dominance concept, which kept the merit of ε-dominance down but avoided the important individual losing-phenomenon. Then we applied it in the elitist-reserving strategy to update the archive set, which made the new algorithm (δ-MOEA) gain solution sets with better diversity than that by other ones(ε-MOEA and NSGA-II).

The rest of this paper is organized as follows. Section 2 briefly introduces some relating definitions of multi-objective optimization problem. Section 3 explicates the ε-domination and ε-MOEA. Section 4 presents the proposed δ-dominance concept, the new elitist-reserving strategy and δ-MOEA. Section 5 shows experiment results and discussions. Finally, Section 6 concludes with a summary of the paper.

2 Basic Concepts

Definition 1 ***(Multi-objective Optimization Problem(MOP)).*** *A general MOP is described as following:*

$$\text{Min } \vec{f}(X) = (f_1(X), f_2(X), \cdots, f_r(X)) \tag{1}$$

$$g_i(X) \geq 0; (i = 1, 2, ..., k) \tag{2}$$

$$h_i(X) = 0; (i = 1, 2, ..., l) \tag{3}$$

where $\vec{f}(X)$ *is the objective vector,* $\boldsymbol{r}$ *is the dimension of objectives, (2) and (3) are equality-constraints and inequality-constraints,* $X = (x_1, x_2, ..., x_n)$ *is variable vector,* $\boldsymbol{n}$ *is the dimension of variables,* $X \in \Omega$, $\Omega \subseteq R^n$ *,where* Ω *is the feasible space, then,* $\vec{f} : \Omega \to \Pi$, $\Pi \subseteq R^r$, Π *is the objective space.*

Definition 2 (*Pareto dominance*). *A solution* $\boldsymbol{x^0}$ *is said to dominate (Pareto optimal) another solution* $\boldsymbol{x^1}$ *(denoted* $\boldsymbol{x^0} \succ \boldsymbol{x^1}$*) if and only if:*
$\forall i \in \{1,..,m\} : f_i(x^0) \leq f_i(x^1) \bigcap (\exists k \in \{1,..,m\} : f_k(x^0) < f_k(x^1))$.

Definition 3 (*Pareto optimal*). *A solution* $\boldsymbol{x^0}$ *is said to be non-dominated (Pareto optimal) if and only if:* $\overline{\exists}\, x^1 \in X : x^1 \succ x^0$.

Definition 4 (*Pareto optimal set*). *The set P_S of all Pareto optimal solutions:* $P_S = \{x^0 \mid \exists\, x^1 \in X : x^1 \succ x^0\}$.

Definition 5 (*Pareto optimal front*). *The set PF of all objective function values corresponding to the solutions in PS:* $P_F = \{f(x) = (f_1(x), \ldots, f_m(x)) \mid x \in P_S\}$.

The optimal result for such multi-objective optimization is no other than the Pareto optimal set PS. However, the size of this set may be infinite, and it is impossible to find this set by using a finite number of solutions. In this case, a representative subset of PS is desired. Generally, the characteristic of MOEAs is to search the decision space by maintaining a finite population of individuals (corresponding to the points in the decision space), which work according to the procedures that resemble the principles of natural selection and evolution. Because we only consider the subset of all the final non-dominated individuals resulted from a MOEA, we call such subset an approximation set and denote it by S, and we call the corresponding objective set a resulting final Pareto optimal front and denote it by PF_{final}. Ideally, we are interested in finding an S of finite size, which contains a selection of individuals from such that the individuals in PF_{final} are diversified as possible. Unfortunately, we usually have no access to PF on beforehand. However, it is common practice to search for a good diversity of the individuals in the objective space because decision makers will ultimately have to pick a single individual as final solution according to its objective vector values. Therefore, it is often best to present a wide variety of tradeoff individuals for the specified goals in constructing MOEAs.

3 Grid-Measure and ε-MOEA

3.1 Generality of Grid-Measure

Generally, the grid-measure divides the objective space into a lot of small grids or hyper-cubes and the size of them usually depends on the value of the objective function. If two individuals are located in the same grid, the difference on each dimension is tolerant and neglectable for the problem.

3.2 The ε-Dominance Concept

Definition 6[4] (*ε-dominance*). *Let f, $g \in \mathbb{R}^{+m}$. Then f is said to ε-dominate g for some ε>0, denoted as $f \succ_\varepsilon g$, if and only if for all $i \in \{1, \ldots, m\}$ (maximizing):*

$$(1+\varepsilon) \cdot f_i \geq g_i \tag{4}$$

When the problem is a *Minimizing*-formulated one, the above inequality formulation (4) should be simply modified.

In the divided objective space, each grid should be endued to ascertain which location of the space that an individual is distributed. And for each candidate solution in the archive set, an identification vector $\boldsymbol{B}$ ($\boldsymbol{B} = (B_1, B_2, \ldots B_m)^{\mathrm{T}}$) is assigned to identify

its location and its ε-dominating area. The identification vector is assigned according to the following formulation:

$$B_j(f) = \begin{cases} \left\lfloor \left(f_j - f_j^{\min}\right) / \varepsilon_j \right\rfloor, & f_j \text{ is to be Minimized;} \\ \left\lceil \left(f_j - f_j^{\min}\right) / \varepsilon_j \right\rceil, & f_j \text{ is to be Maximized.} \end{cases} \tag{5}$$

Where j is the dimension number, $f_j^{\min}$ is the minimum possible value (default as 0) of the j-th dimension, ε_j is the size of the grid on the j-th dimension. The ε-dominated area of an individual is actually the Pareto dominated area of its identification vector; If two are in the same grid (having the same $\boldsymbol{B}$ vector), the one has smaller distance to the $\boldsymbol{B}$ vector is preserved and the other one is deleted. For more detailed description, one can refer to [5].

3.3 ε-MOEA and Its Shortage

The ε-MOEA sets an archive population E(t) and an evolutionary population P(t). In each iteration, an individual by tournament selection from P(t) and another one randomly selected from E(t) are matched. Then they crossover and mutation is processed and finally two new individuals are generated. For each one of them, ε-MOEA uses the general Pareto domination concept to update P(t) while uses the ε-dominance to update E(t). Hence the competitive models are remained in P(t), and the solutions in E(t) are well distributed and the number of them is not too large. Figure 1 shows the results of ZDT1 obtained by ε-MOEA.

As can be seen in the figure, the obtained solution set is nearly well distributed on the PF_{true} (the solid points are scattered in the bolded grids).

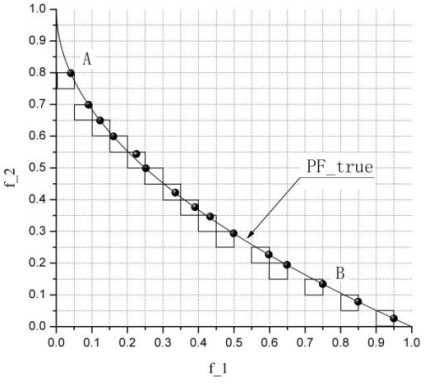


Fig. 1. Results of ZDT1 obtained by ε-MOEA (ε_i=0.05)

However, several grids transited by the true PF contain no solutions, because these grids (for example, the grids above A and the ones besides B in figure 1.) are ε-dominated by the preserved individuals. Obviously, these grid-regions are either extreme or important respective ones, but the algorithm failed to find solutions in these regions. So the diversity of the solution set is not satisfying.

In order to avoid this phenomenon, we proposed a new δ-domination concept and new elitist-reserving strategy. The new strategy based δ-MOEA is also illuminated.

4 New Elitist-Reserving Strategy and δ-MOEA

4.1 The δ-Dominance Concept

As stated above, the ε-dominance uses the identify vector to confirm individual's location, and it allows only one solution preserved in each feasible grid. But, each of the reserved solution has too large dominating ability, which makes some of the extreme and representative solutions lost, and the diversity is dissatisfying. So we improved the dominance concept and allow some individual's (satisfying certain conditions) to be preserved as well as those reserved according to the ε-dominance. It is a more particular concept than the ε-dominance. We call the new dominance concept δ-dominance.

Definition 7 (*δ-dominance*). *Let f,* $g \in \mathbb{R}^{+m}$. *Then f is said to δ-dominate g for some δ>0, denoted as* $f \succ_{\delta} g, \exists_{\Delta_i}$, *with* $0 \le \Delta_i \le \delta_i$, *if and only if for all* $i \in \{1,\ldots,m\}$ *(minimizing):*

$$(f_i - \Delta_i) \le g_i \tag{6}$$

Where δ_i has similar effect as the ε did in the ε-dominance, it sets the upper extent of possible dominating region; while Δ_i helps to confirm the exact δ-dominating area of the preponderant individual. Specially, when $\Delta_i = 0$ *for each i*, the new concept degenerates to the Pareto domination concept, while when $\Delta_i = \delta_i$ *for each i*, it actually equals the ε-dominance. Figure 2 helps to comprehend this relationship.

4.2 The δ-Dominance Based Elitist-Reserving Strategy

For its inclusion in the archive, an individual is compared with each member in the archive for δ-dominance. Every individual in the archive is assigned an identification vector ($\boldsymbol{B}=(B_1, B_2,\ldots B_m)^{\mathbf{T}}$ too, similar as Formulation (5) stated (with the denominator replaced by δ_i). Figure 2 illustrates that the individual P δ-dominates the entire shadowed region, which is large than that of the Pareto dominance definition but not so large than that of ε-dominance. We only discuss the minimization cases alone for brevity, while similar analysis can be followed for maximization or mixed cases as well.

For the individual P, its identification vector is the coordinates of the point B_p in the objective space and its δ-dominating region can be distinctly partitioned into 2 parts: the partition in the grid and the other out of the grid. If two individuals are in the same grid (having the same identification vector), we check which one is closer to the identification vector (in terms of the Euclidean distance), then delete the farer one (for example, individual 2 is to be deleted and 1 is to be preserved). So the in-grid partition is a rectangle with its left-bottom sector removed. And if the comparing individuals have different B vectors (for example, P and Q1 or and Q2), we check whether the

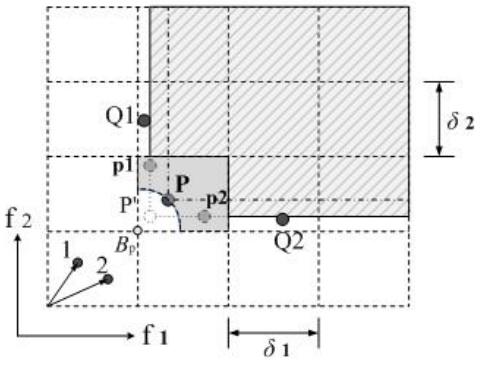


Fig. 2. The δ-dominance concept is illustrated (for minimizing f_1 and f_2)

one with larger B vector is Pareto-dominated by the *suppositional optimum point* of the other's, if so, delete the dominated one, otherwise, both the two are saved in the archive.

The *suppositional optimum point* for P (denoted as P′) is set and updated as:

$$f_i(\mathrm{P'}_{t+1}) = \min\{ f_i(\mathrm{P'}_t), f_i(\mathrm{P}) \} \tag{7}$$

Where *t* denotes the *t*-th updating iteration, *i* is the *i*-th dimension. It should be noticed that each dimension of P′ is actually the *ever-lowest* objective function value (of some individual that has been appeared in the grid). While the *ever-lowest* value for different dimensions can hardly belong to a single individual. Hence, the compositive P′ probably doesn't correspond to any real individual point in the decision space, so we call it ***suppositional*** *optimum point*. The *suppositional optimum point* is related to the real individual and the grid where it stayed, so it should be updated once there is a new individual that entered the archive or replaced an old one in it.

The use of the *suppositional optimum point* insures that the individuals, who should be deleted according to their relationship with the having been deleted former ones, won't be involved in the archive. So the archive set evolves without degradation.

The following procedure explains the elitist-reserving strategy in detail.

Procedure: /* Whether_individual A_enters_the_Archive E(t) */
```
Begin
  If: E(t) is empty
  then A enters E(t), F(A')=F(A);
          // A' is A's suppositional optimum point, F(A) is A's function value
  Else:
  For each P in E(t)
  {  check whether A and P are in the same grid;
     If (TURE) then whether |BpA|-|BpP|<0 ?
             // compare A and P: which one is closer to the B vector
        Yes: A enters E(t),update A'(P'),discard P,
             end procedure;
```

```
        No: discard A,  end procedure;
    Else: whether A is dominated by P'(δ-dominated by P)
                or |AP|<0.5 |δ|;  // or they are too close
       Yes: discard A,  end procedure;
        No: continue;
  }  end For
  A enters E(t), F(A')=F(A);
End.
```

4.3 The Framework of δ-MOEA

The δ-MOEA also sets an evolution population and an archive set that was initialized empty. The crossover operator is SBX and the mutation operator is polynomial mutation. Importantly, we use the new strategy stated above in 4.2 to update the archive set. Figure 3 shows the framework of δ-MOEA.

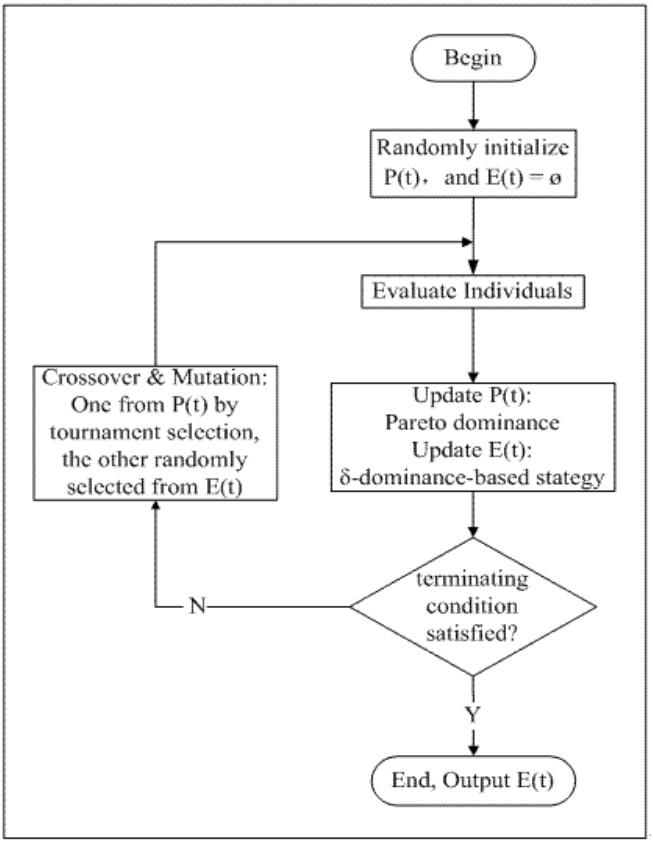


Fig. 3. The flow chart of δ-MOEA

5 Numerical Experiments and Discussion

5.1 Test Functions and Parameter Settings

In order to test the performance of δ-MOEA, we choose some representative and generally used benchmarks. They are: SCH[10], POL[11], FON[12], ZDT2[13][15], DTLZ1[14], DTLZ2[14].The number of decision variables N: N_{SCH} =1, N_{POL} =N_{FON} =2, N_{ZDT2} =30, N_{DTLZ1} =N_{DTLZ2} =7. SCH, POL, FON, ZDT2 have 2 objectives and DTLZ1, DTLZ2 have 3 ones. About the characteristic of these benchmarks, corresponding literatures can be referenced to.

We use the well know NSGA-II[7] and ε-MOEA as comparing algorithms. The parameter settings for δ-MOEA and them are as follows: function evaluations and population size are stated in Table 1. Other parameters are set as that suggested in [7] and [5]. Withη_c=15 for SBX,η_m=20 for polynomial mutation. The size of final solution set is set to equal the population size.

Table 1. Parameter Setting

Objections	2	3
Population size	100	200
Evaluation	20000 (200 gen)	80000 (400 gen)

5.2 Performance Metrics

The metrics to evaluate the performance of the MOEAs were the Spacing Metric(SP) by Schott[8] and the GD by Veldhuizen[9] respectively. The former was to measure the extent of spread achieved among the obtained solutions, and the latter measured the extent of convergence of known set of Pareto-optimal set. For detail, literature [8] and [9] are suggested to be referred to.

5.3 Results and Discussion

For comparison, the SP and GD of the obtained solutions on all or some of the benchmarks by NSGA-II, ε-MOEA and δ-MOEA are shown in Tables 2 and 3.

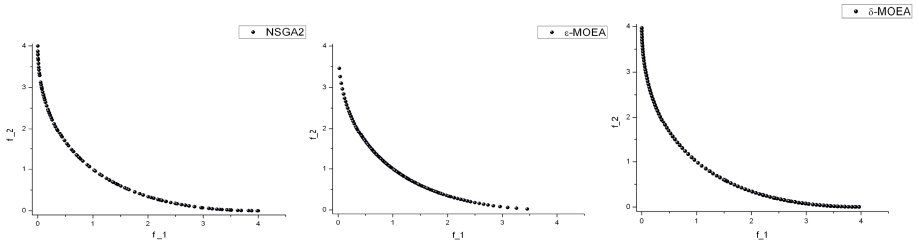
Table 2. Spacing Metric(SP) in 10 runs for NSGA2, ε-MOEA and δ-MOEA

MOEAs	NSGA 2		ε-MOEA		δ-MOEA	
	Sparsity (Avg)	**Std Dev**	**Sparsity (Avg)**	**Std Dev**	**Sparsity (Avg)**	**Std Dev**
SCH	0.03638377	0.00356140	0.04038556	0.00008814	**0.03182887**	0.00012812
POL	0.10597446	0.01097994	0.13732486	0.00950219	**0.03747305**	0.00388055
FON	0.00870666	0.00066549	0.02332467	0.00089394	**0.00434514**	0.00092039
ZDT2	0.00679208	0.00068899	0.00858766	0.00050171	**0.00618374**	0.00052975
DTLZ1	0.04069481	0.00209326	0.01286952	0.00160352	**0.01240354**	0.00061743
DTLZ2	0.09117502	0.02048226	0.03132431	0.00142884	**0.02504852**	0.00106497

Table 3. Convergence Metric(GD) in 10 runs for NSGA2, ε-MOEA and δ-MOEA

MOEAs	NSGA 2		ε-MOEA		δ-MOEA	
	GD (Avg)	**Std Dev**	**GD (Avg)**	**Std Dev**	**GD (Avg)**	**Std Dev**
SCH	0.00041685	0.00002002	**0.00030888**	0.00000282	0.00039228	0.00000651
ZDT2	**0.00014309**	0.00002035	0.00025649	0.00003717	0.00029782	0.00004173
DTLZ1	**0.00004080**	0.00001494	0.00007769	0.00001098	0.00008530	0.00018663
DTLZ2	**0.00028334**	0.00004796	0.00396315	0.00077572	0.00309575	0.00008352

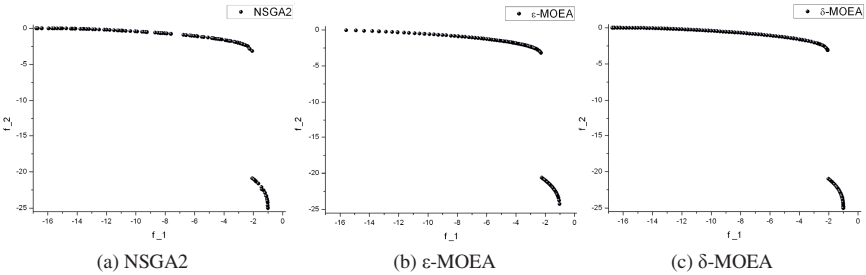
From Table 2, we know that for all problems, δ-MOEA has the best performance in maintaining diversity of the solutions. ε-MOEA is worse than NSGA-II when the objectives are 2 but better than it when the objectives are 3. Table 3 told that NSGA-II gain the best GD of the three on ZDT2, DTLZ1 and DTLZ2, while on SCH, ε-MOEA won the others; Though δ-MOEA is a little weaker than the other two, it is also competitive with them. In the following, we'll give a more intuitionistic illumination by a set of figures (Figure 4-9).



(a) NSGA2 (b) ε-MOEA (c) δ-MOEA

Fig. 4. Distribution of obtained solutions for NSGA2, ε-MOEA and δ-MOEA on SCH

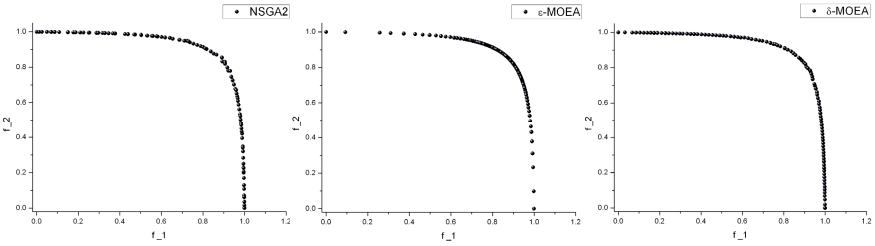
Figure 4 shows that the distribution of obtained solutions by NSGA-II seems discrete and that by ε-MOEA becomes sparser as going to the extremal regions of PF_{true}. Obviously δ-MOEA gained continuous and uniform solutions distributed on the PF_{true}.



(a) NSGA2 (b) ε-MOEA (c) δ-MOEA

Fig. 5. Distribution of obtained solutions for NSGA2, ε-MOEA and δ-MOEA on POL

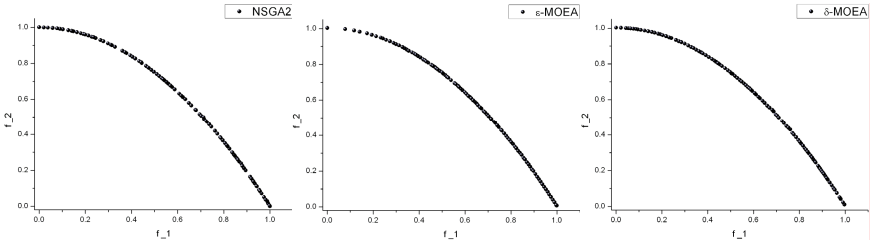
In Figure 5, we can see that on POL, NSGA-II got PF that is not sleek; PF obtained by ε-MOEA is dense in the parted segment but too sparse in the area of PF_{true} closing to f_2. While δ-MOEA gains a well distributed, uniform solution set.



(a) NSGA2 (b) ε-MOEA (c) δ-MOEA

Fig. 6. Distribution of obtained solutions for NSGA2, ε-MOEA and δ-MOEA on FON

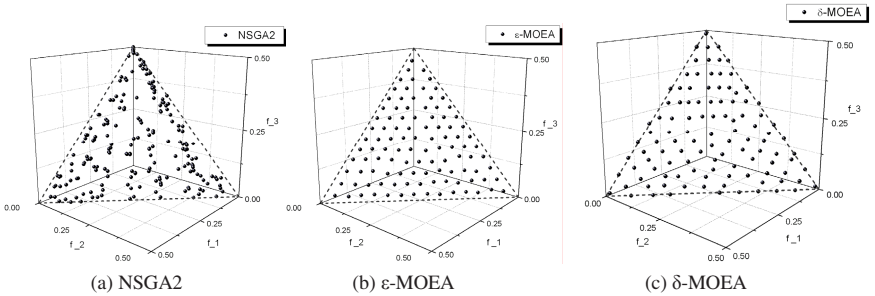
Figure 6 illuminates the shortage of ε-MOEA clearly. It gets too many solutions crowded in the middle part but quite fewer ones in the extremal part. And NSGA-II has dissatisfying results while δ-MOEA is still the best.



(a) NSGA2 (b) ε-MOEA (c) δ-MOEA

Fig. 7. Distribution of obtained solutions for NSGA2, ε-MOEA and δ-MOEA on ZDT2

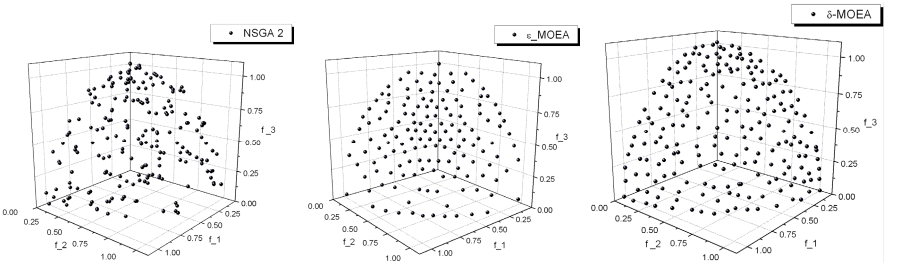
Figure 7 shows the similar situation of results. The PF obtained by NSGA-II is not so good and that by ε-MOEA is obviously sparser when the value of f2 approaching 1. The distribution of solutions obtained by δ-MOEA is acceptable.



(a) NSGA2 (b) ε-MOEA (c) δ-MOEA

Fig. 8. Distribution of obtained solutions for NSGA2, ε-MOEA and δ-MOEA on DTLZ1

For the benchmarks with 3 objectives, DTLZ1 and DTLZ2 are selected. In Figure 8, (a) shows that NSGA-II found solutions asymmetrically distributed; (b) shows the set by ε-MOEA is symmetrical but with no solutions reserved in extremal area;



(a) NSGA2 (b) ε-MOEA (c) δ-MOEA

Fig. 9. Distribution of obtained solutions for NSGA2, ε-MOEA and δ-MOEA on DTLZ2

(c) tells the case that the set by δ-MOEA is satisfying uniformly distributed as well as the extreme solutions were found.

In Figure 9, (a) shows the disorder solution set obtained by NSGA-II; (b) indicates that ε-MOEA still fail to find symmetrical solutions in extremal area; (c) implies that δ-MOEA gained the trade-off between extensiveness and uniformity.

In conclusion, δ-MOEA gained solution sets with better diversity than the other two did. It is because that the new elitist-reserving strategy preserved individuals in the ε-dominated grids especially in those near the extremal region of the PF. Since the individuals that may be eliminated by ε-MOEA are kept down in the grids that are went through by the PF, so the whole solution set is with higher uniformity than that by the ε-MOEA.

6 Conclusions

The proposed δ-dominance concept and new elitist-reserving strategy help δ-MOEA gain a solution set that has good distribution. The δ-MOEA overcomes the difficulties of ε-MOEA in finding and preserving solutions in extremal and some other important regions, it also outperforms the typical NSGA-II as the diversity is concerned. The application of *suppositional optimum point* is also novel and the result of numerical experiments illustrates the good performance of δ-MOEA.

Acknowledgment. This work was supported by The National Natural Science Foundation of China (60773047), and the Natural Science Foundation of Hunan Province (05JJ30125), and the Keystone Science Research Project of the Education Office of Hunan Province (06A074).

References

1. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Chichester (2001)
2. Xie, T., Chen, H.W., Kang, L.S.: Multi-objective Optimal Evolutionary Algorithms. Journal of Computer 26(8), 997–1003 (2003)
3. Zheng, J.H.: Multiobjective Evolutionary Algorithms and Applications. Science Press, Beijing (2007)

4. Laumanns, M., Thiele, L., Deb, K., Zitzler, E.: Combining convergence and diversity in evolutionary multi-objective optimization. Evolutionary Computation 10(3), 263–282 (2002)
5. Deb, K., Mohan, M., Mishra, S.: A Fast Multi-objective Evolutionary Algorithm for Finding Well-Spread Pareto-Optimal Solutions. KanGAL Report. No 2003002
6. Coello Coello, C.A.: Guest Editorial: Special issue on evolutionary multi-objective optimization. IEEE Transactions on Evolutionary Computation 7(2), 97–99 (2003)
7. Kalyanmoy, D., Pratap, A., Agrawal, S., Meyrivan, T.: A Fast and Elitist Multi-objective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
8. Schott, J.R.: Fault tolerant design using single and multicriteria genetic algorithm optimization. Master's theis of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge (1995-05)
9. Van Veldhuizen, D.A., Lamont, G.B.: On measuring multiobjective evolutionary algorithm performance. In: 2000 Congress on Evolutionary Computation, vol. 1, pp. 204–211 (2000)
10. David, S.J.: Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In: Grefenstette: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, pp. 93–100 (1985)
11. Carlo, P.: Multi-objective Optimization by GAs: Application to System and Component Design. Methods in Applied Sciences 1996: Invited Lectures and Special Technological Sessions of the Third ECCOMAS Computational Fluid Dynamics Conference and the Second ECCOMAS Conference on Numerical Methods in Engineering, pp. 258–264. Wiley, Chichester (1996)
12. Fonseca, C.M., Fleming, P.J.: An Overview of Evolutionary Algorithms in Multi-objective Optimization. Evolutionary Computation 3(1), 1–16 (1995)
13. Kalyanmoy, D.: Multi-Objective Genetic Algorithms: Problem difficulties and Construction of Test Problems. Technical Report CI-49/98. Department of Computer Science/LS11,University of Dortmund
14. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable multi-objective optimization test problems. In: Proceeding of the Congress on Evolutionary Computation (CEC 2002), pp. 825–830 (2002)
15. Kalyanmoy, D.: Multi-Objective Genetic Algorithms: Problem difficulties and Construction of Test Problems. Evolutionary Computation 7(3), 205–230 (1999)

The Parameterized Complexity of the Rectangle Stabbing Problem and Its Variants*

Michael Dom[1] and Somnath Sikdar[2]

[1] Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
dom@minet.uni-jena.de
[2] The Institute of Mathematical Sciences,
C.I.T Campus, Taramani, Chennai 600113, India
somnath@imsc.res.in

Abstract. We study an NP-complete geometric covering problem called d-Dimensional Rectangle Stabbing, where, given a set of axis-parallel d-dimensional hyperrectangles, a set of axis-parallel $(d-1)$-dimensional hyperplanes and a positive integer k, the question is whether one can select at most k of the hyperplanes such that every hyperrectangle is intersected by at least one of these hyperplanes. This problem is well-studied from the approximation point of view, while its parameterized complexity remained unexplored so far. Here we show, by giving a nontrivial reduction from a problem called Multicolored Clique, that for $d \geq 3$ the problem is W[1]-hard with respect to the parameter k. For the case $d = 2$, whose parameterized complexity is still open, we consider several natural restrictions and show them to be fixed-parameter tractable.

1 Introduction

A geometric covering problem, in the broadest sense, consists of a set of geometric objects and a set of "resources"; the goal is to find a small set of resources that "covers" all objects. Geometric covering problems arise in many practical applications and are subject of intensive research (see [6,7,11]). In this paper we consider a geometric covering problem known as d-Dimensional Rectangle Stabbing. Here, the input consists of a set R of axis-parallel d-dimensional hyperrectangles, a set L of axis-parallel $(d-1)$-dimensional hyperplanes, and a positive integer k; the question is whether there is a set $L' \subseteq L$ with $|L'| \leq k$ such that every hyperrectangle from R is intersected by at least one hyperplane from L'. In the special case of $d = 2$, the set R consists of axis-parallel rectangles in the plane, and L consists of vertical and horizontal lines. In the polynomial-time approximation setting, the optimization version of d-Dimensional Rectangle Stabbing is considered, which asks for a *minimum-cardinality* set $L' \subseteq L$ to cover all rectangles from R.

The literature provides a bunch of results concerning the approximability of d-Dimensional Rectangle Stabbing. Hassin and Megiddo [8] described

* Supported by the DAAD-DST exchange program D/05/57666.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 288–299, 2008.

a factor-$d2^{d-1}$ approximation algorithm for the problem variant where L consists of lines instead of hyperplanes and all hyperrectangles in R are identical. Gaur et al. [6] gave a factor-2 approximation algorithm for the case $d = 2$ and extended this result to the problem d-Dimensional Rectangle Stabbing, for which they provided a factor-d approximation algorithm. Moreover, Mecke et al. [12] gave a factor-d approximation algorithm for a problem called d-C1P-Set Cover, which is a generalization of d-Dimensional Rectangle Stabbing. The restricted version of 2-Dimensional Rectangle Stabbing where for every rectangle the number of horizontal lines intersecting it is bounded from above by one is known as Interval Stabbing. This problem was considered by Kovaleva and Spieksma [9,10], leading to constant-factor approximation algorithms for several variants of the problem. Hassin and Megiddo [8] gave approximation algorithms for the more general variant of Interval Stabbing where for every rectangle the number of horizontal lines or the number of vertical lines intersecting it is bounded from above by one. Weighted and capacitated versions of 2-Dimensional Rectangle Stabbing have been considered by Even et al. [2].

Here, we consider d-Dimensional Rectangle Stabbing from the viewpoint of parameterized complexity. More specifically, we investigate whether d-Dimensional Rectangle Stabbing is fixed-parameter tractable with respect to the parameter "solution size" k, that is, if there exists an algorithm running in $O(f(k) \cdot |R \cup L|^{O(1)})$ time with f depending only on k. On the one hand, we show in Section 3 that for $d \geq 3$ the problem is W[1]-hard with respect to the parameter k, meaning that it is unlikely that there exists such an algorithm. On the other hand, in Section 4 we consider several natural restrictions of the case $d = 2$ and show them to be fixed-parameter tractable. The parameterized complexity for the case $d = 2$ without further restrictions remains open.

Due to lack of space, some proofs are omitted.

2 Preliminaries

A parameterized problem is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet and $\mathbb{N}$ is the set of natural numbers. An instance of a parameterized problem is, therefore, a pair (I, k), where k is called the parameter. In the framework of parameterized complexity [1,5,13], the running time of an algorithm is viewed as a function of two quantities: the size of the problem instance *and* the parameter. A parameterized problem is said to be *fixed parameter tractable (FPT)* with respect to the parameter k if there exists an algorithm for the problem running in $f(k) \cdot |I|^{O(1)}$ time, where f is a computable function only depending on k.

A common tool in the development of fixed-parameter algorithms is to use a set of *data reduction rules* to obtain what is called a *problem kernel*. A *data reduction rule* is a polynomial-time algorithm which takes as input a problem instance (I, k) and outputs an instance (I', k') such that $|I'| \leq |I|$, $k' \leq k$, and (I', k') is a YES-instances iff (I, k) is a YES-instance. An instance to which none of a given set of data reduction rules applies is called *reduced* with respect

to these rules. A reduced instance (I', k') is called a *problem kernel* if its size is bounded from above by a function f depending only on k. If a parameterized problem has a kernel, then it is clearly fixed-parameter tractable.

A parameterized problem π_1 is *fixed-parameter reducible* to a parameterized problem π_2 if there are two computable functions $f, g : \mathbb{N} \to \mathbb{N}$ and an algorithm Φ which transforms an instance (I, k) of π_1 into an instance $(I', f(k))$ of π_2 in $g(k) \cdot |I|^{O(1)}$ time such that $(I', f(k))$ is a YES-instance for π_2 iff (I, k) is a YES-instance for π_1. The basic complexity class for fixed-parameter intractability is $W[1]$, as there is strong evidence that $W[1]$-hard problems are not fixed-parameter tractable [1,5,13]. To show that a problem is $W[1]$-hard, one needs to exhibit a fixed-parameter reduction from a known $W[1]$-hard problem to the problem at hand.

A graph $G = (V, E)$ is called *k-colorable* if there is a function $c : V \to \{1, \ldots, k\}$ satisfying $\forall\{u, v\} \in E : c(u) \neq c(v)$; the function c is then called a *proper vertex k-coloring* for G.

To achieve our hardness result, we consider d-DIMENSIONAL RECTANGLE STABBING as a covering problem on binary matrices, which is a restriction of the following, very general matrix problem:

SET COVER[1]
Given: A binary matrix M and a positive integer k.
Question: Is there a set of at most k columns of M such that the submatrix M' of M that is induced by these columns has at least one 1 in every row?

To define restricted versions of SET COVER, we need the following definitions.

Definition 1. *1. Given a binary matrix M, a* block of 1s *in a row of M is a maximal set of consecutive 1-entries in this row.*
2. A binary matrix M has the d-consecutive ones property (d-C1P) *if every row of M has at most d blocks of 1s.*
3. A binary matrix M with columns $c_1, \ldots, c_n$ has the separated d-consecutive ones property (d-SC1P) *if the columns of M can be partitioned into d sets of consecutive columns $C^1 = \{c_1, \ldots, c_{j_1}\}$, $C^2 = \{c_{j_1+1}, \ldots, c_{j_2}\}$, $\ldots$, $C^d = \{c_{j_{d-1}+1}, \ldots, c_n\}$ such that for every $i \in \{1, \ldots, d\}$ the submatrix of M induced by C^i has at most one block of 1s per row.*

If SET COVER is restricted by demanding that the input matrix M must have the d-C1P, then we call the resulting problem d-C1P-SET COVER; if M must have the d-SC1P, then we call the resulting problem d-SC1P-SET COVER.

Observation 1. *The problems* d-DIMENSIONAL RECTANGLE STABBING *and* d-SC1P-SET COVER *are equivalent: There is a polynomial-time computable one-to-one mapping between instances of* d-DIMENSIONAL RECTANGLE STABBING *and instances of* d-SC1P-SET COVER *that does not change the value of k and that maps* YES*-instances to* YES*-instances and* NO*-instances to* NO*-instances.*

[1] The equivalence of this definition and the more common definition of SET COVER as a subset selection problem can easily be seen by identifying columns with subsets and rows with elements to be covered.

This observation is easy to see—the ith dimension in a d-Dimensional Rectangle Stabbing instance can be represented by the column set C^i in a d-SC1P-Set Cover instance and vice versa.

For some of our FPT algorithms, we make use of the following well-known fact: Given a set of axis-parallel rectangles and a set of vertical (horizontal) lines, the task of finding a minimum-cardinality subset of these vertical (horizontal) lines that intersects all rectangles is polynomial-time solvable[2]: Order the rectangles with respect to their right (bottom) end. Then, repeatedly take the first rectangle r in this order, include the rightmost vertical (bottommost horizontal) line l that intersects r into the solution, and delete all rectangles intersected by l, until all rectangles are deleted. The solution obtained is a minimum-size set of vertical (horizontal) lines that are required to intersect all rectangles. Moreover, all rectangles r together that are selected by the algorithm form a "certificate" in the sense that they cannot be intersected by a set of vertical (horizontal) lines that is smaller than the solution found by the algorithm. The pseudocode of this algorithm is displayed in Fig. 2.

3 W[1]-Hardness Proof for d-Dimensional Rectangle Stabbing with $d \geq 3$

In this section we prove that d-Dimensional Rectangle Stabbing with parameter k is W[1]-hard for every $d \geq 3$. To this end, we exhibit a fixed-parameter reduction from Multicolored Clique, which is defined as follows, to 3-SC1P-Set Cover.

Multicolored Clique
Given: An undirected k-colorable graph $G = (V, E)$, a positive integer k, and a proper vertex k-coloring $c : V \to \{1, \ldots, k\}$ for G.
Question: Is there a size-k clique in G?

Multicolored Clique is W[1]-complete with respect to the parameter k [4]. Using the "k-Multicolored Clique reduction technique" designed by Fellows et al. [4], a fixed-parameter reduction from Multicolored Clique to 3-C1P-Set Cover can be found in a rather straightforward way [3], which proves the W[1]-hardness of the latter problem. However, the W[1]-hardness of 3-SC1P-Set Cover is more difficult to prove because of the more restricted nature of this problem.

The basic scheme of the reduction. The basic scheme of our reduction follows the technique described by Fellows et al. [4]. The key idea is to use an alternative, equivalent formulation of Multicolored Clique: Given an undirected k-colorable graph $G = (V, E)$, a positive integer k, and a proper vertex k-coloring $c : V \to \{1, \ldots, k\}$ for G, find a set $E' \subseteq E$ with $|E'| = \binom{k}{2}$ and a set $V' \subseteq V$ with $|V'| = k$ that satisfy the following constraints:

[2] This problem is equivalent to Clique Cover on interval graphs.

1. For every unordered pair $\{a, b\}$ of colors from $\{1, \ldots, k\}$, the edge set E' contains an edge whose endpoints are colored with a and b.
2. For every color from $\{1, \ldots, k\}$, the vertex set V' contains a vertex of this color.
3. If E' contains an edge $\{u, v\}$, then V' contains the vertices u and v.

Given an instance (G, k, c) of Multicolored Clique, we construct an equivalent instance (M, k') of 3-SC1P-Set Cover based on this alternative formulation. To this end, define the *color of an edge* $\{u, v\}$, denoted with $d(\{u, v\})$, as $d(\{u, v\}) = \{c(u), c(v)\}$. We assume that the edges $E = \{e_1, \ldots, e_{|E|}\}$ and vertices $V = \{v_1, \ldots, v_{|V|}\}$ of G are ordered in such a way that that edges and vertices of the same color appear consecutively: For every pair $p_1, p_2 \in \{1, \ldots, |E|\}$ with $p_1 < p_2$ and $d(e_{p_1}) = d(e_{p_2})$ it holds that $\forall p_3 \in \{p_1 + 1, \ldots, p_2 - 1\} : d(e_{p_3}) = d(e_{p_1}) = d(e_{p_2})$, and for every pair $q_1, q_2 \in \{1, \ldots, |V|\}$ with $q_1 < q_2$ and $c(v_{q_1}) = c(v_{q_2})$ it holds that $\forall q_3 \in \{q_1 + 1, \ldots, q_2 - 1\} : c(v_{q_3}) = c(v_{q_1}) = c(v_{q_2})$.

The idea of the reduction is that every column of M corresponds to an edge or a vertex of the given graph G; the rows of M are constructed in such a way that any column subset of M that is a solution for 3-SC1P-Set Cover on (M, k') corresponds to a solution (E', V') for Multicolored Clique on (G, k, c). To this end, the rows of M must enforce that the three constraints for Multicolored Clique mentioned above are satisfied. In order to obtain a matrix that has the 3-SC1P, we need not only one, but *two* columns in M for every edge e in G. Hence an instance (G, k, c) of Multicolored Clique is mapped to an instance (M, k'), where $k' = 2 \cdot \binom{k}{2} + k$. We next describe the construction of M.

The columns of M. The matrix M has $2 \cdot |E| + |V|$ columns, partitioned into three sets $C^1 = \{c^1_1, \ldots, c^1_{|E|}\}$, $C^2 = \{c^2_1, \ldots, c^2_{|E|}\}$, and $C^3 = \{c^3_1, \ldots, c^3_{|V|}\}$ and ordered as follows: $c^1_1, \ldots, c^1_{|E|}, c^2_1, \ldots, c^2_{|E|}, c^3_1, \ldots, c^3_{|V|}$.

The rows of M. The rows of M have to ensure that every solution for 3-SC1P-Set Cover on $(M, k' = 2 \cdot \binom{k}{2} + k)$ corresponds to a subset of edges and vertices of G satisfying the three constraints mentioned above. Because there are two columns in M for every edge in G, we need *four* types of rows: Rows of Type 1 and 2 ensure that any set of k' columns that forms a solution for 3-SC1P-Set Cover contains exactly $\binom{k}{2}$ columns from C^1—one of each edge color—, $\binom{k}{2}$ columns from C^2—one of each edge color—, and k columns from C^3—one of each vertex color. Type 3 rows ensure that the columns chosen from C^1 and C^2 are consistent: if a solution contains the column c^1_j then it must contain c^2_j and vice versa. Finally, Type 4 rows ensure that if a solution contains a column c^1_j corresponding to an edge $\{u, v\}$ then it also contains the columns corresponding to the vertices u and v. See Fig. 1 for an illustration of the following construction details.

Type 1 rows. For every edge color $\{a, b\}$, add two rows $r^1_{\{a,b\},C^1}$ and $r^1_{\{a,b\},C^2}$ to M. For $i = 1, 2$, the row $r^1_{\{a,b\},C^i}$ has a 1 in every column $c^i_j \in C^i$ with $d(e_j) = \{a, b\}$, and 0s in all other columns.

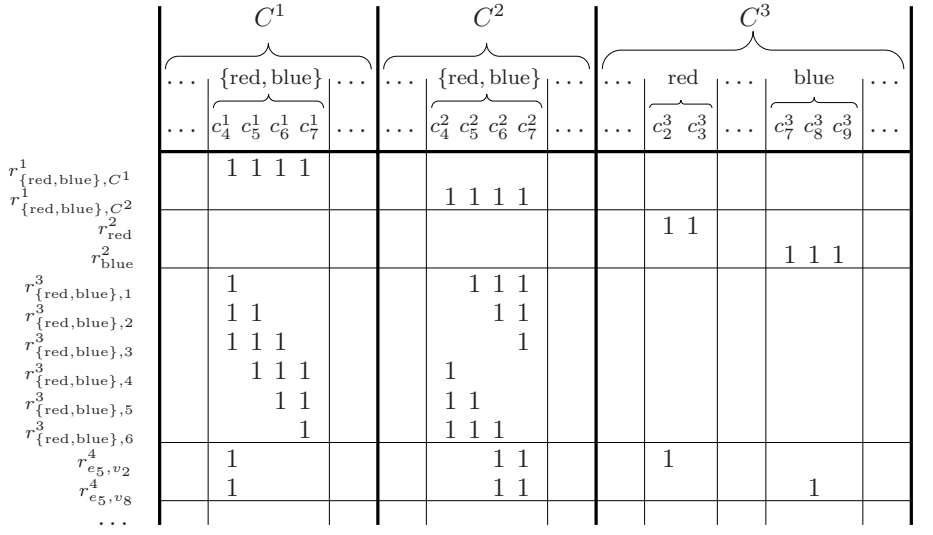


Fig. 1. Example for the construction of M. We assume that in G there are exactly two red vertices v_2, v_3 and exactly three blue vertices v_7, v_8, v_9, among vertices of other colors. Moreover, the only edges between red and blue vertices are e_4, e_5, e_6, e_7 with $e_5 = \{v_2, v_8\}$. Hence, $\text{first}(\{\text{red}, \text{blue}\}) = 4$.

Type 2 rows. For every vertex color $a \in \{1, \ldots, k\}$, add a row r^2_a to M which has a 1 in every column $c^3_j \in C^3$ with $c(v_j) = a$, and 0s in all other columns.

Type 3 rows. For every edge color $\{a, b\}$, define $E_{\{a,b\}} := \{e \in E \mid d(e) = \{a, b\}\}$ and $\text{first}(\{a, b\}) := \min\{p \in \{1, \ldots, |E|\} \mid d(e_p) = \{a, b\}\}$. Now, for every edge color $\{a, b\}$, add a set of $2 \cdot (|E_{\{a,b\}}| - 1)$ rows $r^3_{\{a,b\},i}$ where $1 \leq i \leq 2 \cdot (|E_{\{a,b\}}| - 1)$. A row $r^3_{\{a,b\},i}$ with $i \in \{1, \ldots, |E_{\{a,b\}}| - 1\}$, has a 1 in

- every column $c^1_j \in C^1$ with $d(e_j) = \{a, b\}$ and $j < \text{first}(\{a, b\}) + i$ and
- every column $c^2_j \in C^2$ with $d(e_j) = \{a, b\}$ and $j \geq \text{first}(\{a, b\}) + i$,

and 0s in all other columns. A row $r^3_{\{a,b\},i}$ with $i \in \{|E_{\{a,b\}}|, \ldots, 2 \cdot (|E_{\{a,b\}}| - 1)\}$ has a 1 in

- every column $c^1_j \in C^1$ with $d(e_j) = \{a, b\}$ and $j \geq \text{first}(\{a, b\}) + i - (|E_{\{a,b\}}| - 1)$ and
- every column $c^2_j \in C^2$ with $d(e_j) = \{a, b\}$ and $j < \text{first}(\{a, b\}) + i - (|E_{\{a,b\}}| - 1)$,

and 0s in all other columns.

Type 4 rows. For every edge $e_p = \{v_{q_1}, v_{q_2}\} \in E$, add two rows $r^4_{e_p,v_{q_1}}$ and $r^4_{e_p,v_{q_2}}$ to M. For $i = 1, 2$, the row $r^4_{e_p,v_{q_i}}$ has a 1 in

- every column $c^1_j \in C^1$ with $d(e_j) = d(e_p)$ and $j < p$,
- every column $c^2_j \in C^2$ with $d(e_j) = d(e_p)$ and $j > p$, and
- the column $c^3_{q_i} \in C^3$,

and 0s in all other columns.

Lemma 1. *Let* (G, k, c) *be an instance of* Multicolored Clique *and let* (M, k') *be the instance of* 3-SC1P-Set Cover *obtained by the above construction. Then* G *contains a clique of size* k *if and only if there exists a set of* $k' = 2 \cdot \binom{k}{2} + k$ *columns in* M *that hits a* 1 *in every row.*

Theorem 1. *For every* $d \geq 3$, d-Dimensional Rectangle Stabbing *is* W[1]*-hard with respect to the parameter* k.

By adding some additional columns to the above construction, we get the following result.

Theorem 2. *For every* $d \geq 3$, *the restricted variant of* d-Dimensional Rectangle Stabbing *where every hyperrectangle in* R *is a hypercube is* W[1]*-hard with respect to the parameter* k.

With a similar reduction we can also show the W[1]-hardness of the following problem: Given a set R of axis-parallel d-dimensional hyperrectangles, a set L of axis-parallel lines, and a positive integer k, is there a set $L' \subseteq L$ with $|L'| \leq k$ such that every hyperrectangle from R is intersected by at least one line from L'?

Theorem 3. *For* $d \geq 3$, *the problem of stabbing axis-parallel* d*-dimensional hyperrectangles with* k *axis-parallel lines is* W[1]*-hard with respect to the parameter* k.

4 FPT Algorithms for Restrictions of 2-Dimensional Rectangle Stabbing

In the previous section we have shown that d-Dimensional Rectangle Stabbing with parameter k is W[1]-hard for $d \geq 3$. However, the parameterized complexity of 2-Dimensional Rectangle Stabbing, where a set R of axis-parallel rectangles has to be stabbed with at most k lines chosen from a given set L of vertical and horizontal lines, is still open. In this section we consider some natural restrictions of this problem and show them to be fixed-parameter tractable.

For an instance (R, L, k) of 2-Dimensional Rectangle Stabbing, let $L = V \cup H$, where $V = \{v_1, \ldots, v_n\}$ are the vertical lines ordered from left to right and $H = \{h_1, \ldots, h_m\}$ are the horizontal lines ordered from top to bottom. For a rectangle $r \in R$, let $\mathrm{lx}(r), \mathrm{rx}(r), \mathrm{tx}(r), \mathrm{bx}(r)$ be the index of the leftmost, rightmost, topmost and bottommost line intersecting r. Define the width $\mathrm{wh}(r) := \mathrm{rx}(r) - \mathrm{lx}(r) + 1$ and the height $\mathrm{ht}(r) := \mathrm{bx}(r) - \mathrm{tx}(r) + 1$ as the number of vertical and horizontal lines, respectively, intersecting r.

We start with some well-known data reduction rules for 2-Dimensional Rectangle Stabbing, whose correctness is obvious.

1. If there is a rectangle that is intersected by no line from L, then the given instance is a no-instance.

2. If there is a rectangle that is intersected by exactly one line $l \in L$, then delete all rectangles that are intersected by l, delete l, and decrease k by one.
3. If there are two lines $l_1, l_2 \in L$ such that every rectangle in R that is intersected by l_2 is also intersected by l_1, then delete l_2.
4. If there are two rectangles $r_1, r_2 \in R$ such that every line in L that intersects r_1 also intersects r_2, then delete r_2.

The following observation is an immediate consequence of data reduction rule 3.

Observation 2. *In a reduced problem instance, for every vertical line $v_j \in V$ there exist rectangles $r, r' \in R$ with $\mathrm{lx}(r) = j$ and $\mathrm{rx}(r') = j$. For every horizontal line $h_i \in H$ there exist rectangles $r, r' \in R$ with $\mathrm{tx}(r) = i$ and $\mathrm{bx}(r') = i$.*

In particular, Observation 2 implies that in a reduced problem instance there exist rectangles $r, r' \in R$ such that $\mathrm{wh}(r) = 1$ and $\mathrm{ht}(r') = 1$.

4.1 Case 1: Rectangles Have Bounded Height

We first consider the case where the height of every rectangle in R is bounded by a number b. Even the case $b = 1$ where every rectangle is a horizontal segment is NP-complete; Hassin and Megiddo [8] and Kovaleva and Spieksma [9,10] gave approximation algorithms for this case and some of its variants.

For our FPT considerations, we use a simple search-tree algorithm using Observation 2. At every step, apply the data reduction rules until the current instance is reduced, search for a rectangle r with $\mathrm{rx}(r) = 1$, and branch as follows: either select the single vertical line that intersects r or select one of the at most b horizontal lines that intersect r.

Theorem 4. *The restricted variant of* 2-Dimensional Rectangle Stabbing *where the height $\mathrm{ht}(r)$ of every rectangle $r \in R$ is bounded from above by a number b can be solved in $O((b+1)^k \cdot n^{O(1)})$ time.*

The algorithm described above can be modified to solve also the weighted version of the problem, where every line has a weight that is bounded from below by 1 and from above by a number b'. The data reduction rules need modification for this problem version; the running time of the algorithm is $O((b+b')^k \cdot n^{O(1)})$.

4.2 Case 2: Rectangles Have Bounded Width or Height

Next, we consider a generalization of Case 1: Here, for every rectangle r in R the width $\mathrm{wh}(r)$ *or* the height $\mathrm{ht}(r)$ is bounded from above by a number b. Clearly, even the case $b = 1$, where every rectangle is either a horizontal or a vertical segment, is NP-complete; this case was already considered by Hassin and Megiddo [8] from the approximation point of view.

The approach outlined in Section 4.1 does not work anymore since in a reduced instance the height of every rectangle r with $\mathrm{rx}(r) = 1$ may be unbounded. However, there is again a search-tree algorithm. Let $R_h \subseteq R$ be the set of

```
1  function greedy(R, L, k) {
       // Input:   R: a set of rectangles,
       //          L: a set of lines that are either all vertical or all horizontal,
       //          k: a nonnegative integer.
       // Output: Either L' ⊆ L or R^0 ⊆ R.
       //          If all rectangles from R can be stabbed with a set L' of at most k lines
       //          from L, then such a set L' is returned.
       //          Otherwise, a set R^0 of k + 1 rectangles from R is returned that cannot be
       //          stabbed with at most k lines from L.
2      R' := R;  R^0 := ∅;  L' := ∅;
3      while R' ≠ ∅: {
4          if L contains only vertical lines: { r := a rectangle from R' with minimum rx(r);  l := v_rx(r); }
5          else { r := a rectangle from R' with minimum bx(r);  l := h_bx(r); }
6          R^0 := R^0 ∪ {r};  L' := L' ∪ {l};
7          delete all rectangles from R' that are intersected by l;
8          if |R^0| = k + 1: return R^0;  }
9      return L';  }
```

Fig. 2. Greedy algorithm for stabbing a set R of rectangles with at most k lines chosen from a given set L of vertical lines or horizontal lines

rectangles with bounded height and let $R_v \subseteq R$ be the set of rectangles with bounded width. Now, we write k as a sum $k_h + k_v$ in all possible ways, where k_h and k_v denote the number of horizontal and vertical lines, respectively, allowed to be chosen into the solution. For every splitting of k into k_h and k_v, we run a branching algorithm, which performs in every step the following actions.

First, compute the minimum number of vertical lines required to intersect the rectangles in R_h. This is polynomial-time doable, and the simple greedy algorithm in Fig. 2 obtains such a set of vertical lines. If R_h cannot be stabbed with a set of at most k_v vertical lines, then the algorithm in Fig. 2 outputs a set $R_h^0 \subseteq R_h$ of size $k_v + 1$ such that the optimum number of vertical lines needed to intersect all rectangles in R_h^0 is exactly $k_v + 1$. Any solution for 2-Dimensional Rectangle Stabbing on (R, L, k) consisting of at most k_v vertical and at most k_h horizontal lines must intersect at least one rectangle in R_h^0 by a horizontal line. Hence, branch on the $(k_v + 1) \cdot b$ possibilities to do so.

If, however, all rectangles in R_h can be intersected with k_v vertical lines, we use the greedy algorithm to check whether the rectangles in R_v can be intersected with k_h horizontal lines. If not, we branch on $(k_h + 1) \cdot b$ possibilities in analogy to the branching for R_h^0 described above; otherwise, we return the union of the solutions returned by the two calls to the greedy algorithm. Fig. 3 shows a pseudocode for this algorithm. The branching number is at most bk, which leads to the following theorem.

Theorem 5. *The restricted variant of* 2-Dimensional Rectangle Stabbing *where the width or the height of every rectangle in R is bounded from above by a number b can be solved in $O((bk)^k \cdot n^{O(1)})$ time.*

4.3 Case 3: Bounded Intersection

In this subsection we consider a restriction of 2-Dimensional Rectangle Stabbing in which every horizontal line intersects at most b rectangles from R;

```
1  function stab(R_h, R_v, H, V, k_h, k_v) {
       // Input:   R_h: a set of rectangles with bounded height,
       //          R_v: a set of rectangles with bounded width,
       //          H, V: a set of horizontal lines and a set of vertical lines,
       //          k_h, k_v: nonnegative integers.
       // Output: A subset of H ∪ V containing ≤ k_h lines from H and ≤ k_v lines from V
       //          that stabs all rectangles from R_h ∪ R_v, or null, if no such subset exists.
2      if greedy(R_h, V, k_v) returns a set R_h^0 ⊆ R_h of rectangles: {
3          if k_h = 0: return null;
4          for every rectangle r ∈ R_h^0:  for every line h ∈ H that intersects r: {
5              R'_h := R_h \ R_h(h);  R'_v := R_v \ R_v(h);  H' := H \ {h};
6              A := stab(R'_h, R'_v, H', V, k_h − 1, k_v);
7              if A ≠ null: return A ∪ {h};  }
8          return null;  }
9      if greedy(R_v, H, k_h) returns a set R_v^0 ⊆ R_v of rectangles: {
10         if k_v = 0: return null;
11         for every rectangle r ∈ R_v^0:  for every line v ∈ V that intersects r: {
12             R'_h := R_h \ R_h(v);  R'_v := R_v \ R_v(v);  V' := V \ {v};
13             A := stab(R'_h, R'_v, H, V', k_h, k_v − 1);
14             if A ≠ null: return A ∪ {v};  }
15         return null;  }
16     return the union V' ∪ H' of the solutions returned by the two calls (lines 2 and 9) of greedy();  }
```

Fig. 3. Branching algorithm for stabbing a set $R_v \cup R_h$ of rectangles with at most k_v lines chosen from a given set V of vertical lines and at most k_h lines chosen from a given set H of horizontal lines. For a line l, we denote with $R_h(l)$ and $R_v(l)$ the set of all rectangles in R_h and R_v, respectively, that are intersected by l.

this restriction was already considered by Kovaleva and Spieksma [9,10] from the approximation point of view. For $b = 1$, this problem is clearly polynomial-time solvable since the horizontal lines can just be ignored. For $b = 2$ the problem is NP-complete, but one can easily find an $O(k^k \cdot n^{O(1)})$-time branching algorithm. However, this algorithm cannot be generalized for the case $b \geq 3$. In this subsection, we show that this restriction of 2-Dimensional Rectangle Stabbing is fixed-parameter tractable with respect to the combined parameters k and b by developing a problem kernel.

First, in addition to the previously mentioned data reduction rules, we use the following data reduction rule:

5. If there are $bk + 2$ rectangles $r_1, \ldots, r_{bk+2} \in R$ such that for each $i \in \{1, \ldots, bk + 1\}$ it holds that every vertical line that intersects r_i also intersects r_{bk+2}, then delete r_{bk+2}.

The correctness of this data reduction rule follows from the fact that k horizontal lines cannot intersect all rectangles $r_1, \ldots, r_{bk+1}$. Hence, if the instance with r_{bk+2} deleted is a yes-instance, every solution must contain a vertical line stabbing some of the rectangles $r_1, \ldots, r_{bk+1}$, and this line also stabs r_{bk+2} in the original instance, which, therefore, is also a yes-instance.

The following two observations are immediate consequences of data reduction rule 5.

Observation 3. *For every rectangle r in a reduced instance there are at most bk rectangles $r' \neq r$ with $\mathrm{lx}(r') \geq \mathrm{lx}(r)$ and $\mathrm{rx}(r') \leq \mathrm{rx}(r)$.*

Observation 4. *In a reduced instance, for every* $j \in \{1, \ldots, n\}$ *there are at most* $bk + 1$ *rectangles* r *with* $\mathrm{lx}(r) = j$.

Lemma 2. *For every rectangle* $r \in R$ *in a reduced instance it holds that* $\mathrm{rx}(r) \leq (bk + 1) \cdot \mathrm{lx}(r)$.

Proof. By induction on $\mathrm{lx}(r)$. Details are omitted due to lack of space. □

Lemma 3. *In a reduced instance, for every* $j \in \{1, \ldots, n-1\}$ *there is a rectangle* $r \in R$ *with* $\mathrm{lx}(r) > j$ *and* $\mathrm{rx}(r) \leq (bk + 1) \cdot j + 1$.

Proof. Assume for the sake of contradiction that there exists $j \in \{1, \ldots, n-1\}$ such that for every rectangle $r \in R$ with $\mathrm{lx}(r) > j$ it holds that $\mathrm{rx}(r) > (bk + 1) \cdot j + 1$. Consider a rectangle r' with $\mathrm{rx}(r') = (bk + 1) \cdot j + 1$. Such a rectangle exists by Observation 2. Then it holds that $\mathrm{lx}(r') \leq j$ due to our assumption. But by Lemma 2 we have $\mathrm{rx}(r') \leq (bk + 1) \cdot j$, a contradiction. □

Corollary 1. *Let* $q \leq n$, *and let* $\{v_{j_1}, v_{j_2}, \ldots, v_{j_q}\} \subseteq V$ *with* $j_1 < j_2 < \ldots < j_q$ *be a set of vertical lines stabbing all rectangles from* R *in a reduced instance. Then for every* $i \in \{1, \ldots, q\}$ *it holds that* $j_i \leq ((bk + 1)^i - 1)/bk$.

Proof. By induction on i. For $i = 1$, the statement holds because in any reduced instance there is a rectangle r with $\mathrm{lx}(r) = \mathrm{rx}(r) = 1$. Assume that the statement holds for $i - 1$, that is, $j_{i-1} \leq ((bk + 1)^{i-1} - 1)/bk$. By Lemma 3, there is a rectangle $r \in R$ with $\mathrm{lx}(r) > j_{i-1}$ and $\mathrm{rx}(r) \leq (bk + 1) \cdot j_{i-1} + 1$. Clearly this rectangle is not stabbed by any line from $\{v_{j_1}, \ldots, v_{j_{i-1}}\}$ and therefore, we have $j_i \leq \mathrm{rx}(r) \leq (bk + 1) \cdot j_{i-1} + 1 \leq ((bk + 1)^i - 1)/bk$. □

Observation 5. *If an instance of the restricted variant of* 2-Dimensional Rectangle Stabbing *is a* yes*-instance, then there is a set* $V' \subseteq V$ *of at most* bk *vertical lines that intersect all rectangles in* R.

Proof. Replace every horizontal line h in an optimal solution by at most b vertical lines that intersect the rectangles intersected by h. □

Theorem 6. *The restricted variant of* 2-Dimensional Rectangle Stabbing *where every horizontal line intersects at most* b *rectangles has a problem kernel of size* $O((bk + 1)^{bk})$ *and is therefore fixed-parameter tractable with respect to the combined parameters* k *and* b.

Proof. Given an instance of this restricted version, first find the optimal number of vertical lines needed to intersect all rectangles. As noted before, this is polynomial-time doable. If the optimal solution size is greater than bk, report that the given instance is a no-instance. Otherwise, by Corollary 1, we know that every set of vertical lines $\{v_{j_1}, \ldots, v_{j_{bk}}\}$ that intersects all rectangles in R has $j_{bk} \leq ((bk + 1)^{bk} - 1)/bk$. If the given instance is a yes-instance, then R cannot contain any rectangle r with $\mathrm{lx}(r) > j_{bk}$. For every $j \in \{1, \ldots, j_{bk}\}$, however, there are at most $bk + 1$ rectangles r with $\mathrm{lx}(r) = j$ due to Observation 4. Hence, if R contains more than $O((bk + 1)^{bk})$ rectangles, report that the given instance is a no-instance. □

5 Open Questions

We have shown that d-Dimensional Rectangle Stabbing with $d \geq 3$ is W[1]-hard. However, the parameterized complexity of the perhaps most interesting case $d = 2$ remains open, as well as that of 2-C1P-Set Cover. Even for the restriction of 2-Dimensional Rectangle Stabbing where no two rectangles from R "overlap" (two rectangles r_1, r_2 overlap if there exist a vertical line v and a horizontal line h that both intersect r_1 and r_2) we do not know the parameterized complexity.

Acknowledgement

We thank Michael R. Fellows for explaining us the unpublished "k-Multicolored Clique reduction technique".

References

1. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
2. Even, G., Rawitz, D., Shahar, S.: Approximation algorithms for capacitated rectangle stabbing. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 18–29. Springer, Heidelberg (2006)
3. Fellows, M.R.: Personal communication (September 2007)
4. Fellows, M.R., Hermelin, D., Rosamond, F.A., Vialette, S.: On the parameterized complexity of multiple-interval graph problems (manuscript, 2007)
5. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
6. 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)
7. Giannopoulos, P., Knauer, C., Whitesides, S.: Parameterized complexity of geometric problems. The Computer Journal (2007), doi:10.1093/comjnl/bxm053
8. Hassin, R., Megiddo, N.: Approximation algorithms for hitting objects with straight lines. Discrete Appl. Math. 30, 29–42 (1991)
9. Kovaleva, S., Spieksma, F.C.R.: Approximation of a geometric set covering problem. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 493–501. Springer, Heidelberg (2001)
10. Kovaleva, S., Spieksma, F.C.R.: Approximation algorithms for rectangle stabbing and interval stabbing problems. SIAM J. Discrete Math. 20(3), 748–768 (2006)
11. Langerman, S., Morin, P.: Covering things with things. Discrete Comput. Geom. 33(4), 717–729 (2005)
12. Mecke, S., Schöbel, A., Wagner, D.: Station location – complexity and approximation. In: Proc. 5th ATMOS, IBFI Dagstuhl, Germany (2005)
13. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)

Solving Medium-Density Subset Sum Problems in Expected Polynomial Time: An Enumeration Approach

Changlin Wan[1,2] and Zhongzhi Shi[1]

[1] Institute of Computing Technology, Chinese Academy of Sciences
Beijing 100080, China
[2] Graduate University of Chinese Academy of Sciences
Beijing 100080, China
changlin.wan@gmail.com

Abstract. The subset sum problem (SSP) can be briefly stated as: given a target integer E and a set A containing n positive integer a_j, find a subset of A summing to E. The *density* d of an SSP instance is defined by the ratio of n to m, where m is the logarithm of the largest integer within A. Based on the structural and statistical properties of subset sums, we present an improved enumeration scheme for SSP, and implement it as a complete and exact algorithm (EnumPlus). The algorithm always equivalently reduces an instance to be low-density, and then solve it by enumeration. Through this approach, we show the possibility to design a sole algorithm that can efficiently solve arbitrary density instance in a uniform way. Furthermore, our algorithm has considerable performance advantage over previous algorithms. Firstly, it extends the density scope, in which SSP can be solved in expected polynomial time. Specifically, It solves SSP in expected $O(n \log n)$ time when density $d \geq c \cdot \sqrt{n}/\log n$, while the previously best density scope is $d \geq c \cdot n/(\log n)^2$. In addition, the overall expected time and space requirement in the average case are proven to be $O(n^5 \log n)$ and $O(n^5)$ respectively. Secondly, in the worst case, it slightly improves the previously best time complexity of exact algorithms for SSP. The worst-case time complexity of our algorithm is proved to be $O(n \cdot 2^{n/2} - c \cdot 2^{n/2} + n)$, while the previously best result is $O(n \cdot 2^{n/2})$.

1 Introduction

Let us denote $\mathbb{N}_+$ as the set of positive integers. The subset sum problem is a classical NP-complete problem, in which one asks, given a set $A = \{a_1, a_2, ..., a_n\}$ with $a_j \in \mathbb{N}_+$ $(1 \leq j \leq n)$ and $E \in \mathbb{N}_+$, if there exists a subset $A' \subseteq A$ such that the sum of all elements of A' is E. More formally, the subset sum problem can be formulated as an integer programming problem:

$$\text{Maximize } z = \sum_{j=1}^{n} a_j x_j$$
$$\text{Subject to } \sum_{j=1}^{n} a_j x_j \leq E; \forall j, x_j = 0 \text{ or } 1.$$

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 300–310, 2008.

Extensive study has been conducted on SSP and its related problems: knapsack problem [1] and integer partition problem [2]. Many noticeable results have been achieved. For example, the hardness distribution of those problems are carefully investigated in [2] [3] [4] [5] et al., and it is now known that the hardness of SSP varies greatly with density d (see [6]).

Low-density: an instance with density $0 < d < c$, for some constant c, can be efficiently solved by lattice reduction based algorithms, e.g., [7] [8] [9]. However, these algorithms have two main limits. Firstly, they cannot solve instance with $d \geq c$ efficiently, though the bound of constant c is recently extended from 0.6463 to 0.9408. Secondly, they are not complete, i.e., they may fail to find any solution of an instance when the instance actually has solution.

High-density: an instance with density $d > c \cdot n/\log n$ can be efficiently solved by various techniques such as branch-and-bound, dynamic programming, and number theory analysis. Specifically, the algorithm YS87 [10] adopts branch-and-bound technique; NU69 [11] and HS74 [12] adopt dynamic programming technique; ST02 [13] adopts both branch-and-bound and dynamic programming; CFG89 [14] and GM91 [15] utilize number theory analysis. However, these algorithms have two main limits. Firstly, they cannot solve instance with $d \leq c \cdot n/\log n$ efficiently. Secondly, their average-case complexity is expected to increase with n, thus they have difficulty in handling large size instance.

Medium-density: an instance with density $c \leq d$ and $d \leq c \cdot n/\log n$ is usually hard to solve. As far as we know, the algorithm DenseSSP [6] is the only previous algorithm that works efficiently in part of this density scope. It solves uniformly random instances with density $d \geq 16n/(\log n)^2$ in expected polynomial time $O(n^{3/2})$.

Other than exact algorithms, it is worth to mention that highly efficient approximation methods (e.g., [16] [17]) can solve SSP at polynomial time and space cost. However, they cannot guarantee the exactness of their solutions. In this paper, we concentrate on solving SSP through exact methods, and we propose a complete and exact algorithm, which we call EnumPlus. The two main ingredients of EnumPlus are a new pruning mechanism and a new heuristic. Based on the structural property of subset sums, the pruning mechanism allows to dynamically partition the integer set into two parts and to prune branches in the search tree. Based on the statistical property of subset sums, the heuristic predicts which branch of the tree is more likely to contain the solution (and this branch is explored first by the algorithm).

1.1 Contributions

The main contribution of this work is two-fold. First, by equivalently reducing an instance to be low-density in linear time (see Section 4 and 6.2), we show the possibility to design a sole algorithm that can efficiently solve arbitrary density instance in a uniform way. Second, we propose a complete and exact algorithm that has considerable advantage over previous exact algorithms. Specifically, it extends the density scope, in which SSP can be solved in expected polynomial

time, and it slightly improves the previously best worst-case time complexity of exact algorithms for SSP.

1.2 Notation and Conventions

If it is not specifically mentioned, we assume that the elements of A are sorted in decreasing order ($a_1 > a_2 > .. > a_n$), and use S to denote the sum of A. Following the notation and description style of [1], we denote some basic notations that are used for the algorithm description as follows:

A_k denotes the subset $\{a_k, a_{k+1}, .., a_n\}$ of A;
S_k denotes the sum value of A_k $\left(= \sum_{j=k}^{n} a_j\right)$;
d_k denotes the density of A_k $\left(= \frac{n-k+1}{\log max\{a_j | a_j \in A_k\}}\right)$;
$W(E)$ denotes the number of solutions for a given target E and integer set A;
$\hat{x}_k$ denotes current partial solution $\{x_j = 0, 1 | 1 \leq j \leq k\}$;
$\hat{z}_k$ denotes current partial solution value $\left(= \sum_{j=1}^{k} a_j x_j\right)$;
$\hat{c}_k$ denotes current residual capacity $(= E - \hat{z}_k)$;
$\neg\hat{c}_k$ denotes current residual opposite capacity $(= S_{k+1} - \hat{c}_k)$;
$b_{MAX}|\,(A_k, \hat{c}_k)$ denotes the maximum subset sum of A_k while $b_{MAX} \leq \hat{c}_k$;
$b_{MIN}|\,(A_k, \hat{c}_k)$ denotes the minimum subset sum of A_k while $b_{MIN} \geq \hat{c}_k$.

2 Motivation

There are two main causes of performance discrepancy of different enumeration (searching) scheme. In the first place, the efficiency to prune infeasible solutions contributes to the performance both in the worst case and in the average case. In the second place, proper search strategy contributes to the performance in the average case. Specifically, algorithm HS74 has the best time complexity $O(n \cdot 2^{n/2})$ in the worst case. It enumerates all possible solutions following breadth first strategy; it prunes redundant branches by dividing the original problem into two subproblems and considering all equal subset sums as one state. However, HS74 does not work well in two situations. Firstly, when processing low-density instance, almost all subset sums are different to each other, thus few pruning can be made. Secondly, because of its breadth-first search strategy, HS74 is slow to approach solutions when the size, i.e. breadth, of an instance is considerable large.

The central idea of our approach is dynamically partitioning the original instance $A[1..n]$ to two sub-instances $A[1..k]$ and $A[k+1..n], 1 < k < n$. We treat the whole enumeration space as a binary tree (like the route colored by red in Figure 1) that is stemmed from $A[1]$ and ended by $A[n]$. During the enumeration of $A[1..n]$, all enumerated subset sums of $A[k+1..n]$ are organized as "block bounds", which serve as block barriers that can prevent further expending of the k-th level nodes. Therefore, for any partition point k, both $A[1..k]$ and A[k+1..n] are incrementally and simultaneously enumerated by enumerating $A[1..n]$ as a binary tree. In addition, a heuristic is utilized to accelerate the searching for

global solution. The heuristic predicts which branch of the tree is more likely to contain the answer. Therefore, a large problem is recursively reduced into a smaller one in linear time, and it has high possibility that the two problems have at least one common solution. To clarify the description of our algorithm, we present the main phases separately.

3 Branch and Prune

The pruning mechanism is inspired by the partition operation of HS74. In HS74, the original instance is divided into two sub-instances, and their subset sums are separately computed and stored in two lists. For any subset sum s_i in a list, if a subset sum s_j can be found in the other list such that $s_i + s_j = E$, a feasible solution is located. While HS74 explicitly partitions the the oriental instance only one time before enumeration, our algorithm implicitly performs partition multiple times during enumeration.

A "block bound" of an integer set A is defined as a two elements structure $[b_{MAX}, b_{MIN}]$, in which b_{MAX} and b_{MIN} are subset sums of A. Furthermore, a block bound must conform two constraints: (1) $b_{MAX} < b_{MIN}$; (2) no subset sum of A falls between b_{MAX} and b_{MIN}. Block bounds are recursively calculated as follows:

$$\begin{aligned}
&\text{If } \hat{c}_k \geq S_k, \qquad b_{MIN}|(A_k, \hat{c}_k) = S, b_{MAX}|(A_k, \hat{c}_k) = S_k.\\
&\text{If } \hat{c}_k \leq 0, \qquad b_{MIN}|(A_k, \hat{c}_k) = 0, b_{MAX}|(A_k, \hat{c}_k) = -a_{k+1}.\\
&\text{If } S_k > \hat{c}_k > 0, \; b_{MIN}|(A_k, \hat{c}_k) = min\left\{\begin{array}{l} b_{MIN}|(A_{k+1}, \hat{c}_k),\\ a_k + b_{MIN}|(A_{k+1}, \hat{c}_k - a_k)\end{array}\right\},\\
&\qquad\qquad\qquad\quad b_{MAX}|(A_k, \hat{c}_k) = max\left\{\begin{array}{l} b_{MAX}|(A_{k+1}, \hat{c}_k),\\ a_k + b_{MAX}|(A_{k+1}, \hat{c}_k - a_k)\end{array}\right\}.
\end{aligned}$$

Let us consider a sorted integer array $A[1..n]$, we create an n elements list $V[1..n]$. Each element $V[k]$ of $V[1..n]$ is a collection of block bounds of the integer set A_k. Therefore, if there is an integer $s = b_{MAX}$ (or b_{MIN}), $[b_{MAX}, b_{MIN}] \in V[k+1]$, and $s + \hat{z}_k = E$, a feasible solution for target integer E is located. If there is a block bound $[b_{MAX}, b_{MIN}] \in V[k+1]$ such that $b_{MAX} < \hat{c}_k < b_{MIN}$, we can determine that there is no subset sum s of A_{k+1} such that $s + \hat{z}_k = E$. In this way, a block bound $[b_{MAX}, b_{MIN}]$ of A_k acts as a bounded block that prevents all attempts to find target E in A_k when $b_{MAX} \leq E \leq b_{MIN}$. To describe the mechanism of block bound, a case that has an integer set $A[1..4] = \{52, 40, 30, 16\}$ and the target value $E = 69$ is illustrated in Figure 1.

As we can see in Figure 1, the first node $\hat{c}_1 = 69$ is expended to two nodes $\hat{c}_2 = \{69, 17\}$, i.e., finding $E = 69$ and 17 in subset $A[2..4]$. Suppose the node having larger E is always expended first, the first block bound $[16, 138]$ is generated when finding $E = 69$ in subset $A[4..4]$. Therefore, later finding of $E = 39, 29$ in subset $A[4..4]$ is blocked by the block bound $[16, 138]$. In the same way, let us observe the case of $k = 3$, the searching for $\hat{c}_3 = 29$ is finished with the generation of a block bound $[16, 30]$, and the later searching for $\hat{c}_3 = 17$ will be blocked by the block bound $[16, 30]$. When the enumeration is finished, 6 block bounds $\{[68, 70], [16, 30], [56, 70], [16, 30], [46, 138], [16, 138]\}$ are generated.

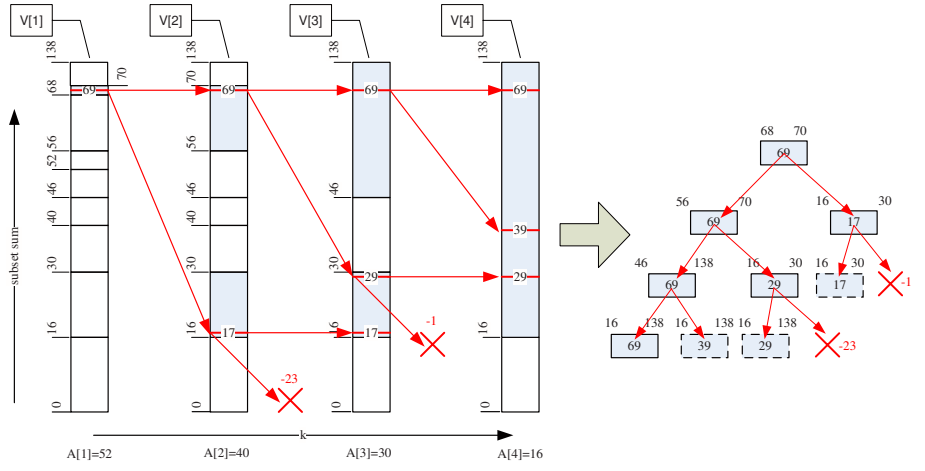


Fig. 1. The generation of block bounds for target value $E = 69$ and integer set $A[1..4] = \{52, 40, 30, 16\}$

4 Heuristic Search Strategy

Instead of pure depth-first or breadth-first search strategy, we introduce a new heuristic to accelerate the approach to feasible solution. At each state of enumeration, the expanding branch that has larger possibility to find feasible solution will be explored first. The heuristic is inspired by a previous study result of [3] in the context of canonical ensemble, which is usually studied in the physics literature. The main purpose of [3] is to study the property of the number of solutions in SSP, and then explain the experiential asymptotic behavior of $W(E)$. As [3] suggested, given uniformly random input integer set A and target value E, the number of solutions $W(E)$ is a central symmetric function with central point at $E = S/2$. Moreover, $W(E)$ monotonically increases with the increase of E in $[0, S/2]$. If we denote $\mathbf{Pr}[E]$ as the possibility of that there exists at least one solution of E, given two target value E_1 and E_2, we have that $\mathbf{Pr}[E_1] > \mathbf{Pr}[E_2]$ iff $|S/2 - E_2| > |S/2 - E_1|$. Suppose current partial solution is $\hat{x}_k$, weather $x_{k+1} = 1$ (i.e. $\hat{c}_{k+1} = \hat{c}_k - a_{k+1}$) or $x_{k+1} = 0$ (i.e. $\hat{c}_{k+1} = \hat{c}_k$) should be tried first is decided by the inequation:

$$|(\neg\hat{c}_k + \hat{c}_k)/2 - (\hat{c}_k - a_{k+1})| > |(\neg\hat{c}_k + \hat{c}_k)/2 - \hat{c}_k|\,. \tag{1}$$

Thus, we obtain the new heuristic: if inequation (1) holds, try $x_{k+1} = 0$ first, otherwise, try $x_{k+1} = 1$ first.

5 The New Algorithm

Based on the "block bound" and "heuristic search" techniques, we propose a complete and exact algorithm EnumPlus for SSP. In this algorithm, the whole

search space is enumerated as a binary tree T. For any given target value v at a branch node, the algorithm try to find both $b_{MAX}|(A_k, v)$ and $b_{MIN}|(A_k, v)$ in the sub-tree T_k that has x_k as root node. If the block bound is already existed in the block bound list $V[k]$, the existed block bound will be returned. Otherwise, T_k is expended to find the block bound $[b_{MAX}|(A_k, v), b_{MIN}|(A_k, v)]$, and the newly found block bound is inserted into $V[k]$ as a new element. The enumeration procedure terminates in 2 cases: 1) a feasible solution is found, 2) it is backtracked to the root of T. At each branch node (x_k) of T, if the target value v is more possible to be found when $x_k = 0$, then the branch $x_k = 0$ is enumerated first, otherwise the branch $x_k = 1$ is enumerated first.

The pseudo-code of EnumPlus and SetSum are given in Algorithm 1 and Algorithm 2 respectively, while the concrete implements of sub-algorithms QBB and UBB are not given since they can be implemented by simply adopting some classic data structures/algorithms (e.g., AVL-balance tree and Red-Black-balance tree).

Algorithm 1. EnumPlus($A[1..N], E$)

Input: an integer set $A[1..N]$; target value E.
Output: the maximum subset sum $b1 \leq E$; the minimum subset sum $b2 \geq E$.
1: allocate the vector of block bound sets $V[1..N]$;
2: $S \Leftarrow$ sum value of $A[1..N]$;
3: $[b1, b2] \Leftarrow$ SetSum$(1, E)$;
4: destroy the vector of block bound sets $V[1..N]$;
5: **return** $[b1, b2]$;

6 Performance Analysis

Before analyzing the complexity of our algorithm, we assume that the requirement of time and space of our algorithm is maximized when target value $E = S/2$. The assumption is reasonable because our algorithm simultaneously search both E and $S - E$ in the answer space. Moreover, $E = S/2$ is the hardest case for the dynamic programming algorithm (see [15]). Therefore, all our following analysis will be provided in case of that $S/2$ is chosen as target value E.

6.1 Worst-Case Complexity

Before the presentation of our results about the worst-case complexity of EnumPlus, we first introduce a lemma as follows:

Lemma 1. *For a certain subset $A[k..n]$ of A, the number of block bounds generated by SetSum is less than $min\{2^{k-1}, 2^{n-k+1}\}$.*

Proof. In case of $2^{k-1} \geq 2^{n-k+1}$, the number of all possible subset sums of $A[k..n]$ is less than $\left(2^{n-k+1} - 1\right)$, therefore the number of all possible block bounds of $A[k..n]$ is less than 2^{n-k+1}, i.e. $min\{2^{k-1}, 2^{n-k+1}\}$. In case of $2^{k-1} <$

Algorithm 2. SetSum(k, v)

Input: k = start position of residual subset $A[k..n]$;
v = residential capacity $\hat{c}$.
Output: $[b_{MAX}, b_{MIN}]$ = block bound of $A[k..N]$ for $\hat{c} = v$.

```
1: if v ≥ S_k return [S_k, S];
2: if v ≤ 0 return [−a_{k+1}, 0];
3: [b_MAX, b_MIN] ⇐ QBB(V[k], v); // query block bound [b_MAX, b_MIN] in V[k] such
   that b_MAX ≤ v ≤ b_MIN.
4: if b_MAX ≤ v ≤ b_MIN then
5:    if v = b_MAX or v = b_MIN then
6:       identify solution; halt; // found solution
7:    else
8:       return [b_MAX, b_MIN];
9:    end if
10: else if inequation 1 holds then
11:    [b3, b4] ⇐ SetSum(k + 1, v);
12:    [b1, b2] ⇐ SetSum(k + 1, v − A[k]); b1+ = A[k]; b2+ = A[k];
13: else
14:    [b1, b2] ⇐ SetSum(k + 1, v − A[k], v2); b1+ = A[k]; b2+ = A[k];
15:    [b3, b4] ⇐ SetSum(k + 1, v);
16: end if
17: b_MAX ⇐ max{b1, b3}; b_MIN ⇐ min{b2, b4};
18: UBB(V[k], b_MAX, b_MIN); // insert [b_MAX, b_MIN] into V[k]
19: return [b_MAX, b_MIN];
```

2^{n-k+1}, the search tree has at most 2^{k-1} nodes at level k. Because each node generates at most one block bound, the number of all possible block bounds of $A[k..n]$ is less than 2^{k-1}, i.e. $min\{2^{k-1}, 2^{n-k+1}\}$.

About the worst-case complexity of EnumPlus, there are 2 propositions given as follows:

Proposition 2. *The worst-case space complexity of EnumPlus is* $O(2^{n/2})$.

Proof. For a subset A_k, the number of block bounds generated by SetSum is less than $\min\{2^{k-1}, 2^{n-k+1}\}$, therefore the total number $\mathrm{Num}(n)$ of generated block bounds is

$$\mathrm{Num}(n) \leq \sum_{k=1}^{n} \min\{2^{k-1}, 2^{n-k+1}\} \leq 2 \times \sum_{k=1}^{n/2} 2^{k-1} \leq 2^{n/2+1}.$$

Thus the worst-case space complexity of EnumPlus is $O(2^{n/2})$.

Proposition 3. *The worst-case time complexity of EnumPlus is* $O(n \cdot 2^{n/2} - c \cdot 2^{n/2} + n)$.

Proof. As we proved in the proposition 2, there are at most $2^{n/2+1}$ block bounds are generated, and each recursive call for SetSum generates one block bound. The

main time cost of each block bound is to search and insert it in a collection $V[k]$. There are some classic data structures/algorithms, such as AVL-balance tree and Red-Black-balance tree, can efficiently manage the search and insert operations on storable data collection. The worst-case time cost of these algorithms to search or insert in the n elements collection is $\log n$. Therefore we have the worst-case time cost $\mathrm{Time}(n)$ of EnumPlus as follows:

$$\begin{aligned}\mathrm{Time}(n) &= 2 \times \sum_{k=1}^{n/2} \sum_{i=1}^{2^{k-1}} \lceil \log i \rceil = 2 \times \sum_{k=1}^{n/2} \sum_{i=1}^{k} ((i-1) \times 2^{i-2}) \\ &= 2 \times \sum_{k=1}^{n/2} ((k-2) \times 2^{k-1} + 1) \\ &\leq (n-6) \times 2^{n/2} + n + 8\end{aligned}$$

Thus the worst-case time complexity of algorithm EnumPlus is $O(n \cdot 2^{n/2} - c \cdot 2^{n/2} + n)$.

6.2 Average-Case Complexity

Before analyzing the average-case complexity of our algorithm, 2 lemmas are introduced as follows:

Lemma 4. *EnumPlus always reduces an instance A_1 with $\hat{c}_1 = S_1/2$ to A_k with $\hat{c}_k$, $|\hat{c}_k - S_k/2| \leq a_{k-1}/2$, in linear time.*

Proof. **[Induction]** We first consider $k = 2$. Because of the heuristic search strategy, EnumPlus first expends the branch that leads to a sub-problem, in which $|\hat{c}_k - S_k/2|$ is smaller. Then we have

$$\hat{c}_2 = \begin{cases} S_1/2, \text{if } |S_1 - S_2| \leq |S_1 - 2a_1 - S_2| \\ S_1/2 - a_1, \text{if } |S_1 - S_2| > |S_1 - 2a_1 - S_2| \end{cases}.$$

Therefore,

$$\hat{c}_2 - S_2/2 = \begin{cases} a_1/2, \text{if } |S_1 - S_2| \leq |S_1 - 2a_1 - S_2| \\ -a_1/2, \text{if } |S_1 - S_2| > |S_1 - 2a_1 - S_2| \end{cases}.$$

Thus EnumPlus reduces A_1 with $\hat{c}_1 = S_1/2$ to A_2 with $\hat{c}_2$, $|\hat{c}_2 - S_2/2| \leq a_1/2$, in 1 step.

Then we assume that EnumPlus reduces A_1 with $\hat{c}_1 = S_1/2$ to A_k with $\hat{c}_k$, $|\hat{c}_k - S_k/2| \leq a_{k-1}/2$, in $k-1$ steps.

Consider A_{k+1} and $\hat{c}_{k+1}$, we have

$$\hat{c}_{k+1} = \begin{cases} \hat{c}_k, \text{if } |\hat{c}_k - S_{k+1}| \leq |\hat{c}_k - 2a_{k+1} - S_{k+1}| \\ \hat{c}_k - a_k, \text{if } |\hat{c}_k - S_{k+1}| > |\hat{c}_k - 2a_{k+1} - S_{k+1}| \end{cases}.$$

Combine the above definition of $\hat{c}_{k+1}$ and the assumption that $|\hat{c}_k - S_k/2| \leq a_{k-1}/2$, we have that $|\hat{c}_{k+1} - S_{k+1}/2| \leq a_k/2$. Thus EnumPlus reduces A_1 with $\hat{c}_1 = S_1/2$ to A_{k+1} with $\hat{c}_{k+1}$, $|\hat{c}_{k+1} - S_{k+1}/2| \leq a_k/2$, in k steps.

Lemma 5. *Let $M = 2^m$ and the n elements of A is uniformly random in $[1..M]$, the number of distinct subset sums of A is expected to be $O(n^4)$.*

Proof. We use $S_1, ..., S_M$ to denote the sequence of all subsets of A listed in non-decreasing order of their sums. Let the sum of subset S_u be $P_u = \sum_{j \in S_u} a_j$. For any $2 \leq u \leq M$, define $\Delta_u = P_u - P_{u-1} \geq 0$, then P_u is a distinct subset sum if $\Delta_u > 0$, and P_1 is always a distinct subset sum. Let every element a_j of A be a non-negative random variable with density function $f_j : [1..M] \to [0, 1]$, i.e., $f_j(t) = \mathbf{Pr}(a_j = t), t \in [1..M]$. We notice that there is a theorem, which is proved by [18] for general discrete distributions, shows that:

Suppose $\pi = max_{j \in [1..n]}(max_{x \in [1..M]}(f_j(x))$ and $\mu \geq max_{j \in [1..n]}(\mathbf{E}[a_j])$. Then the expected number of dominating sets is $\mathbf{E}[q] = O(\mu n^2(1 - e^{-\pi n^2})) = O(\mu \pi n^4)$.

Because a distinct subset sum is a special case of dominating set on condition that weight w_j and profit p_j are both equal to a_j, the number of distinct subset sums is equal to the number of dominating sets on this condition. Since a_j is uniformly random in $[1..M]$, we have $\pi = 1/M$, $\mu = M/2$. Therefore, the number of distinct subset sums is expected to be

$$\mathbf{E}[q] = O(\frac{M}{2} \cdot \frac{1}{M} \cdot n^4) = O(n^4).$$

Assume that the elements of A is uniformly random in $[1..M]$, we have 2 propositions about the complexity of EnumPlus in the average case:

Proposition 6. *Given an integer set A whose elements are uniformly distributed, the overall expected time and space requirement of EnumPlus in the average case are $O(n^5 \log n)$ and $O(n^5)$.*

Proof. According to Lemma 5, the number of distinct subset sums of A is expected to be $O(n^4)$. Therefore the expected space cost is $O(n \cdot n^4) = O(n^5)$, and the expected time cost is $O(n \cdot n^4 \cdot \log(n^4)) = O(n^5 \log n)$. Thus the overall time and space complexity of EnumPlus are expected to be $O(n^5 \log n)$ and $O(n^5)$ respectively.

Proposition 7. *EnumPlus solves SSP in $O(n \log n)$ time when density $d \geq c \cdot \sqrt{n}/\log n$.*

Proof. Consider an integer set $A[1..n]$ whose elements are uniformly random in $[1..2^m]$. As the previous result shown by [3] and [2], if density $d > 1$, there is a high possibility that the instance with target value $S/2$ has many solutions. Thus it is expected that the sub-instance A_{n-m+1} with $S_{k-m+1}/2$ has many solutions, and it takes at most $O(m2^{m/2})$ time to locate these solutions. Furthermore, as we proved in Lemma 4, EnumPlus reduces instance $A[1..n]$ with $S/2$ to sub-instance A_k with $S_k/2$ in linear time. Thus the expected time complexity for the problem A with $S/2$ is $O(m2^{m/2}) + O(n)$. If $n = O(2^{m/2l})$, $\frac{m}{2 \log m} > l \geq 1$, then the expected time complexity for instance $A[1..n]$ with $S/2$ is $O(n^l \log n)$, and $d = O(\sqrt[2l]{2^m}/m) = O(\sqrt[2l]{n}/\log n)$. Thus EnumPlus solves SSP in $O(n \log n)$ time when density $d \geq c \cdot \sqrt{n}/\log n$.

6.3 Comparison of Related Works

Among the previously exact algorithms for SSP, HS74 has the best time complexity $O(n \cdot 2^{n/2})$ in the worst case. EnumPlus is an overall improvement of HS74, its worst-case time complexity is $O(n \cdot 2^{n/2} - c \cdot 2^{n/2} + n)$. As we described in Section 2, HS74 always reduce the original instance to 2 half size sub-instances. As we know, if the density of sub-instance is larger than 1, a solution is expected to be found by solving one sub-instance whose size is half of the original instance. However, by using a new heuristic, EnumPlus reduce the original instance to a smaller sub-instance, in which the solution can be found (see the proof of Proposition 7). Thus the performance of EnumPlus is better than HS74 in average case, especially when handling large size instance. Specifically, EnumPlus solves SSP in $O(n \log n)$ time when density $d \geq c \cdot \sqrt{n}/\log n$. This density bound is better than the density bound $d \geq c \cdot n/(\log n)^2$ of DenseSSP, which is the only previous algorithm working efficiently beyond the magnitude bound of $O(n/\log n)$. However, it must be noticed that the performance of EnumPlus is still not good enough when handling low-density instance. When density $d < 0.9408$, some incomplete algorithms, which are based on lattice reduction, are expected to outperform EnumPlus.

7 Conclusions and Future Work

In this work, we proposed a new enumeration scheme that utilizes both structural property and statistical property of subset sums to improve the efficiency of enumeration. The improved enumeration scheme is implemented as a complete and exact algorithm (EnumPlus). The algorithm always equivalently reduces an instance to be low-density, and then solve it by enumeration. Through this approach, we show the possibility to design a sole algorithm that can efficiently solve arbitrary density instance in a uniform way. Furthermore, our algorithm has considerable performance advantage over previous exact algorithms. It slightly improves the previously best time complexity of exact algorithms for SSP in the worst case; it extends the density scope to $d \geq c \cdot \sqrt{n}/\log n$, in which SSP can be solved in polynomial time. In addition, the overall expected time and space requirements are proved to be $O(n^5 \log n)$ and $O(n^5)$ respectively in the average case.

As we previously described, arbitrary density SSP instance can be equivalently reduced to and solved as low density instance by our approach. Thus the efficiency of EnumPlus mainly relies on efficiently solving low density problem. Since the lattice reduction approach shows particular efficiency when dealing low density instance, the integration of the two approaches may be a potential way to further improve the performance of our algorithm. Therefore, the relationship between lattice reduction and enumeration scheme is an important issue in our future work.

Acknowledgments. The authors thank the anonymous reviewers for their helpful comments, and Dr. Qiufeng Wang for assistance with the experiments.

References

1. Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., New York (1990)
2. Borgs, C., Chayes, J., Pittel, B.: Phase transition and finite-size scaling for the integer partitioning problem. Random Struct. Algorithms 19(3-4), 247–288 (2001)
3. Sasamoto, T., Toyoizumi, T., Nishimori, H.: Statistical mechanics of an np-complete problem: subset sum. Journal of Physics A: Mathematical and General 34, 9555–9567 (2001)
4. Bauke, H., Franz, S., Mertens, S.: Number partitioning as random energy model. Journal of Statistical Mechanics: Theory and Experiment (2004), P04003 http://arxiv.org/abs/cond-mat/0402010
5. Pisinger, D.: Where are the hard knapsack problems? Comput. Oper. Res. 32(9), 2271–2284 (2005)
6. Flaxman, A.D., Przydatek, B.: Solving medium-density subset sum problems in expected polynomial time. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 305–314. Springer, Heidelberg (2005)
7. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. J. ACM 32(1), 229–246 (1985)
8. Frieze, A.M.: On the lagarias-odlyzko algorithm for the subset sum problem. SIAM J. Comput. 15(2), 536–539 (1986)
9. Coster, M.J., Joux, A., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C.P., Stern, J.: Improved low-density subset sum algorithms. Comput. Complex. 2(2), 111–128 (1992)
10. Yanasse, H., Soma, N.: A new enumeration scheme for the knapsack problem. Discrete applied mathematics 18(2), 235–245 (1987)
11. Nemhauser, G., Ullmann, Z.: Discrete dynamic programming and capital allocation. Management Science 15(9), 494–505 (1969)
12. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. J. ACM 21(2), 277–292 (1974)
13. Soma, N., Toth, P.: An exact algorithm for the subset sum problem. European Journal of Operational Research 136(1), 57–66 (2002)
14. Chaimovich, M., Freiman, G., Galil, Z.: Solving dense subset-sum problems by using analytical number theory. J. Complex. 5(3), 271–282 (1989)
15. Galil, Z., Margalit, O.: An almost linear-time algorithm for the dense subset-sum problem. SIAM J. Comput. 20(6), 1157–1189 (1991)
16. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM 22(4), 463–468 (1975)
17. Kellerer, H., Mansini, R., Pferschy, U., Speranza, M.G.: An efficient fully polynomial approximation scheme for the subset-sum problem. J. Comput. Syst. Sci. 66(2), 349–370 (2003)
18. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. In: STOC 2003: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, pp. 232–241. ACM Press, New York (2003)

A Scalable Algorithm for Graph-Based Active Learning

Wentao Zhao, Jun Long, En Zhu, and Yun Liu

National University of Defense Technology, Changsha, Hunan 410073, China

Abstract. In many learning tasks, to obtain labeled instances is hard due to heavy cost while unlabeled instances can be easily collected. Active learners can significantly reduce labeling cost by only selecting the most informative instances for labeling. Graph-based learning methods are popular in machine learning in recent years because of clear mathematical framework and strong performance with suitable models. However, they suffer heavy computation when the whole graph is in huge size. In this paper, we propose a scalable algorithm for graph-based active learning. The proposed method can be described as follows. In the beginning, a backbone graph is constructed instead of the whole graph. Then the instances in the backbone graph are chosen for labeling. Finally, the instances with the maximum expected information gain are sampled repeatedly based on the graph regularization model. The experiments show that the proposed method obtains smaller data utilization and average deficiency than other popular active learners on selected datasets from semi-supervised learning benchmarks.

Keywords: Active Learning, Graph-based Learning.

1 Introduction

Supervised learning methods had been successfully used in many tasks. However, their application may be limited due to the inability to obtain sufficient labeled training instances because of heavy labeling cost associated with specific tasks. Active learning is an important approach to reduce the burden of labeling effort by only sampling the most informative instances to present for labeling by human experts.

A general active learning method comprises two parts: a learning engine and a sampling engine [1]. The whole process of active learning could be described as follows. Initially, a labeled training set L and an unlabeled set UL are available. Then, the learning engine trains a base classifier on the original training set L. After that, the sampling engine chooses the most informative instance x from the unlabeled instances and then has x labeled by human experts before $< x, c(x) >$ is added into the labeled set L where $c(x)$ is the label of x. Then the learning engine constructs a new classifier on the updated labeled set. The whole process runs repeatedly until the accuracy of the base classifier reaches the target value.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 311–322, 2008.

According to the sampling criterion used to select instances for labeling, the current research falls under three categories: uncertainty reduction, expected-error minimization and version space reduction [2]. The uncertainty reduction approach [3] selects the instances on which the current classifier has the least certainty of prediction. Many sampling methods apply the similar strategy [4,5]. The expected-error minimization approach [6,7] samples the instances that minimize the future expected error rate on the test set. Such methods expect to achieve the lowest error, but they are computationally expensive and their performance depends on the loss function which they chose to estimate the future error. The version space reduction approach [8,9] tries to select the instances that can reduce the volume of version space by half. Query-by-Committee is a representative method of this approach that constructs a committee consists of randomly selected hypotheses from the version space and selects the instances on which the disagreement within the committee is the greatest. The version space reduction approach also includes QBag [10], QBoost [10] and Active DECORATE [11].

A majority of these methods sample instances based on the prediction of the base classifier (or the version space) trained on labeled instances and ignore the effect of unlabeled instances. However, learning methods could be strengthened by unlabeled instances to train stronger classifiers on certain assumptions. For example, semi-supervised learning methods employ both unlabeled and labeled instances to generate more powerful classifiers on the smoothness assumption which says:

If two points x_1, x_2 in a high-density region are close, then so should be the corresponding outputs y_1, y_2.

Holding such assumption, semi-supervised learning methods could be used to strengthen active learners. Some researchers have made their contributions based on such idea. McCallum and Nigam [12] presented an active learning method which constructed the base classifier on labeled and unlabeled instances with the EM algorithm for text categorization. It is efficient in text categorization, but can only be suitable for the generative model. Muslea [2] proposed the multi-view active learning method which selected the instances with the largest disagreement from multi-view learners. However, it requires the learning task to be a multi-view one.

In recent years, graph-based methods are popular in semi-supervised learning due to clear mathematical framework and strong performance with suitable models. However, they suffer heavy computation when the whole graph is in huge size. In this paper, we propose a scalable algorithm for graph-based active learning which constructs a backbone graph instead of the whole graph using the MST(Minimum Spanning Tree) algorithm, then chooses the instances in the backbone graph for labeling first. After that, the active learner samples the instances with the maximum expected information gain based on the graph regularization model. The experiments show that the proposed method obtains smaller *data utilization* and *average deficiency* than other popular active learners on selected datasets from semi-supervised learning benchmarks.

The rest of the paper is organized as follows. Section 2 provides the basic notations. Section 3 presents the graph-based active learning method. Section 4 describes the proposed scalable algorithm for active learning called SGAL in detail. Section 5 shows the experimental results of the SGAL method as well as other methods on selected data sets. Section 6 draws the conclusion.

2 Preliminary

2.1 Basic Notation

Let X be the instance space and $x_i \in X$ represents an instance which is a feature vector $< x_{i1}, x_{i2}, \cdots, x_{im} >$. Let $Y = \{y_1, y_2, \cdots, y_p\}$ be the set of possible labels.

Let $c : X \to Y$ be the target function which classifies any $x \in X$ as an element in Y. The notion $< x, c(x) >$ denotes a labeled instance and $< x, ? >$ denotes an unlabeled instance where $? \in Y$. L denotes the whole set of labeled instances and U denotes the whole set of unlabeled instances.

There are l labeled instances: $< x_1, y_1 >, \cdots, < x_l, y_l >$, and u unlabeled instances:$< x_{l+1}, ? >, \cdots, < x_{l+u}, ? >$. We have $l \ll u$. The total number of instances is $n = l + u$.

Let $G =< V, E >$ be a connected graph with vertices V and edges E. Each vertex $v_i \in V$ represents an instance in $L \bigcup U$. Each edge $< v_j, v_k > \in E$ connects two vertices v_j and v_k. We have the adjacency matrix W_{nn} of G and its entry w_{ij} is the similarity between v_i and v_j. Here W_{nn} is given by Gaussian kernel of width σ:

$$w_{ij} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} \tag{1}$$

Furthermore, we define a diagonal matrix D, in which $D_{ii} = \sum_{j=1}^{n} w_{ij}$.

Let Y_{mp} be a $m \times p$ matrix on G, and y_{ik} in Y_{mp} is the probability for instance x_i to be labeled as $y \in Y$.

3 The Graph-Based Active Learning method

Here we present an active learning method based on the graph regularization model [13]. The graph regularization model, which is a semi-supervised learning method, employs label propagation on graphs for smoothness. In our method, the learning engine constructs the base classifier based on the graph regularization model and the sampling engine samples the instance with the largest expected information gain.

3.1 Learning Engine in GAL

Regularization on Graph. Based on the smoothness assumption, the graph G has two constraints: first, the labeled vertices should be consistent with the

initial labels; second, the labels on G should not be changed quickly between neighboring vertices. Then we obtain the following labeling cost[13]:

$$C(\hat{Y}) = \sum_{i=1}^{l} (\hat{y}_i - y_i)^2 + \mu \cdot (\frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (\hat{y}_i - \hat{y}_j)^2) \tag{2}$$

where y_i denotes the label of the instance x_i and $\hat{y}_i$ denotes the predicted label of the instance x_i. Then the first part of Eq.(2) evaluates the consistence with the initial labels, the second part of Eq.(2) evaluates the consistence with the smoothness assumption, and μ is a trade-off between them.

Moreover, $\sum_{i=1}^{l} (\hat{y}_i - y_i)^2$ can be rewritten as the matrix form: $\|\hat{Y}_l - Y_l\|^2$, and $\frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (\hat{y}_i - \hat{y}_j)^2$ can also be rewritten as the matrix form:

$$\frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (\hat{y}_i - \hat{y}_j)^2 = \hat{Y}^T L \hat{Y} \tag{3}$$

where $L = D - W$ is the un-normalized graph Laplacian.

The aim is to find $\hat{Y}$ which minimize the labeling cost according to Eq. (2). After calculating the derivative of Eq.(2) and setting it to 0, we obtain:

$$\hat{Y} = (S + \mu L)^{-1} S Y \tag{4}$$

where $S_{ii} = I_{[l]}(i)$ [13]. Then the new labels can be calculated by a simple matrix inversion. Eq.(4) shows that the predicted labels on unlabeled instances rely on the graph Laplacian L and the initial labels Y on labeled instances. Thus the graph regularization model can be viewed as that the original labels are propagated to the whole graph depending on the intrinsic structure of the graph.

Induction Learning on Graph. When a new instance x should be predicted by the regularization framework, we simply use the following function to calculate the label of x [13]:

$$\hat{y} = \frac{\sum_j W_X(x, x_j) \hat{y}_j}{\sum_j W_X(x, x_j)} \tag{5}$$

where W_X is the Gaussian kernel function as:

$$W_X(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} \tag{6}$$

We take Eq.(5) as the base classifier in the GAL method.

3.2 Sampling Engine in GAL

Just using the graph regularization model as the base classifier in learning engine is not enough for active learning. The sampling criterion should also be modified to fit for the graph Regularization model.

Maximizing Expected Information Gain. Let $H(x_i)$ be the entropy of label probability distribution of instance x_i. Then let $H(G)$ denote the sum of $H(x)$ on all instances. Then

$$H(G) = \sum_{x_i \in L \bigcup U} \sum_{k=1}^{p} (-y_{ik} \log y_{ik}) \tag{7}$$

Thus, $H(G)$ reflects the certainty of the label probability distribution on $L \bigcup U$. Larger $H(G)$ indicates more uncertain labels. If all instances were labeled, $H(G)$ equals to 0.

Therefore, we try to reduce $H(G)$ as possible as we can in learning tasks. When we select some instances for labeling, $H(x)$ on those instances were changed from some positive value to 0. Meanwhile, the process of graph regularization could also change $H(G)$. When the labels spread from labeled instances to those unlabeled instance, H decreased on those unlabeled instances. Since we train the semi-supervised classifier in learning engine, we should select the instance which could reduce $H(G)$ most after label propagation.

Let $IG(G, x_i)$ denote the information volume gained when x_i was labeled based on regularization model. Then

$$IG(G, x_i) = H(RG(G)) - H(RG(Lable(G, v)))) \tag{8}$$

where $RG(G)$ is the regularization operation on G, and $Lable(G, v)$ is the operation which labels v in G.

To find the most informative instance for labeling, we sample the instances with the maximum expected information gain. Thus, the sampling criterion is

$$ES_i = \sum_{y \in Y} p(y|x_i) Gain(G, v_i, y) \tag{9}$$

where $Gain(G, v_i, y)$ denotes the value of $IG(G, v_i)$ when labeling v_i as y and $p(y|x_i)$ denotes the probability of x_i being labeled as y. We sample the instances with the largest ES_i.

3.3 The Process of Graph-Based Active Learning

The process of GAL method is given in Algorithm 1.

4 The Scalable Graph-Based Active Learning method

The time complexity of the original graph-base active learning is $O(n^4)$ and the memory complexity of that is $O(n^2)$. It can not scale well when U is in large size. Thus we propose a scalable active learning method to deal with this problem. The method includes two aspects: backbone graph construction and two-stage active learning.

The main strategy of backbone graph construction is keeping a subset S of the unlabeled instances to represent U. It can be described as follows: first, we

Algorithm 1. The GAL method

Input: an initial labeled set L, an unlabeled set UL, a stopping criterion S, and an integer M which specifies the number of instances sampled in each iteration.
Begin:
Construct G, W, D;
repeat
1.For each instance $x_i \in UL$ compute

$$ES_i = \sum_{y \in Y} p(y|x_i) Gain(G, v_i, y) \tag{10}$$

2.Select a subset A of size M from UL in which instances x_i have the largest R_{x_i};

3.Remove A from UL;
4.Label instances in A;
5.Add A into L;
until the stopping criterion S is satisfied
End.
Output: The classifier I trained by the final labeled set L as Eq.(5).

construct a backbone graph B with the size of $m(m \ll n)$ on unlabeled instances to reduce the scale of the original graph and maintain the geometry in data; second, to utilize the impact of "label propagation", a few unlabeled instances of size r $(r \ll n)$ are chosen randomly according to the same distribution as U and added into S . Thus, the computing and storing expense could be limited to a reasonable level.

Furthermore, the two-stage active learning can be described as follows. First, we select all the instances in B for labeling and train the semi-supervised learning classifier on B. Second, we run the GAL method until the target accuracy is reached.

The details will be introduced in the following sections.

4.1 Backbone Graph Construction

MST Clustering. Since the backbone graph B should reflect the geometry of the original graph G, the area with high density should have representative nodes in B. Therefore, clustering technology is a good choice to obtain those high dense areas.

We choose the following method to divide U into clusters. First, we compute a minimum spanning tree on G. Second, we break the edge with the smallest similarity repeatedly until m clusters are generated.

Center Points Selection. After MST clustering, areas with high density are found. We choose one point in each cluster to represent all the points in it.

We define I_i as the *influence degree* of a point x_i in a cluster Clu.

$$I_i = \sum_{x_j \in Clu} w_{ij} \tag{11}$$

I_i reflects the influence of x_i on other points in the cluster Clu.

Thus, we select the point with the highest *influence degree* in each cluster as the center points and add them into the backbone graph.

Random Points Selection. The vertices of the backbone graph represent the high dense areas of U, thus they maintain the geometry structure of G. However, such backbone graph can not propagate labels because the points in it share very low similarity. Then we choose a few unlabeled instances of size r ($r \ll n$) at random according to the same distribution as U.

4.2 Two-Stage Active Learning

In the first stage of our SGAL, we simply select the points in B for labeling. Those points have the largest *influence degree* in each cluster.

In the second stage of our SGAL, we run the standard graph-based active learning method based on the reduced unlabeled instances.

The process of the SGAL method is given in Algorithm 2.

Algorithm 2. The SGAL method

Input: an initial labeled set L, an unlabeled set UL, a stopping criterion S, and an integer M which specifies the number of instances sampled in each iteration.
Begin:
Construct G, W, D;
Run MST algorithm on G and generate clusters as section 4.1 describes;
Select center points in each cluster as B and random points, keep them in the unlabeled instances and abandon the rest unlabeled instances;
Remove B from UL;
Label instances in B;
Add B into L;
repeat
1.For each instance $x_i \in UL$ compute

$$ES_i = \sum_{y \in Y} p(y|x_i) Gain(G, v_i, y) \tag{12}$$

2.Select a subset A of size M from UL in which instances x_i have the largest ES_i;
3.Remove A from UL;
4.Label instances in A;
5.Add A into L;
until the stopping criterion S is satisfied
End.
Output: The classifier I trained by the final labeled set L as Eq.(5).

4.3 Efficiency Improving

The original GAL method should compute matrix inversion for each unlabeled instance in each iteration. Thus the total requirements for computation and storing are $O(n^4)$ and $O(n^2)$, respectively.

In the SGAL method, the size of the constructed graph was reduced from $l+u$ to $l+r+m$, where $l, m, r \ll u$.

For MST construction, we can use Prim's or Kruskal's algorithm. The running time of Prim's algorithm is $O(e \log n)$ while the running time of Kruskal's algorithm is $O(e \log e)$, where e denotes the number of edges in G. Which algorithm should be chosen depends on the problem to resolve.

Moreover, the running time of the two-stage active learning is $O(r \cdot (l+r)^3)$ and the storing expense of that is $O((l+r+m)^2)$.

5 Experiment

5.1 Methodology

To evaluate the performance of our SGAL method, we conducted a series of experiments. Four representative active learning algorithms were tested:

- Random sampling: choosing the instance at random;
- Uncertainty sampling: choosing the instance with the largest uncertainty of prediction, as in [3];
- QBC sampling: choosing the instance that the committee members disagree with most, as in [8];
- SGAL sampling(the method introduced in this paper).

Naive bayes was selected to be the base classifier of all other active learners. 10-fold cross-validation was used to obtain the target accuracy of the base classifier. The target accuracy is defined as the accuracy obtained by the base learning method trained on the whole dataset. All results presented were averages of ten runs. The committee size in QBC were set to 5. For our proposed SGAL method, we set the size of B and R to 20 and 200, respectively.

We divided each dataset into 10 equal partitions at random and each in turn was used for testing and the remainder was used as the sampling set. Before the test started, the sampling set was divided into two parts: one was the labeled set and another was the unlabeled set. The labeled set contained only one instance selected randomly and the unlabeled set contained all the rest. When the test started, the active learner sampled 1 instance from the unlabeled set for labeling in each iteration. While the active learner reached the target accuracy, the test stopped.

5.2 Datasets

The experiments were conducted on *g241c*, *handwritten digits*, *coil* and *sec-str5000*. These datasets were from the benchmarks of *Semi-supervised learning*

[13]. The reason we selected these datasets for experiments is that they can be easily obtained to compare different active learners and are widely accepted as the benchmarks for semi-supervised learning.

Table 1 shows the basic properties of these datasets.

Table 1. Basic properties of the datasets

Data set	Classes	Dimension	Instances	Comment
g241c	2	241	1500	artificial
digit1	2	241	1500	artificial
coil	6	241	1500	natural
secstr5000	2	315	5000	natural

5.3 Metrics

Two metrics were used to compare the performance of different active learners: *data utilization* [11] and *average deficiency* [14].

Data utilization is defined as the number of sampling that an active learner requires to reach the target accuracy. This metric reflects how efficiently the active learner can use the data. Smaller values of *data utilization* indicate more efficient active learning. Moreover, it is employed by many other researchers [11,10].

Average deficiency is used to evaluate how much an active learner could improve accuracy over random sampling. It is defined as:

$$Def_n(Active) = \frac{\sum_{t=1}^{n}(Acc_n(Random) - Acc_t(Active))}{\sum_{t=1}^{n}(Acc_n(Random) - Acc_t(Random))} \tag{13}$$

where n denotes the size of the whole unlabeled set, *Active* denotes the active learner we want to evaluate, *Random* denotes the random sampling method, and $Acc_t(Active)$ denotes the average accuracy achieved by *Active* after t sampling. Furthermore, the value of $Def_n(Active)$ is always non-negative and smaller values in $[0, 1)$ indicate more efficient active learning [14].

5.4 Results

We summarized the *data utilization* of the different active learners in Table 2 and *deficiency* in Table 3. In Table 2 and Table 3, the least *data utilization* and the least *deficiency* are marked in bold in each row. In the head of the tables , target accuracy is denoted by TA.

According to Table 2 and Table 3, it shows that our SGAL method has a superior performance than other sampling methods on most datasets. On *secstr5000*, the SGAL method requires just 376 labeling to reach the target accuracy while the random sampling method requires 564. The former obtains almost 1/3 labeling saving.

Table 2. Average data utilization of the different active learners

Data set	Random	Uncertain	QBC	SGAL	TA
g241c	291	268	349	**220**	81.23%
digit1	74	55	110	**41**	95.56%
coil	288	351	172	**156**	64.42%
secstr5000	564	552	511	**376**	65.29%

Table 3. Average deficiency of the different active learners

Data set	Uncertain	QBC	SGAL
g241c	0.7525	0.6096	**0.5251**
digit1	0.2436	0.1873	**0.1149**
coil	0.6690	0.5979	**0.5036**
secstr5000	1.1854	1.2275	**0.6391**

Figure 1-4 show the learning curves on *g241c*, *digit1*, *coil* and *secstr5000* . In all these figures, the vertical axis shows the accuracy of the classifiers and the horizontal axis shows the number of labels. To compare the beginning stage of SGAL method with other methods, we recorded the accuracy of the SGAL method after each instance in the backbone graph was selected for labeling.

In Figure 1, all the learning curves climb substantially. The SGAL method finally reaches the highest point.

In Figure 2, 3 and 4, our SGAL method almost outperforms the other active learners throughout the whole learning curve. The SGAL method starts with a very sharp increase and then rises steadily in these figures.

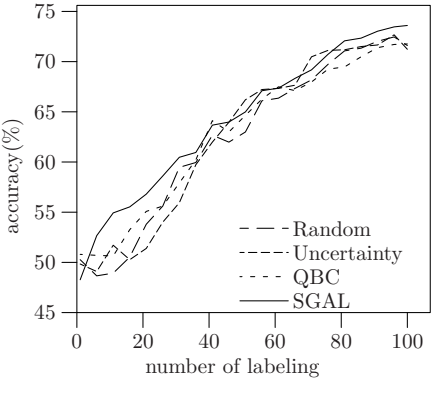


Fig. 1. learning curves on *g241c*

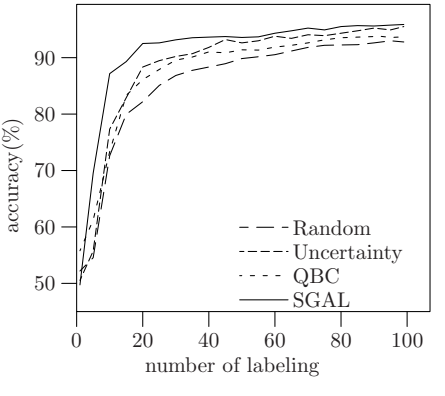


Fig. 2. learning curves on *digit1*

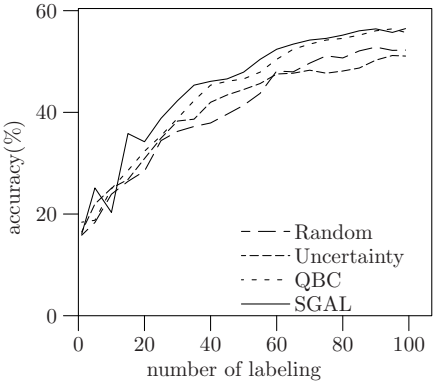


Fig. 3. learning curves on *coil*

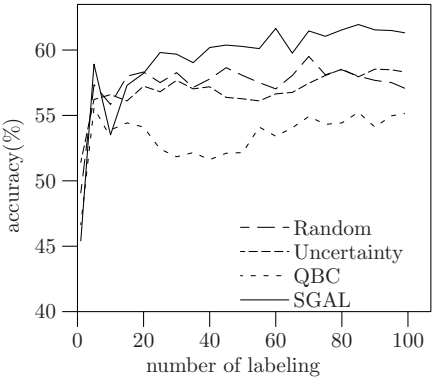


Fig. 4. learning curves on *secstr5000*

It is interesting that our SGAL gets very prominent performance improving than other methods at the beginning of all tests. This indicates that instances in the backbone graph play a key role in accuracy promoting.

6 Conclusion

In this paper, we use graph-based semi-supervised learning, which employs unlabeled instances when training, to strengthen active learning. However, standard graph-based learning methods suffer heavy burden of computing and memory requirements. For consideration of efficiency saving, we propose a scalable algorithm for graph-based active learning which constructs a backbone graph instead of the whole graph using MST algorithm, then chooses the instances in the backbone graph for labeling first. After that, the active learner selects the instances which could gain the maximum expected information gain based on the graph regularization model. Thus the computing complexity and memory complexity become $O(r \cdot (l+r)^3)$ and $O((l+r+m)^2)$, respectively. The experiments show that the proposed method outperforms the traditional sampling methods on most selected datasets.

We make several contributions in this paper. First, a general framework for semi-supervised learning method to strengthen active learning was proposed. Second, a backbone graph construction method for complexity reduction was provided. Third, the strategy that selects instances according to global geometry of unlabeled instances was presented.

We would like to pursue the following directions: extending the proposed method to the imbalanced-data problem and developing efficient algorithms for active learning on manifold structures.

Acknowledgments. This research was supported by the National Natural Science Foundation of China (No.60603015, 60603062).

References

1. Hieu, T.N., Arnold, S.: Active learning using pre-clustering. In: Proc. 21th International Conf. on Machine Learning, Banff. Morgan Kaufmann (2004)
2. Muslea, I., Minton, S., Knoblock, C.A.: Active learning with multiple views. Journal of Artificial Intelligence Research 27, 203–233 (2006)
3. Lewis, D.D., Gale, W.A.: A sequential algorithm for training text classifiers. In: 17th ACM International Conference on Research and Development in Information Retrieval, pp. 3–12. Springer, Heidelberg (1994)
4. Tong, S., Koller, D.: Support vector machine active learning with applications to text classification. Journal of Machine Learning Research 2, 45–66 (2001)
5. Schohn, G., Cohn, D.: Less is more: Active learning with support vector machines. In: Proc. 17th International Conf on Machine Learning, pp. 839–846. Morgan Kaufmann, San Francisco (2000)
6. Cohn, D.A., Ghahramani, Z., Jordan, M.I.: Active learning with statistical models. Journal of Artificial Intelligence research 4, 129–145 (1996)
7. Roy, N., McCallum, A.: Toward optimal active learning through sampling estimation of error reduction. In: Proc. 18th International Conf. on Machine Learning, pp. 441–448. Morgan Kaufmann, San Francisco (2001)
8. Seung, H.S., Opper, M., Sompolinsky, H.: Query by committee. In: Proceedings of the Fifth Workshop on Computational Learning Theory, pp. 287–294. Morgan Kaufmann, San Mateo (1992)
9. Freund, Y., Seung, H.S., Shamir, E., Tishby, N.: Selective sampling using the query by committee algorithm. Machine Learning 28, 133–168 (1997)
10. Abe, N., Mamitsuka, H.: Query learning using boosting and bagging. In: Proc. 15th International Conf. on Machine Learning, Madison, pp. 1–10. Morgan Kaufmann (1998)
11. Melville, P., Mooney, R.J.: Diverse ensembles for active learning. In: Proc. 21th International Conf. on Machine Learning, Banff, pp. 584–591. Morgan Kaufmann (2004)
12. McCallum, A., Nigam, K.: Employing em and pool-based active learning for text classification. In: ICML, pp. 350–358 (1998)
13. Chapelle, O., Schölkopf, B., Zien, A. (eds.): Semi-Supervised Learning. MIT Press, Cambridge (2006)
14. Baram, Y., El-Yaniv, R., Luz, K.: Online choice of active learning algorithms. In: ICML, pp. 19–26 (2003)

A Supervised Feature Extraction Algorithm for Multi-class

Shifei Ding[1,2], Fengxiang Jin[3], Xiaofeng Lei[1], and Zhongzhi Shi[2]

[1] School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221008 China
[2] Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080 China
[3] College of Geoinformation Science and Engineering, Shandong University of Science and Technology, Qingdao 266510 P.R. China
dingsf@cumt.edu.cn

Abstract. In this paper, a novel supervised information feature extraction algorithm is set up. Firstly, according to the information theories, we carried out analysis for the concept and its properties of the cross entropy, then put forward a kind of lately concept of symmetry cross entropy (SCE), and point out that the SCE is a kind of distance measure, which can be used to measure the difference of two random variables. Secondly, Based on the SCE, the average symmetry cross entropy (ASCE) is set up, and it can be used to measure the difference degree of a multi-class problem. Regarding the ASCE separability criterion of the multi-class for information feature extraction, a novel algorithm for information feature extraction is constructed. At last, the experimental results demonstrate that the algorithm here is valid and reliable, and provides a new research approach for feature extraction, data mining and pattern recognition.

1 Introduction

Feature extraction is one of the most importmant steps in pattern recognition, data mining, machine learning and so on[1,2]. In order to choose a subset of the original features by reducing irrelevant and redundant, many feature selection algorithms have been studied. The literature contains several studies on feature selection for unsupervised learning in which he objective is to search for a subset of features that best uncovers "natural" groupings (clusters) from data according to some criterion. For example, principal components analysis (PCA) is an unsupervised feature extraction method that has been successfully applied in the area of face recognition, feature extraction and feature analysis[3-5]. But the method of PCA is effective to deal with the small size and low-dimensional problems, and gets the extensive application in Eigenface and feature extraction. In high-dimensional cases, it is very difficult to compute the principal components directly[6]. Fortunately, the algorithm of Eigenfaces artfully avoids this difficulty by virtue of the singular decomposition technique. Thus, the problem of calculating the eigenvectors of the total covariance matrix, a high-dimensional matrix, is transformed into a problem of calculating the eigenvectors of a much lower dimensional matrix[7].

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 323–329, 2008.

Now an important question is how to deal with supervised information feature extraction. For supervised feature extraction problem, some authors have studied by discriminate analysis, bayes decision theory et al. But these methods depend on probability distributions of some classifications. In this paper, the authors have studied this field on the basis of these aspects. Firstly, we study and discuss the information theory, cross entropy theory, and point out its shortage. Secondly, a new concept of symmetry cross entropy (SCE) is put forward, and proved that the SCE is a kind of distance measure. At the same time, based on the SCE, we give the average SCE, i.e. ASCE. Which is regarded multi-class separability criterion. Thirdly, according to ASCE, a new information feature extraction algorithm is constructed. At last, the proposed algorithm here is tested in practice, and the experimental results indicate that it is efficient and reliable.

2 Feature Extraction Algorithm

In order to set up information feature extraction algorithm, we firstly discuss the following new concept of symmetry cross entropy and feature extraction theorem.

2.1 Symmetry Cross Entropy

Shannon[8] put forward the concept of information entropy for the very first time in 1948. The cross entropy (CE), or the relative entropy, is used for measuring difference information between the two probability distributions. But the CE satisfies only nonnegativity, normalization and dissatisfies symmetry and triangle inequation. For this reason, we carry out the improvement, and give the following definition.

Definition 1. Let X be a discrete random variable with two probability distribution vectors P and Q, where $P=(p_1,p_2,\cdots,p_n)$, $Q=(q_1,q_2,\cdots,q_n)$, the CE between P and Q is defined as

$$H(P\|Q)=\sum_{i=1}^{n}p_i\log\frac{p_i}{q_i}=E\left(\log\frac{p_i}{q_i}\right) \tag{1}$$

In the above definition denoted by formula (1), we show that the CE is always nonnegative and is zero if and only if $p_i=q_i$. However, it is not a true distance between distributions since it is not symmetric and does not satisfy the triangle inequality. In order to make it true distance between distributions, we improve the CE as follows.

Definition 2. Suppose that the $H(P\|Q)$ and $H(Q\|P)$ are CEs of P to Q and Q to P respectively, the symmetric cross entropy (SCE) between P and Q, denoted by $D(P,Q)$, defined as

$$D(P,Q)=H(P\|Q)+H(Q\|P)$$

$$=\sum_{i=1}^{n}p_i\log p_i+\sum_{i=1}^{n}q_i\log q_i-\sum_{i=1}^{n}p_i\log q_i-\sum_{i=1}^{n}q_i\log p_i \tag{2}$$

It is called Symmetric Cross Entropy (SCE) of $\boldsymbol{P}$ and $\boldsymbol{Q}$.

According to the definition of the SCE, we have the following theorem.

Theorem 1. Suppose that the SCE is defined by formula (2), then the SCE is a kind of distance measure, i.e. $D(P,Q)$ satisfies basic properties as follows.

(Ⅰ) Non-negativity: $D(P,Q) \geq 0$, and $D(P,Q) = 0 \Leftrightarrow P = Q$;

(Ⅱ) Symmetry: $D(P,Q) = D(Q,P)$;

(Ⅲ) Triangle inequation: Suppose that $W = (w_1, w_2, \cdots, w_n)$ is another probability distribution vector of the discrete random variable X, then

$$D(P,Q) \leq D(P,W) + D(W,Q) \tag{3}$$

Therefore, the SCE is a distance measure, which can be used to measure the degree of variation between two random variables. The SCE is considered as separability criterion of the two-class for information feature extraction. It can be seen that the smaller the SCE is, the smaller the difference of two-class. In particular, when the SCE=0, the two-class are same completely. For information feature extraction, under the condition of the given reduction dimensionality denoted by d, we should select such d characteristics that make the value of the SCE approach maximum. For convenience, we use the following function, denoted by $H(P,Q)$, in instead of above the SCE.

$$H(P,Q) = \sum_{i=1}^{n} (p_i - q_i)^2 \tag{7}$$

For a multi-class problem, based on the formula (4), the SCE is computed for every class i and j, where i and j denote number of class

$$H_{ij} = \sum_{k=1}^{n} (p_k^{(i)} - p_k^{(j)})^2 \tag{5}$$

The average symmetric cross entropy (ASCE) can be expressed as follows

$$H = \sum_{i=1}^{M} \sum_{j=1}^{M} p_k^{(i)} p_k^{(j)} d_{ij} = \sum_{i=1}^{M} \sum_{j=1}^{M} \sum_{k=1}^{n} p_k^{(i)} p_k^{(j)} (p_k^{(i)} - p_k^{(j)})^2 \tag{6}$$

being equivalent to the SCE, we should select such d characteristics that make the value of H approach maximum. In fact, H approaching maximum is equivalent to H_{ij} approaching maximum, so information feature extraction for a multi-class problem is also equivalent to a two-class problem.

In order to set up the information feature extraction algorithm, we first give the following theorem.

Theorem 2. Suppose $\{x_j^{(1)}\}$ $(j=1,2,\cdots,N_1)$ and $\{x_j^{(2)}\}$ $(j=1,2,\cdots,N_2)$ with covariance matrices $G^{(1)}$ and $G^{(2)}$ are squared normalization feature vectors, so-called squared normalization indicates

$$\sum_{k=1}^{n}(x_{jk}^{(i)})^2=1 \tag{7}$$

where $x_{jk}^{(i)}(i=1,2)$ denotes the kth feature component of the feature vector $x_j^{(i)}$. Then the SCE, i.e. the $H(P,Q)$ =maximum if and only if the coordinate system is composed of d eigenvectors corresponding to the first d eigenvalues of the matrix $A=G^{(1)}-G^{(2)}$.

So for $\{\boldsymbol{X}_j^{(1)}\}$ $(j=1,2,\cdots,N_1)$ and $\{X_j^{(2)}\}$ $(j=1,2,\cdots,N_2)$ $(j=1,2,\cdots,N_2)$ with covariance matrices $G^{(1)}$ and $G^{(2)}$ are squared normalization feature vectors. The k-th feature component of $\boldsymbol{X}_{\boldsymbol{j}}^{(\boldsymbol{i})}$ is denoted by $x_{jk}^{(i)}(i=1,2;k=1,2,\cdots,n)$, and the square mean of each component for every class is $\gamma_k^{(i)}=\frac{1}{N_i}\sum_{j=1}^{N_i}(x_{jk}^{(i)})^2$, where $i=1,2;j=1,2,\cdots,n$. Obviously $\gamma_k^{(i)}\geq 0$, and then

$$\sum_{k=1}^{n}\gamma_k^{(i)}=\sum_{k=1}^{n}\frac{1}{N_i}\sum_{j=1}^{N_i}(x_{jk}^{(i)})^2=\sum_{j=1}^{N_i}\frac{1}{N_i}\sum_{k=1}^{n}(x_{jk}^{(i)})^2=\sum_{j=1}^{N_i}\frac{1}{N_i}=1 \tag{8}$$

Namely $\gamma_k^{(i)}\geq 0$ and $\sum_{k=1}^{n}\gamma_k^{(i)}=1$. Therefore, we can comprehend $\{\gamma_k^{(i)}\}$ as the probability distribution defined by $\boldsymbol{X}_{\boldsymbol{j}}^{(\boldsymbol{i})}$. Suppose that the (k,l) element of symmetric matrix $\boldsymbol{G}^{(\boldsymbol{i})}$ $(i=1,2)$ is $g_{kl}^{(i)}=\frac{1}{N_i}\sum_{j=1}^{N_i}x_{jk}^{(i)}x_{jl}^{(i)}$. Record $\gamma^{(i)}=(\gamma_1^{(i)},\gamma_2^{(i)},\cdots,\gamma_n^{(i)})$, then every components of $\gamma^{(i)}$ is element of $\boldsymbol{G}^{(\boldsymbol{i})}$ $(i=1,2)$ in diagonal line. Let

$$s=s(\gamma^{(1)},\gamma^{(2)})=\sum_{k=1}^{n}(\gamma_k^{(1)}-\gamma_k^{(2)})^2 \tag{9}$$

2.2 Feature Extraction Algorithm

Suppose three classes C_1, C_2, and C_3 with covariance matrices $G^{(1)}, G^{(2)}$ and $G^{(3)}$ are squared normalization feature vectors. According to the discussion above, an

algorithm of information feature extraction based on the ASCE is derived and is as follows.

Step 1. Data pretreatment. Perform square normalization transformation for two classes original data according to the formula (7), and get data matrix $x^{(1)}, x^{(2)}, x^{(3)}$ respectively.

Step 2. Compute symmetric matrix A, B, C. Calculate the covariance matrixes $G^{(1)}, G^{(2)}, G^{(3)}$ and then get symmetric matrix as follows.

$$A = G^{(1)} - G^{(2)}\ ,\ B = G^{(1)} - G^{(3)}\ ,\ C = G^{(2)} - G^{(3)} \tag{10}$$

Step 3. Calculate all eigenvalues and corresponding eigenvectors of the matrix A according to Jacobi method.

Step 4. Construct extraction index. The total sum of variance square is denoted by

$$V_n = \sum_{k=1}^{n} \lambda_k^2\ ,\ V_d = \sum_{k=1}^{d} \lambda_k^2 \tag{11}$$

and then the variance square ratio (VSR) is VSR= $V_d / V_n = V_d / s_0$. The VSR value can be used to measure the degree of information extraction. Generally speaking, so long as $V_i \geq 80\%$, the purpose of feature extraction is reached.

Step 5. Construct extraction matrix. When $V_i \geq 80\%$, we select d eigenvectors corresponding to the first d eigenvalues, and construct the information extraction matrix $T = (u_1, u_2, \cdots, u_d)$.

Step 6. Feature extraction. The data matrixes $x^{(1)}, x^{(2)}, x^{(3)}$ is transformed by

$$y^{(i)} = T' x^{(i)} (i = 1,2,3) \tag{12}$$

and the purpose to compress the data information is attained.

2.3 Experimental Results

The original data sets come from reference[9], they are divided into three classes C_1, C_2, and C_3, and denote light occurrence, middle occurrence, and heavy occurrence about the occurrence degree of the pests respectively.

According to the algorithm set up above, and applying the DPS data processing system, the compressed results for three classes are expressed in Fig. 1.

Fig.1. shows that the distribution of feature vectors after compressed for the class C_1 denoted by "+", the class C_2 denoted "*" and the class C_3 denoted "^", is obviously concentrated relatively, meanwhile for these three classes, the within-class distance is small, the between-class distance is big, and the ASCE is maximum. Therefore, 2-dimensional pattern vector loaded above 99% information contents of the original 5-dimensional pattern vector. The experimental results demonstrate that the algorithm presented here is valid and reliable, and takes full advantage of the class-label information of the training samples.

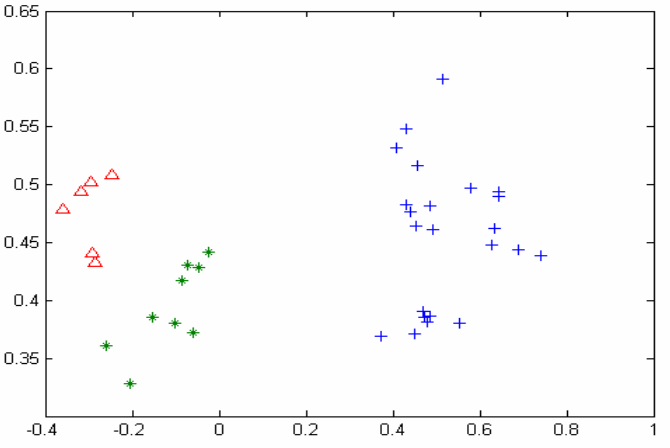


Fig. 1. The compressed results for three classes

3 Conclusions

From the information theory, studied and discussed the compression problem of the information feature in this paper, and come to a conclusion. According to the definition of the CE, a new concept of the SCE is proposed, and proved that the SCE is a distance measure which can be used to measure the degree of two-class random variables. The average SCE (ASCE) is given based on SCE, and it is to measure the difference degree for the multi-class problem. Regarding the ASCE separability criterion of the multi-class for information feature compression, we design a novel information feature compression algorithm. The experimental results show that algorithm presented here is valid, and compression effect is significant.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant No.40574001, 863 National High-Tech Program under Grant No. 2006AA01Z128, and the Opening Foundation of the Key Laboratory of Intelligent Information Processing of Chinese Academy of Sciences, under Grant No.IIP2006-2.

References

1. Duda, R.O., Hart, P.E. (eds.): Pattern Classification and Scene Analysis. Wiley, New York (1973)
2. Fukunaga, K. (ed.): Introduction to Statistical Pattern Recognition, 2nd edn. Academic Press, London (1990)
3. Ding, S.F., Shi, Z.Z.: Supervised Feature Extraction Algorithm Based on Improved Polynomial Entropy. Journal of Information Science 32(4), 309–315 (2006)
4. Hand, D.J. (ed.): Discrimination and Classification. Wiley, New York (1981)

5. Nadler, M., Smith, E.P. (eds.): Pattern Recognition Engineering. Wiley, New York (1993)
6. Yang, J., Yang, J.Y.: A Generalized K-L Expansion Method That Can Deal With Small Sample Size and High-dimensional Problems. Pattern Analysis Applications 6(6), 47–54 (2003)
7. Zeng, H.L., Yu, J.B., Zeng, Q.: System Feature Reduction on Principal Component Analysis. Journal of Sichuan Institute of Light Industry and Chemical Technology 12(1), 1–4 (1999)
8. Shannon, C.E.: A Mathematical Theory of Communication. Bell Syst. Tech. J. 27, 379–423 (1948)
9. Tang, Q.Y., Feng, M.G. (eds.): Practical Statistics and DPS Data Processing System. Science Press, Beijing (2002)

An Incremental Feature Learning Algorithm Based on Least Square Support Vector Machine

Xinwang Liu, Guomin Zhang, Yubin Zhan, and En Zhu

School of Computer Science, National University of Defense Technology, Changsha, 410073,
Hunan, China
liuxingwang023@163.com,
Guomin_Zhang@163.com, Zhanyubin_dm@yahoo.com.cn, nudt_EN@263.net

Abstract. Incremental learning has been widely addressed in machine learning literature to deal with tasks where the learning environment is steadily changing or training samples become available one after another over time. Support Vector Machine has been successfully used in pattern recognition and function estimation. In order to tackle with incremental learning problems with new features, an incremental feature learning algorithm based on Least Square Support Vector Machine is proposed in this paper. In this algorithm, features of newly joined samples contain two parts: already existing features and new features. Using historic structural parameters which are trained from the already existing features, the algorithm only trains the new features with Least Square Support Vector Machine. Experiments show that this algorithm has two outstanding properties. First, different kernel functions can be used for the already existing features and the new features according to the distribution of samples. Consequently, this algorithm is more suitable to deal with classification tasks which can not be well solved by using a single kernel function. Second, the training time and the memory space can be reduced because the algorithm fully uses the structural parameters of classifiers trained formerly and only trains the new features with Least Square Support Vector Machine**.** Some UCI datasets are used to demonstrate the less training time and comparable or better performance of this algorithm than the Least Square Support Vector Machine.

Keywords: Support Vector Machine, Least Square Support Vector Machine, Incremental Learning.

1 Introduction

Example-based learning is an attractive framework for extracting knowledge from empirical data, with the goal of generalizing well on new input patterns. Many real-world processes may be solved by using example-based learning methods. When designing classifiers with learning methods, although more training samples can reduce the prediction error, the learning process can itself get computationally intractable. This issue is becoming more evident today, because there are many complex classification problems in real-world domains, such as medical diagnosis, needed to be solved.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 330–338, 2008.

Ideally, it is desirable to be able to consider all samples at one time, to get the best possible estimate of class distribution. However, in many cases, the data for training can not be gotten simultaneously. One approach to overcome this constraint is to train the classifier using an incremental learning technique, whereby only subsets of the data are to be considered at any one time and results are subsequently combined. Support Vector Machine has shown good results when used for batch learning.

Support Vector Machines (SVM's) [1, 2] proposed by Vapnik is a new learning technique based on the Statistical Learning Theory. Due to the solid theory foundation and good generalization capability, it has drawn much attention on this topic in recent years. The quality and complexity of the SVM solution does not directly depend on the dimensionality of the input space. After nonlinearly mapping the input space into a higher dimensional space, called feature space, the SVM constructs an optimal separating hyperplane in this feature space. The explicit construction of this mapping is avoided by the application of Mercer's condition. Kernels that satisfy Mercer's condition and are commonly used in SVM's are linear, polynomial, radial basis function and multilayer perceptron with one hidden layer [4]. The training of SVM's is done by quadratic programming.

In the SVM incremental feature learning algorithm, we make use of a least square version of Support Vector Machine. Least Square Support Machine (LSSVM) is a SVM version which involves equality instead of inequality constraints and works with a sum squared error (SSE) cost function as it is frequently used in training of classical neural networks [6]. In this way the solution follows from a linear Karush-Kuhn-Tucher condition instead of a quadratic programming problem.

The common disadvantage of traditional incremental learning methods based on SVM [7, 8] is that they can not tackle with newly joined samples with new features. In many real-world applications, however, the training data used for classification are available from several sensors and may contain many other new features. To overcome the disadvantage, an incremental feature learning algorithm based on Least Square Support Vector Machine is proposed. In this algorithm, features of newly joined training samples contain two parts: already existing features and new features. Using historic structural parameters which are trained from the already existing features, the algorithm only trains the new features with Least Square Support Vector Machine.

This paper is organized as follows: in the following section we give an introduction to LSSVM; we introduce the incremental feature learning algorithm based on Least Square Support Vector Machine in Section3; in Section4, we show and analyze the results of experiments; Section5 is the conclusion.

2 Least Square Support Vector Machine (LSSVM)

Given a training set of L points $\{(x_k, y_k)\}_{k=1}^{L}$ with input data $x_k \in \mathbb{R}^M$ and output data $y_k \in \{+1, -1\}$, one considers the following optimization problem:

$$\min J(w,b,e) = \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{i=1}^{L} e_k{}^2 \tag{1}$$

$$s.t. \quad y_k[w^T\varphi_1(x_k)+b]=1-e_k, k=1,\cdots,L \tag{2}$$

where $\varphi_1(.):\mathbb{R}^N \to \mathbb{R}^F$ is a function with which the input space is mapped into a so-called higher dimensional space.

In order to find the optimal hyperplane, one defines the following Lagrangian:

$$L(w,b,e,\alpha)=J(w,b,e)-\sum_{k=1}^{L}\alpha_k\{y_k[w^T\varphi_1(x_k)+b]-1+e_k\} \tag{3}$$

where α_k is a Lagrange multiplier. The condition for optimality

$$\begin{cases} \dfrac{\partial L}{\partial w}=0 \to w=\sum_{k=1}^{L}\alpha_k y_k\varphi_1(x_k) \\ \dfrac{\partial L}{\partial b}=0 \to \sum_{k=1}^{L}\alpha_k y_k=0 \\ \dfrac{\partial L}{\partial e_k}=0 \to \alpha_k=\gamma e_k, k=1,2,\cdots,L \\ \dfrac{\partial L}{\partial \alpha_k}=0 \to y_k[w^T\varphi_1(x_k)+b]-1+e_k=0 \end{cases} \tag{4}$$

can be written as the solution to the following set of linear equations after the elimination of w and e_k,

$$\begin{bmatrix} 0 & Y^T \\ Y & \Omega+I/\gamma \end{bmatrix}\begin{bmatrix} b \\ \alpha \end{bmatrix}=\begin{bmatrix} 0 \\ \vec{e} \end{bmatrix} \tag{5}$$

where $Y=[y_1,y_2,\cdots,y_L]^T, \vec{e}=[1,1,\cdots,1]^T, \alpha=[\alpha_1,\alpha_2,\cdots,\alpha_L]^T$, **I** is a unit diagonal matrix and $\Omega_{ij}=y_i y_j\varphi_1(x_i)^T\varphi_1(x_j)=y_i y_j K_1(x_i,x_j), 1\le i,j\le L$, and K_1 is a kernel function.

After solving the set of linear equation (5), for a given test sample $x\in\mathbb{R}^M$, the corresponding label y is calculated by equation (6).

$$y(x)=sign(\sum_{k=1}^{L}\alpha_k y_k K_1(x_k,x)+b) \tag{6}$$

3 Incremental Feature Learning Algorithm Based on LSSVM

As the tasks of detection, recognition and decision get further, newly joined training data in classification may contain many other new features. However, traditional incremental learning based on SVM can not tackle with training data with new features. Hereby we propose Incremental Feature Learning Algorithm based on Least Square Support Vector Machine to get over this limitation. The main idea of this algorithm is using two local minima to approximate the global minimum on some constrains. It can learn new structural parameters from new features when keeping historic structural parameters learned from previous data available so that the training time is reduced while classification precision is kept.

Suppose the former training data contains M-dimension features, denoted as $x \in \mathbb{R}^M$, newly joined training data contains (M+N)-dimension features with N new features, and suppose the structural parameters of classifier trained from newly joined training data with (M+N)-dimension features are w and b. In order to make use of the historical structural parameters w_M and b_M, w can be approximately denoted as $w = [w_M, w_N]$ with w_M representing the structural parameter which are trained from previous M-dimension features and w_N representing the structural parameter which are trained from the rest of new N-dimension features.

Given a set of data points $\{(x_k, y_k)\}_{k=1}^{L'}$ with input data $x_k \in \mathbb{R}^{M+N}$ and output data $y_k \in \{+1, -1\}$, x_k can be denoted as $x_k = [x_{kM}, x_{kN}]$ with x_{kM} representing the previous M-dimension features of x_k and x_{kN} representing the rest new N-dimension features. Applying two different implicit functions to the already existing M-dimension and the rest new N-dimension features respectively, we reformulate the Least Square Support Vector Machine to resolve the incremental features learning problem as follows:

$$\min J(w_N, b, e) = \frac{1}{2}\begin{pmatrix} w_M \\ w_N \end{pmatrix}^T \begin{pmatrix} w_M \\ w_N \end{pmatrix} + \gamma \sum_{i=1}^{L'} e_k{}^2 \tag{7}$$

$$s.t. \quad y_k \left(\begin{pmatrix} w_M \\ w_N \end{pmatrix}^T \begin{pmatrix} \varphi_1(x_{kM}) \\ \varphi_2(x_{kN}) \end{pmatrix} + b \right) = 1 - e_k, k = 1, 2, \cdots L' \tag{8}$$

where $\varphi_1(.): \mathbb{R}^M \to \mathbb{R}^F, \varphi_2(.): \mathbb{R}^N \to \mathbb{R}^{F'}$ are functions with which the input space is mapped into a so-called higher dimensional feature space, respectively.

In order to find the optimal hyperplane, one defines the following Lagrangian:

$$L(w_N, b, e, \alpha'') = J(w_N, b, e) - \sum_{k=1}^{L'} \alpha_k'' \{ y_k [\begin{pmatrix} w_M \\ w_N \end{pmatrix}^T \begin{pmatrix} \varphi_1(x_{kM}) \\ \varphi_2(x_{kN}) \end{pmatrix} + b] - 1 + e_k \} \tag{9}$$

where α_k'' is a Lagrange multiplier. The condition for optimality

$$\begin{cases} \dfrac{\partial L}{\partial w_N} = 0 \to w_N = \sum_{k=1}^{L'} \alpha_k'' y_k \varphi_2(x_{kN}) \\ \dfrac{\partial L}{\partial b} = 0 \to \sum_{k=1}^{L'} \alpha_k'' y_k = 0 \\ \dfrac{\partial L}{\partial e_k} = 0 \to \alpha_k'' = \gamma e_k, k = 1, 2, \cdots, L' \\ \dfrac{\partial L}{\partial \alpha_k''} = 0 \to y_k [\begin{pmatrix} w_M \\ w_N \end{pmatrix}^T \begin{pmatrix} \varphi_1(x_{kM}) \\ \varphi_2(x_{kN}) \end{pmatrix} + b] - 1 + e_k = 0 \end{cases} \tag{10}$$

can be written as the solution to the following set of linear equations after the elimination of w_N and e_k,

$$\begin{bmatrix} 0 & Y^T \\ Y & \Omega + I/\gamma \end{bmatrix} \begin{bmatrix} b \\ \alpha'' \end{bmatrix} = \begin{bmatrix} 0 \\ \vec{1} - \beta \end{bmatrix} \tag{11}$$

where $Y = [y_1, y_2, \cdots, y_{L'}]^T, \vec{1} = [1, 1, \cdots, 1]^T, \alpha'' = [\alpha_1'', \alpha_2'', \cdots, \alpha_{L'}'']^T$, $\mathbf{I}$ is a unit diagonal matrix, $\Omega_{ij} = y_i y_j \varphi_2(x_{iN})^T \varphi_2(x_{jN}) = y_i y_j K_2(x_{iN}, x_{jN})$, $\beta_j = y_j w_M{}^T \varphi_1(x_{jM}) = y_j \sum_{k=1}^{L} \alpha_k'' y_k K_1(x_k, x_{jM}), 1 \le i, j \le L'$, K_1 and K_2 are two different kernel functions.

After solving the set of linear equations (11), for a given test sample $x \in \mathbb{R}^{M+N}$ denoted as $[x_M, x_N]$, the corresponding label y is calculated by equation (12).

$$\begin{aligned} y(x) &= sign(w_M{}^T \varphi_1(x_M) + w_N{}^T \varphi_2(x_N) + b) \\ &= sign(\sum_{i=1}^{L} \alpha_i y_i K_1(x_i, x_M) + \sum_{k=1}^{L'} \alpha_k'' y_k K_2(x_{kN}, x_N) + b) \end{aligned} \tag{12}$$

4 Experiments

The performance of the proposed method is measured on four UCI datasets which are summarized in Table 1.

Table 1. The datasets used in the experiments

Dataset	# Examples	#features	# training data	#testing data
Iris	150	4	135	15
Wine	178	13	161	17
Spambase	4601	57	4201	400
Waveform	5000	40	4500	500

The number of already existing features, the number of new features, the kernel functions used in already existing features and new features in our experiments are presented in table 2.

Table 2. The number of already existing, new features and kernel functions

Dataset	#existing features	#new features	kernel function used in already existing features	kernel function used in new features
Iris	2	2	linear	linear
Wine	8	5	linear	linear
Spambase	40	1:1:17	Gaussian	linear
Waveform	25	1:1:15	Gaussian	linear

The performance of this algorithm is measured by means of classification precision and training time. The results presented in the following tables and figures are got by the 10-fold cross validation using above measures.

Table 3 and table 4 list the comparison in classification precision and training time between Incremental Feature Learning Algorithm based on LSSVM and LSSVM, respectively, with 2 additional features on Iris while 5 on Wine.

Table 3. The Comparison of Incremental Feature Learning Algorithm based on LSSVM and LSSVM in classification precision

Dataset	Incremental Feature Learning Algorithm based on LSSVM	LSSVM	
		Training with the precious M-dimension features	Training with (M+N)-dimension features
Iris	0.99	1	1
Wine	0.94	0.96	0.98

Table 4. The Comparison of Incremental Feature Learning Algorithm based on LSSVM and LSSVM in training time (sec.)

Dataset	Incremental Feature Learning Algorithm based on LSSVM	LSSVM(Training with (M+N)-dimension features)
Iris	0.0093	0.0167
Wine	0.0085	0.0181

The comparisons between Incremental Feature Learning Algorithm based on LSSVM and LSSVM in classification precision as the number of the new features increasing are shown in Figure 1 and Figure 2 on Spambase and Waveform, respectively. Figure 3 and Figure 4 demonstrate the comparison of the training time as the number of the new additional features increasing on Spambase and Waveform, respectively. It is got by the 10-fold cross validation using above measures.

In table 3, for Iris and Wine datasets, the incremental feature learning algorithm based on LSSVM is inferior to LSSVM in classification precision; for the later two datasets, our algorithm is superior. We can see that our algorithm achieves comparable or better performance than LSSVM when the training data which can not be separated in features space after mapping using an implicit function. It is clear that this method is more suitable to tackle with classification tasks which can not be well solved using a single kernel function. When dealing with classification task with a large amount of new additional features, our experiments validate that the proposed algorithm have higher classification precision while saving training time.

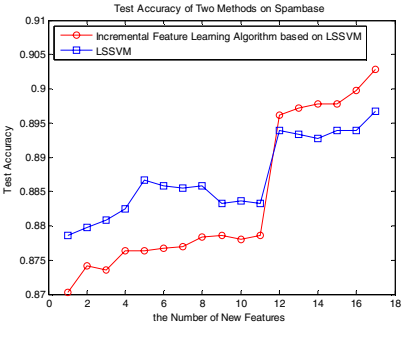


Fig. 1. Comparison of Classification Accuracy between the Incremental Feature Learning Algorithm based on LSSVM and LSSVM on Spamebase as the number of the new additional features increasing

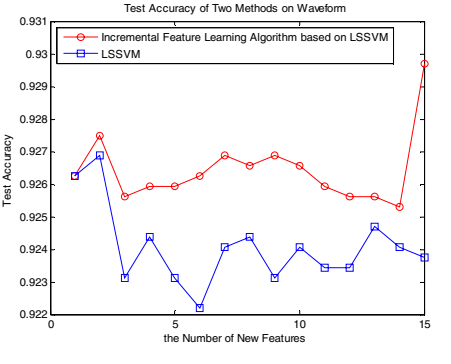


Fig. 2. Comparison of Classification Accuracy between the Incremental Feature Learning Algorithm based on LSSVM and LSSVM on Waveform as the number of the new additional features increasing

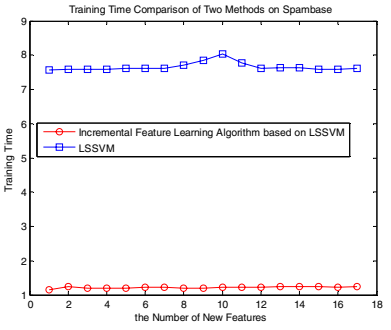


Fig. 3. Comparison of Training Time between the Incremental Feature Learning Algorithm based on LSSVM and LSSVM on Spamebase as the number of the new features increasing

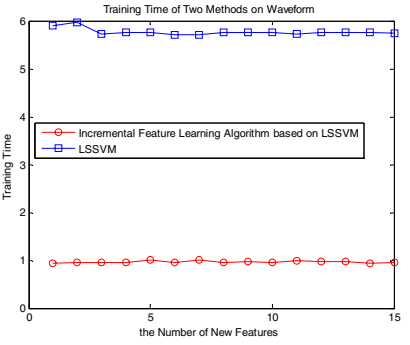


Fig. 4. Comparison of Training Time between the Incremental Feature Learning Algorithm based on LSSVM and LSSVM on Waveform as the number of the new features increasing

5 Conclusions and Future Work

We propose a novel Incremental Features Learning Algorithm based on LS-SVM. In this algorithm, we make use of the historical structural parameters and apply different kernel functions to the already existing M-dimension features and the new additional N-dimension feature. We conduct experiments to evaluate the performance of the proposed Incremental Feature Learning Algorithm based on LSSVM on some UCI datasets. In general, this algorithm achieves better classification precision and can reduce the training time and memory space.

How to select a suitable kernel function and its parameters for a given dataset in incremental feature learning algorithm is our future work.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (NO.60603015).

References

1. Vapnik, V.: The nature of statistical learning theory. Springer, New York (1995)
2. Vapnik, V.: Statistical learning theory. John Wiley, New York (1998)
3. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines. Cambridge University Press, Cambridge (2000)
4. Schökopf, B., Burges, C., Smola, A. (eds.): Advances in Kernel Methods-Support Vector Learning. MIT press, Cambridge (1999)
5. Schökopf, B., Mika, S., Burge, C., Knirsch, P., Müller, K.-R., Rätsch, G., Smola, A.: Input Space vs. Feature Space in Kernel-Based Methods. IEEE Transactions on Neural Networks 10(5), 1000–1017 (1999)

6. Suykens, J.A.K., Vandewalle, J.: Least squares support vector machine classifiers. Neural Processing Letters 9(3), 293–300 (1999)
7. Domeniconi, C., Gunopulos, D.: Incremental support vector machine construction. In: Proceedings IEEE International Conference on Data Mining, 2001, ICDM 2001, pp. 589–592 (2001)
8. Alistair Shilton, M.: Incremental Training of Support Vector Machines. IEEE transactions on neural networks 16(1), 114–131 (2005)
9. Ong, C.S., Smola, A.J., Williamson, R.C.: Learning the Kernel with Hyperkernels. Journal of Machine Learning Research 6, 1043–1071 (2005)
10. Zhao, Y., He, Q.: An Incremental Learning Algorithm Based on Support Vector Domain Classifier. In: Proc. 5th IEEE Int. Conf. on Cognitive Information, pp. 805–809 (2006)
11. Shilton, A., Palaniswami, M., Ralph, D., Tsoi, A.: Incremental training of support vector machines. IEEE Trans. Neural Netw. 16, 114–131 (2005)
12. Pelckmans, K., Karsmakers, P., Suykens, J.A.K., De Moor, B.: Ordinal Least Squares Support Vector Machines – a Discriminant Analysis Approach. In: Proc. of the Machine Learning for Signal Processing (MLSP 2006), Maynooth, Ireland, September 2006, pp. 1–8 (2006)
13. Kazushi, I., Takemasa, Y.: Incremental support vector machines and their geometrical analyses. In: Neurocomputing, pp. 2528–2533 (2007)

A Novel Wavelet Image Fusion Algorithm Based on Chaotic Neural Network

Hong Zhang[1,2], Yan Cao[2], Yan-feng Sun[2], and Lei Liu[1,*]

[1] College of Computer Science and Technology, Jilin University,
2699 Qianjin Street, 130012, Changchun, China
hong168z@126.com
[2] State Key Laboratory on Integrated Optoelectronics, College of Electronic Science and Engineering, Jilin University, 2699 Qianjin Street, 130012, Changchun, China
zhong@jlu.edu.cn

Abstract. In this paper, Transiently Chaotic Neural Network (TCNN) is used in wavelet image fusion method. This paper adopts the weighted average strategy for the fusion of the wavelet transform coefficients. The TCNN outputs the weighting coefficient of every wavelet transform pixel when the energy function of the neural network has achieved the global minimum. At the same time, the average gradient value of the region around every wavelet transform pixel gets the global maximum according to the relationship between the average gradient and energy. The wavelet transform coefficients of the fused image are got by using the weighting coefficients. The advantage of the algorithm is that the weighting coefficient is obtained through the dynamic searching optimization of the average gradient. Experiments show that the average gradient values of the fusion images using the proposed method are greater than the results using the region energy method. The TCNN method improves the performance of the fusion image effectively.

1 Introduction

Image fusion is to integrate complementary information from the same view point under different focal sensors. Due to the limited depth of focus of optical lenses in image sensors, it is often not possible to get an image that contains all relevant objects in focus. In an image obtained by different sensors, only those objects within the depth of field are focus, while other objects are blurred. Image fusion process is required to give the new image which is more suitable for the purpose of human visual perception and computer-processing tasks such as segmentation, feature extraction, and object recognition. As a promising research field in recent years [1-2], it has been a hot spot in the research area of object detection, automatic target recognition, remote sensing, computer vision, smart buildings, robotics, battlefield surveillance, guidance and control of autonomous vehicles, monitoring of complex machinery, meteorological imaging and military applications.

* Corresponding author.

F.P. Preparata, X. Wu, and J. Yin (Eds.): FAW 2008, LNCS 5059, pp. 339–348, 2008.

In recent years, various image fusion methods have been explored is by using wavelet decomposition [3-6]. Among the fusion methods of the wavelet coefficients, the weighted average decision method is typically used, in which choose-max decision method is a special case of the weighted average decision method. Examples of this approach include the Gabor filter method [7], local region variance method [8], window-based standard deviation method [9] and region energy method [10]. Among these methods, the region energy method is supposed to achieve the better result. In these methods the subjective factors and rigid formulas are used to calculating the weighting coefficients and the fusion results are not good as expected.

This paper proposes a novel wavelet TCNN fusion algorithm. Firstly, discrete wavelet transform is used and the source images are decomposed into a series of frequency channels. The weighted average strategy is taken up for the fusion of the wavelet transform coefficients. The weighting coefficient of every wavelet pixel is obtained through the dynamic optimization of the chaotic neural network when the energy function of the chaotic neural network comes to the minimum. At this time, according to the relationship between the average gradient and energy function, the value of the region average gradient around the wavelet pixel comes to be the global maximum. The region average gradient can be calculated according to the weighting coefficient of the wavelet pixel. After t times of iteration of the neural network, the weighting coefficient will be outputted by the chaotic neural network. At last, the fused image is obtained through the inverse wavelet transform. Experimental results show that the TCNN method has the greater value of the average gradient compared with the results got by the region energy method. The following sections of this paper are organized as follows. Section 2 presents the wavelet and TCNN theory. The propose algorithm is described in Section 3. Experimental results compared with region energy method will be presented in Section 4. Finally Section 5 summarizes the conclusions.

2 The Theory of TCNN Algorithm

In this section, the theories of wavelet decomposition and chaotic neural network are introduced.

2.1 The 2-D Wavelet Decomposition

With the development of mathematics and its applications in technology, researchers have explored a large number of methods for multi-resolution analysis, such as the well-known Laplacian pyramid introduced by Burt and Adelson [11], the morphological pyramid [12-13], or the ratio-of-low-pass pyramid [14]. In recent years, because of its ability to analyze the signal in both the spatial and frequency fields, DWT is a most powerful tool in multi-resolution analysis.

The most convenient representation of an image requires that the image can be recovered without any information lost. This process is always viewed as perfect reconstruction condition and ψ is defined as the scale function and ω is defined as the wavelets function. Thus the perfect reconstruction can be represented as:

$$\Psi^{\downarrow}\left(\psi^{\uparrow}(x),\omega^{\uparrow}(x)\right)=x,\quad \text{for } x\in V_0 \tag{1}$$

$$\psi^{\uparrow}\left(\Psi^{\downarrow}(x,y)\right)=x \quad and \quad \omega^{\uparrow}\left(\Psi^{\downarrow}(x,y)\right)=y, \qquad for \quad x\in V_1, y\in W_1 \tag{2}$$

This paper will give a brief scheme of 2-D wavelet analysis and more detail information about the wavelet theory. Fig. 1 shows a two-level decomposition process of DWT, and it can be easily extended to any n-level transform. L symbolizes the 1-D low-pass filter and H symbolizes as the 1-D high-pass filter. Four sub images, $I_{LL}(x,y)$, $I_{LH}(x,y)$, $I_{HL}(x,y)$ and $I_{HH}(x,y)$, can be obtained by applying DWT to both rows and columns. In this framework $I_{LL}(x,y)$ is a smoothed version of the original image I(x,y). $I_{LH}(x,y)$, $I_{HL}(x,y)$ and $I_{HH}(x,y)$ are sub images representing the detail information in horizontal, vertical and diagonal directions of the input image.

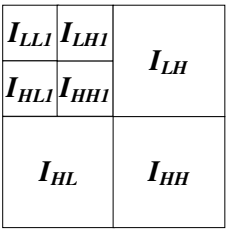


Fig. 1. A representation of 2-D wavelet decomposition

2.2 The Chaotic Neural Network

Neural network is a very complicate nonlinear system, and it contains all kinds of dynamic behaviors. Some chaotic behaviors have been observed in human brains and animals' neural systems, so they would improve the intelligent ability in neural network. Further more, artificial neural network would have much more use in application if chaotic dynamics mechanism is introduced. Chaotic neural networks [15-17] have been proved to be powerful tools for escaping from local minimum. Chaotic neural networks with chaotic dynamics have much rich and far-from equilibrium dynamics with various coexisting attractors, not only of fixed and periodic points but also of strange attractors. Chaotic neural networks can be applied to dynamically associative memory, chaotic forecasting and chaotic optimization. The characteristic of chaotic optimization is used in this paper.

Aihara K. proposed chaotic neural network (CNN) in [18]. A negative self-feedback term is added in Hopfield neural network. This term introduces chaotic dynamics to the network and can help network escape from the attraction of local minimum points effectively. Chen L. Proposed Transiently Chaotic Neural Network in [19], when the self-feedback term is large enough, TCNN utilize chaotic dynamics to escape from local minimum points, when it becomes small enough, the network fades away to Hopfield neural network and promises convergence to a near-optimal solution. The structure of chaotic neural network is showed in fig. 2.

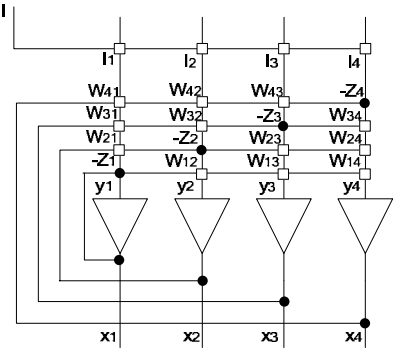


Fig. 2. Structure of chaotic neural network

where x_i is output of neuron i ; y_i denotes internal state of neuron i ; Z_i is self-feedback connection weight; I_i is input bias of neuron i ; W_{ij} describes connection weight from neuron j to neuron i .

3 TCNN Algorithm

In this section, the frame and the process of the algorithm are given.

3.1 The Frame of the Algorithm

The frame of TCNN algorithm is denoted in fig. 3. First, the source images are decomposed into multi-resolution representation. Then the weighted average decision met- hod is taken to integrate all these decompositions to produce a composite representati-on. The weighing coefficients will be outputted by the chaotic neural network until the region average gradient value achieves the maximum. The final step is to do Inve- rse Discrete Wavelet Transform (IDWT), and the final fusion result is obtained.

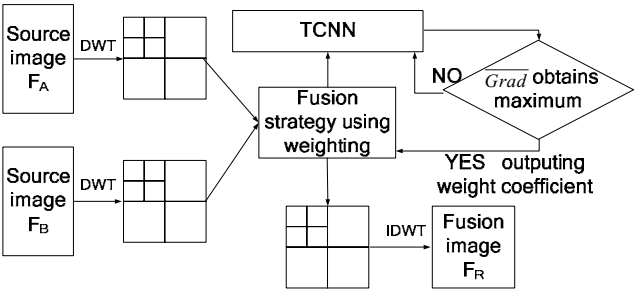


Fig. 3. The frame of TCNN algorithm

The weighting fusion strategy is presented as follows:

$$f_R(i,j) = \omega_A \times f_A(i,j) + \omega_B \times f_B(i,j) \tag{3}$$

$$\omega_B = 1 - \omega_A \tag{4}$$

where f_A, f_B and f_R are respectively the decomposition coefficients of wavelet of the source image F_A, F_B and fused image F_R; ω_A and ω_B are respectively the weighting coefficients of wavelet of the source image F_A and F_B.

3.2 The Average Gradient Quality Assessment

The average gradient [20] is used to measure the clarity of the image and it can reflect the contrast between the tiny details. The average gradient is presented as follows:

$$\overline{Grad} = \frac{1}{m \times n} \sum_{x=1}^{m} \sum_{y=1}^{n} \sqrt{\frac{f_x^2(i,j) + f_y^2(i,j)}{2}} \tag{5}$$

Where $m \times n$ is the size of the region around the wavelet pixel (i,j); The average gradient is calculated in the region. $f_x(i,j)$ and $f_y(i,j)$ are the differences of the x axes direct and the y axes direct. The greater the value of $\overline{Grad}$, the more information the image can be obtained. For this reason, the image can have higher clarity.

3.3 The Chaotic Neural Network and the Setting

W. Zhong proposed a model of the transiently chaotic neural network in [21] and it is defined as below:

$$x_i(t) = \frac{1}{1 + \exp\left(-y_i(t) / \varepsilon\right)} \tag{6}$$

$$y_i(t+1) = k y_i(t) + \left[1 - \alpha(t)\right]\left(-\frac{\partial E}{\partial x_i}\right) - z_i(t)\left[x_i(t) - I_0\right] \tag{7}$$

$$z_i(t+1) = (1-\beta) z_i(t) \tag{8}$$

$$\alpha(t+1) = \gamma \alpha(t) \tag{9}$$

where i is the index of neurons and n is the number of neurons ($1 \le i \le n$); $x_i(t)$ is output of the i th neuron at the discrete time t, $t \in$ N (positive integers); ε is steepness parameter of the activation function ($\varepsilon > 0$); y_i denotes internal state of neuron i; k is the damping factor of the nerve membrane ($0 \le k \le 1$); I_0 is the positive

parameter; $z_i(t)$ is self-feedback connection weight or refractory strength ($z_i(t)>0$); α is the positive scaling parameter for neural inputs; E is energy function. β is damping factor of the time-dependent $z_i(t)$; γ is damping factor of α.

The negative self-feedback is the main factor of the chaotic phenomenon in chaotic neural network. The objective function is often mapped into the energy function and then the minimum of the objective function can be solved by the chaotic neural network. In this method the output $x_i(t)$ and the energy function E are represented as follows:

$$x_1(t) = \omega_A(t) \tag{10}$$

$$E = -\overline{Grad} \tag{11}$$

Neuron output $x_i(t)$ represents the output of weighting coefficient ω_A at the discrete time t. The negative value of the average gradient is presented as the energy function. Because the average gradient is increasing function, the average gradient can obtain global maximum when the energy function has fallen down to the global minimum. At last, the output $\omega_A(t)$ is the optimal weighting coefficient that can be used to obtain the greater value of the average gradient.

3.4 The Process of the TCNN Algorithm

The proposed fusion method consists of the following steps:

Step 1. Initialize parameters of the chaotic neural network is set as follows: k=0.95; ε=0.25; I_0=0.5; β=0.0275; γ=0.9; $\alpha(0)$=0.985 and $z_i(0)$=3.5. If β is smaller, the searching process is longer. If $\alpha(0)$ is too small, the neural network can not produce the chaotic dynamics. If $z_i(0)$ is smaller, the inverse-bifurcation process is shorter, and the chaotic neural network converges sooner to a stable state. For these reasons mentioned above, the initial parameters must be chosen the suitable values. The size of region $m \times n$ for the calculation of the average gradient is set to 3×3.

Step 2. The source images are decomposed into multi-resolution representation and each level of the wavelet coefficients are obtained.

Step 3. (x, y) is the fused image pixel that need to be calculated. The negative value of the average gradient of the region around the (x, y) is represented the energy function. After t times of iteration of the neural network, $\omega_A(t)$ of the pixel (x, y) is outputted by the chaotic neural network. According the formula 3, the wavelet coefficient of pixel (x, y) is calculated by the weighted average strategy.

Step 4. Repeat the step 3, until the pixels of the fused image are all traveled.

Step 5. The fused image is obtained by the inverse wavelet transform.

4 Fusion Experiment

To illustrate chaotic dynamic behaviors of the chaotic neural units, the time evolution figure of a single neuron that is $\omega_A(t)$ and energy function obtained by the experiment are represented as follows:

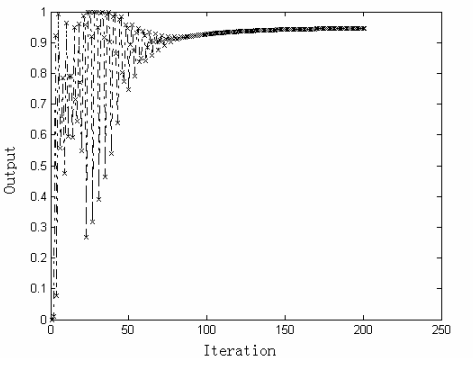


Fig. 4. The time evolution figure of a single neuron

The time evolution figure of a single neuron model which is one of the wavelet weighting coefficients is plotted in fig. 4. TCNN maintains chaotic dynamics at the beginning, converges sooner to a stable state after about 75 steps, and reach a saturated state by using about 170 steps. The output value was about 0.95, so the last value of the wavelet weighting coefficient is 0.95.

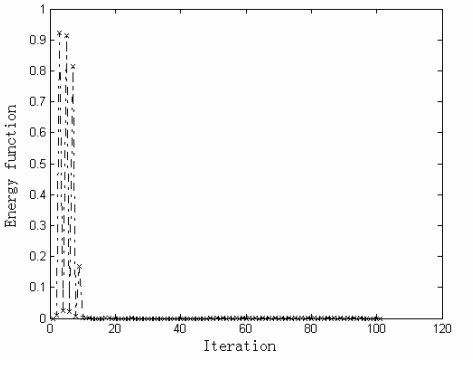


Fig. 5. The time evolution figure of the energy function

According to formulas 6-9, the time evolution figure of the energy function is plotted in fig. 5. The energy function can converge into the minimum after the falling down process.

(a) (b) (c) (d)

Fig. 6. TANK images and fused images.(a) Image1 (focus on the around).(b) Image2 (focus on the center).(c) Fused image with region energy method.(d)Fused image with the TCNN method.

(a) (b) (c) (d)

Fig. 7. CLOCK images and fused images.(a) Image1(focus on the left).(b) Image2(focus on the right).(c) Fused image with region energy method.(d) Fused image with the TCNN method.

Two pairs of the source images and the fused images using the region energy method and TCNN method are shown in fig.6 and fig.7. The region energy method is used to compare with TCNN method. As shown in figure, the images obtained by TCNN method（ fig. 6 (d) and fig. 7(d) ） outperform the images obtained by region energy method (fig 6(c) and fig. 7(c)).

Table 1. $\overline{Grad}$ of two groups of fused images

image	method	$\overline{Grad}$
TANK images	Region energy method	8.4754
	The TCNN method	9.3803
CLOCK images	Region energy method	6.4080
	The TCNN method	7.5723

Table 1 shows the average gradient values of the two sets of multi-focus images (fig. 6 (c,d) and fig. 7 (c,d)). Compared with the region energy method, the values of the average gradient of the fusion images obtained by the TCNN method are greater. It indicates that the fused images obtained by the TCNN method have higher clarity and more abundance information of the two source images.

5 Conclusion

A wavelet image fusion algorithm using the TCNN is presented in this paper. The average gradient value of fused image is used to measure the clarity of image blocks. The weighting coefficient is outputted by chaotic neural network when the average gradient achieved the maximum. The rigid calculation of the weighting coefficient is replaced by dynamic chaotic optimization of TCNN. Experimental results show that the TCNN method by using average gradient as criterion outperforms the region energy method.

References

1. Varshney, P.K.: Scanning the special issue on data fusion. Proc. IEEE 85, 3–5 (1997)
2. Phol, C.: Multisensor Image Fusion in Remote Sensing: Concepts, Methods and Application. International Journal of Remote sensing 9(5), 823–854 (1998)
3. Chipman, L.J., Orr, Y.M., Graham, L.N.: Wavelets and image fusion. In: Proc. Internat. Conf. on Image Processing, Washington, USA, pp. 248–251 (1995)
4. Koren, I., Laine, A., Taylor, F.: Image fusion using steerable dyadic wavelet. In: Proc. Internat. Conf. on Image Processing, Washington, USA, pp. 232–235 (1995)
5. Li, H., Manjunath, B.S., Mitra, S.K.: Multisensor image fusion using the wavelet transform. Graph. Models Image Process. 57(3), 235–245 (1995)
6. Yocky, D.A.: Image merging and data fusion by means of the discrete two-dimensional wavelet transform. J. Opt. Soc. Am. A: Opt., Image Sci. Vision 12(9), 1834–1841 (1995)
7. Li-ming, G., Hong-lin, C.: An image fusion method based on wavelet transformation. Computer Simulation 24(3), 194–197 (2007)
8. Guan-qun, T., Guang-hua, L.: Medical Image Fusion Based on Wavelet Transforms and 3D Reconstruction Based on a Searching Algorithm of Volume Data. Journal of Zhejiang Wanli University 16(4), 58–62 (2003)
9. Shu-gang, C., Xue-jie, Z.: Fusing anatomical and functional medical images based on wavelet coefficient s adaptive weighted averaging. Journal of Yunnan University (Natural Sciences) 27(3), 200–205 (2005)
10. Kai, H., Mingyi, H., Xiaorong, W., Tao, G.: A New Medical Image Fusion Method Based on Energy Weighting Algorithm in Wavelet Domain. Science Technology and Engineering 6(13), 1949–1954 (2006)
11. Burt, P.J., Adelson, E.H.: The Laplacian pyramid as a compact image code. IEEE Transactions on Communication 31, 245–253 (1983)
12. Goutsias, J., Heijmans, H.J.A.M.: Nonlinear multiresolution signal decomposition schemes. Part I: Morphological pyramids. IEEE Transactions on Image Processing. 9(11), 1862–1876 (2000)
13. Toet, A.: A morphological pyramidal image decomposition. Pattern Recognition Letters 9, 255–261 (1989)
14. Toet, A.: Image fusion by a ratio of low-pass pyramid. Pattern Recognition 9, 245–253 (1989)
15. Lu, J., Cao, J.: Synchronization-based approach for parameters identification in delayed chaotic neural networks. Physica A 382(2), 672–682 (2007)
16. Xu, X., Tang, Z., Wang, J.: A method to improve the transiently chaotic neural network. Neurocomputing 67, 456–463 (2005)
17. Wang, X., Qiao, Q.: A Quickly Searching Algorithm for Optimization Problems Based on Hysteretic Transiently Chaotic Neural Network. In: Liu, D., Fei, S., Hou, Z., Zhang, H., Sun, C. (eds.) ISNN 2007. LNCS, vol. 4492, pp. 72–78. Springer, Heidelberg (2007)

18. Aihara, K., Takabe, T., Toyoda, M.: Chaotic neural networks. Physical Letters A 144(6), 333–340 (1990)
19. Chen, L.N., Aihara, K.: Chaotic simulated annealing by a neural network model with transient chaos. Neural Networks 8(6), 915–930 (1995)
20. Chunmei, L., Rulin, W., Shuxia, L., Guoxin, L., Jie, L.: Wavelet Image Fused Based on Neighborhood Average Grads. Control & Automation 22(12-3), 306–307 (2006)
21. Zhong, W., Cheng, S.X.: Multiuser detection using time-varying scaling-parameter transiently chaotic neural networks. Electronic Letters 35(12), 987–989 (1999)

Author Index