Thomas Erlebach
Giuseppe Persiano (Eds.)

LNCS 7846

Approximation and Online Algorithms

10th International Workshop, WAOA 2012
Ljubljana, Slovenia, September 2012
Revised Selected Papers

Springer

Lecture Notes in Computer Science 7846

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

Thomas Erlebach Giuseppe Persiano (Eds.)

Approximation and Online Algorithms

10th International Workshop, WAOA 2012
Ljubljana, Slovenia, September 13-14, 2012
Revised Selected Papers

Springer

Volume Editors

Thomas Erlebach
University of Leicester
Department of Computer Science
University Road
Leicester LE1 7RH, UK
E-mail: t.erlebach@leicester.ac.uk

Giuseppe Persiano
Università di Salerno
Dipartimento di Informatica "Renato M. Capocelli"
Via Ponte Don Melillo
84081 Fisciano (SA), Italy
E-mail: giuper@dia.unisa.it

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-38015-0 e-ISBN 978-3-642-38016-7
DOI 10.1007/978-3-642-38016-7
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013936596

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

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The 10th Workshop on Approximation and Online Algorithms (WAOA 2012) focused on the design and analysis of algorithms for online and computationally hard problems. Both kinds of problems have a large number of applications from a variety of fields. WAOA 2012 took place in Ljubljana, Slovenia, during September 13–14, 2012. The workshop was part of the ALGO 2012 event that also hosted ESA, WABI, ALGOSENSORS, IPEC, ATMOS, and MASSIVE.

Topics of interest for WAOA 2012 were: algorithmic game theory, algorithmic trading, coloring and partitioning, competitive analysis, computational advertising, computational finance, cuts and connectivity, geometric problems, graph algorithms, inapproximability results, mechanism design, natural algorithms, network design, packing and covering, paradigms for the design and analysis of approximation and online algorithms, parameterized complexity, real-world applications, and scheduling problems. In response to the call for papers we received 60 submissions, one of which was subsequently withdrawn. Each submission was reviewed by at least three referees. The submissions were mainly judged on originality, technical quality, and relevance to the topics of the conference. Based on the reviews, the Program Committee selected 22 papers. This volume contains final revised versions of these papers as well as an abstract of the invited talk.

We would like to thank all the authors who submitted papers to WAOA 2012, and our plenary invited speaker Nikhil Bansal for accepting our invitation. Furthermore, we are grateful to the local organizers of ALGO 2012: Andrej Brodnik (Co-chair), Uroš Čibej, Gašper Fele-Žorž, Matevž Jekovec, Jurij Mihelič, Borut Robič (Co-chair), and Andrej Tolič.

March 2013

Thomas Erlebach
Giuseppe Persiano

Organization

Program Co-chairs

Thomas Erlebach	University of Leicester, UK
Giuseppe Persiano	Università di Salerno, Italy

Program Committee

Evripidis Bampis	Université Pierre et Marie Curie, Paris, France
Cristina Bazgan	Université Paris Dauphine, France
Wolfgang Bein	University of Nevada, Las Vegas, USA
Marek Chrobak	University of California, Riverside, USA
Andrea Clementi	University of Rome "Tor Vergata", Italy
Thomas Erlebach	University of Leicester, UK
Stanley Fung	University of Leicester, UK
Martin Hoefer	RWTH Aachen, Germany
Klaus Jansen	University of Kiel, Germany
Christos Kaklamanis	University of Patras and CTI, Greece
Nicole Megow	Max Planck Institute for Informatics, Saarbrücken, Germany
Seffi Naor	Technion, Haifa, Israel
Zeev Nutov	The Open University of Israel
Giuseppe Persiano	Università di Salerno, Italy
Kirk Pruhs	University of Pittsburgh, USA
Jiří Sgall	Charles University, Prague, Czech Republic
Roberto Solis-Oba	University of Western Ontario, Canada
Rob van Stee	Max Planck Institute for Informatics, Saarbrücken, Germany
Andreas Wiese	Sapienza University of Rome, Italy
Paul Wollan	Sapienza University of Rome, Italy

Additional Referees

James Andro-Vasko	Lucas Bang	Henning Bruhn
Spyros Angelopoulos	Marcin Bienkowski	Vincent Chau
Antonios Antoniadis	Vittorio Bilò	Shiri Chechik
Vincenzo Auletta	Marin Bougeret	Chandra Chekuri
János Balogh	Joan Boyar	Lin Chen
Evangelos Bampas	Peter Brucker	Shahar Chen

Christine Chung
Nachshon Cohen
Daniel Cole
Basile Couëtoux
Pierluigi Crescenzi
Ovidiu Daescu
Miriam Di Ianni
Yann Disser
Reza Dorrigiv
György Dósa
Zdenek Dvorak
Matthias Englert
Leah Epstein
Lene Favrholdt
Moran Feldman
Diodato Ferraioli
Esteban Feuerstein
Rudolf Fleischer
Dimitris Fotakis
Pierre Fraigniaud
Takuro Fukunaga
Martin Gairing
Heidi Gebauer
Yiannis Giannakopoulos
Fabrizio Grandoni
Alexander Grigoriev
Xin Han
Tony Huynh
Hiro Ito
Takehiro Ito
Gwenaël Joret
Lior Kamma
Frank Kammer
Panagiotis Kanellopoulos
Nikos Karanikolas
Thomas Kesselheim
Kim Manuel Klein
Guy Kortsarz
Stefan Kraft
Felix Kumm
Maria Kyropoulou
Kati Land
Lawrence Larmore
Kim S. Larsen
Erik Jan van Leeuwen
Dimitrios Letsios
Asaf Levin
Fei Li
Alejandro López-Ortiz
Giorgio Lucarelli
Manor Mendel
Ioannis Milis
Debajyoti Mondal
Benjamin Moseley
Kim Thang Nguyen
Serguei Norine
Marcel Ochel
Vinayaka Pandit
Ondrej Pangrac
Vangelis Paschos
Francesco Pasquale
Lars Prädel
Dror Rawitz
Damien Regnault
Daniel Reichman
Marc Renault
Christina Robenek
Gianluca Rossi
Matthias Stallmann
Zoya Svitkina
Eric Torng
Gerhard Trippen
José Verschae
Fabio Vitale
Tjark Vredeveld
Jens Vygen
Yicheng Wen
Matthias Westermann
Li Yan
Jonathan Yaniv
Guochuan Zhang
Georgios Zois

Table of Contents

Invited Contribution

Session 1: Graphs and Networks

Session 2: Geometric Problems

Session 3: Online Algorithms

Session 4: Scheduling

Session 5: Algorithmic Game Theory

Session 6: Approximation Algorithms

The Primal-Dual Approach for Online Algorithms

Nikhil Bansal

Eindhoven University of Technology, Eindhoven, The Netherlands
n.bansal@tue.nl

Abstract. Online algorithms deal with settings where the input data arrives over time and the current decision must be made by the algorithm without the knowledge of future input. In the last few years, the online primal-dual approach, pioneered by Buchbinder and Naor [4], has emerged as a very powerful and general method to systematically design and analyze online algorithms.

In this talk, I will give an overview of the method and show how it unifies and simplifies various previous results. I will also describe the recent successes of this approach in addressing some classic problems such as weighted paging and the randomized k-server problem [2,1]. Finally, we will also see some recent extensions of the method [3,6,5], beyond the original framework of Buchbinder and Naor [4].

Based on joint works with Niv Buchbinder, Aleksander Madry and Joseph (Seffi) Naor.

References

1. Bansal, N., Buchbinder, N., Madry, A., Naor, J.: A polylogarithmic-competitive algorithm for the k-server problem. In: Foundations of Computer Science, FOCS (2011)
2. Bansal, N., Buchbinder, N., Naor, J.: A primal-dual randomized algorithm for weighted paging. In: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, pp. 507–517 (2007)
3. Bansal, N., Buchbinder, N., Naor, J.: Towards the randomized k-server conjecture: A primal-dual approach. In: SODA, pp. 40–55 (2010)
4. Buchbinder, N., Naor, J.: Online primal-dual algorithms for covering and packing problems. In: Proceedings of the 13th Annual European Symposium on Algorithms (2005)
5. Anand, S., Garg, N., Kumar, A.: Resource augmentation for weighted flow-time explained by dual fitting. In: SODA, pp. 1228–1241 (2012)
6. Gupta, A., Krishnaswamy, R., Pruhs, K.: Online Primal-Dual For Non-linear Optimization with Applications to Speed Scaling. In: Erlebach, T., Persiano, G. (eds.) WAOA 2012. LNCS, vol. 7846, pp. 173–186. Springer, Heidelberg (2013)

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, p. 1, 2013.

Independent Set with Advice: The Impact of Graph Knowledge
(Extended Abstract)

Stefan Dobrev[1], Rastislav Královič[2], and Richard Královič[3,4]

[1] Institute of Mathematics, Slovak Academy of Sciences, Bratislava, Slovakia
[2] Department of Computer Science, Comenius University, Bratislava, Slovakia
[3] ETH Zurich, Switzerland
[4] Google Zurich, Switzerland

Abstract. We are interested in online graph problems where the knowledge of the underlying graph G (all arriving vertices are from G) has a profound impact on the size of the advice needed to solve the problem efficiently. On one hand, we show that, for sparse graphs, constant-size advice is sufficient to solve the maximum independent set problem with constant competitive ratio, even with no knowledge of the underlying graph. On the other hand, we show a lower bound of $\Omega(\log(n/a)/\log\log(n/a))$ on the competitive ratio of finding a maximum independent set in bipartite graphs if no knowledge of the underlying graph is available and if the advice is of size a. We complement the lower bounds by providing corresponding upper bounds.

1 Exposition and Motivation

Given a simple undirected graph $G = (V, E)$, a subset $I \subseteq V$ is called *independent* if the subgraph induced by I does not contain any edges, i.e., $\forall u, v \in I : \{u, v\} \notin E$. The problem of finding the independent set of maximum cardinality (MIS) is one of the most studied computational problems on graphs with applications in many areas ranging from computer vision, to coding theory, molecular biology, scheduling, or wireless networking, to name just a few. However, this problem is, in general, computationally hard: in [21] it is proven that, unless $NP = ZPP$, the problem[1] cannot be approximated within a factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$ (i.e., there is no polynomial-time algorithm that would always find an independent set of size at least $opt/n^{1-\varepsilon}$).

We are interested in the *online* version of the problem. In online problems, the input is not known to the algorithm at the beginning, instead, it arrives piecewise. The algorithm must present a partial output to each chunk of the input before reading next chunk, and cannot revise its decision afterwards. More formally, the input $\boldsymbol{x}$ is a sequence of *requests* $\boldsymbol{x} = (x_1, \ldots, x_n)$. The output $\boldsymbol{y}$ is a sequence of *answers* $\boldsymbol{y} = (y_1, \ldots, y_n)$ computed by the algorithm in such a

[1] Actually, the paper proves inapproximability of Maximum-Clique problem, where the aim is to find the subset of vertices with largest cardinality that form a clique. However, the two problems are obviously equivalent w.r.t. approximation.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 2–15, 2013.

way that each y_i is a function of $x_1, \ldots, x_i$ (for randomized algorithms also a function of the random bits used so far). The goal is to maximize or minimize a *cost* function defined over the whole output $\boldsymbol{y}$. A standard way of measuring the quality of online algorithms is to compare the (worst case) performance of the algorithm to the optimal solution of the instance (i.e., with known input). For maximization problems, the definition of *competitive ratio* states that an algorithm is *c-competitive* if for any input the cost of the output produced by the algorithm is at least[2] $1/c \cdot opt$. Note that in online problems the main concern is not the computational complexity, but the inherent loss of performance due to the unknown future. Online computation has received considerable attention over the past decades as a natural way of modeling real-time processing of data. For example, in resource scheduling problems a server has to handle a stream of requests (e.g., phone calls), each of them requiring certain subset of available resources (e.g., lines). The goal of maximizing the throughput of the server leads naturally to an online version of MIS (see e.g., [25] in the context of scheduling intervals). For an exposition to online problems, we refer the reader to [5].

When dealing with the online graph problems (and in particular with the online MIS), there are several ways how the input can be presented to the algorithm. A natural way (which we shall refer to as *unknown graph* model) is to present the graph vertex by vertex: Each time a new vertex v_i arrives, the algorithm learns the edges connecting v_i to already presented vertices $v_i, \ldots, v_{i-1}$. Before the next vertex v_{i+1} is presented, the algorithm must decide whether the vertex v_i will be included in the resulting independent set or not. Obviously, no deterministic algorithm can attain a good competitive ratio: consider an instance where at the beginning, a isolated vertices are presented. If the algorithm does not include any one of them to the independent set, no more vertices arrive. On the other hand, if the algorithm includes some of them to the independent set, $b \gg a$ vertices arrive such that they form a complete bipartite graph with the first a vertices. In [6] authors show that the competitive ratio of online MIS on the class of σ-bounded disk graphs is $\Theta(\min\{n, \sigma^2\})$ where the upper bound is attained by a simple first fit algorithm, and the lower bound holds also for randomized algorithms[3].

The inherent difficulty of the unknown graph model led to the study of various other models. In [2], authors use what we shall refer to as the *known supergraph* model: the graph G presented to the algorithm is a subgraph of some larger graph H that is known in advance to the online algorithm. Thus the situation is as follows: first, the graph $H = (V', E')$ is presented to the algorithm with $V' = v_1, \ldots, v_m$. Each request $x_i \in \{1, \ldots, m\}$ represents a vertex, and the algorithm must decide whether to include vertex v_{x_i} in the independent set or not. The optimal solution is the largest independent set of the subgraph

[2] Some works use a slightly relaxed definition by allowing an additive constant, i.e., the algorithm is c-competitive if there exists a constant α such that the cost of the worst-case output (for randomized algorithms the worst-case expected output) is at least $1/c \cdot opt - \alpha$.

[3] If the disk representation is given, the bound is $\Theta(\min\{n, \log \sigma\})$.

of H induced by the presented vertices. This model was motivated by routing problems in networks, where the topology of the network is usually known to the algorithms but the routing requests are not. It was proven in [2] that, even if the algorithms can use randomization, the competitive ratio is $\Omega(n^\varepsilon)$ for some constant ε. The conference version of the paper, [1], mentions a similar lower bound even in the case when preemption is allowed (i.e., the algorithm may, at a later stage, remove a vertex from the independent set in construction, but it cannot be reinserted). In [7], an $O(\log n)$-competitive algorithm for online MIS in chordal graphs is presented using the known supergraph model.

A number of other modifications of the unknown graph model has been studied. In [20], a model is considered where the algorithm is allowed to maintain multiple independent sets under construction, and each vertex can be assigned to at most $r(n)$ different sets. The largest set is chosen as the final output. The competitive ratio in this model is shown to be $\Theta(n/\log n)$ when $r(n)$ is polynomial in n, and $\Theta(n)$ if $r(n)$ is constant. In a more powerful *inheritance* variation, a lower bound $\Omega(n/\log^3 n)$ is proven for polynomial $r(n)$. The graphs used as lower bounds are split graphs (vertex set can be partitioned into an independent set and a clique), and at the same time interval graphs (subclass of chordal graphs). In yet another model from [13] the algorithm starts from a complete graph on n vertices, and each request removes an edge; the algorithm is allowed to add to the constructed independent set endpoint(s) of the removed edge. Again, in this model, a competitive ratio is bounded from below by $(n-1)/2$ and Δ where Δ is the maximum degree of the resulting graph. Still another model, from [9], considers the requests containing not one vertex, but a subset of vertices (together with all induced edges); the authors give an algorithm with a competitive ratio $O(n\sqrt{t(n)}/\log n)$, where $t(n)$ is the number of requests, and n is the number of vertices. In [19], a variation of the unknown graph model, in which the algorithm is at the beginning presented with a graph isomorphic to the presented graph G, is studied in the context of online coloring problems. The author also addresses the independent set: the algorithm is required to produce an online coloring of the input graph, and the largest color class is taken as the solution. Even with this relaxation, the competitive ratio of deterministic algorithms is proven to be $\Omega(n)$, and $\Theta(n/\log n)$ for randomized algorithms against oblivious adversary.

Several modifications of the unknown graph model share the same high-level idea: to enhance the online algorithm with some a-priori information about the input, either in general by using restricted class of input graphs or per-instance by providing a supergraph or an isomorphic graph. In the context of online and distributed algorithms where a crucial role is played by some missing information (about topology of the communication network or the future input), the notion of *advice* has recently become popular as a general way to treat the additional information. The idea of the approach is to augment the algorithm by some information about the instance, and to study the relation of the amount of the added information to the solution quality. There are no computational

constraints on the added information, only that it is a function of the instance and the algorithm at hand.

In the context of online algorithms, the original model from [11] has been developed in two ways: the model from [12] considers that the algorithm receives, with each request, a b-bit string of advice (a function of the whole input and the algorithm). Hence, the case with $b = 0$ corresponds to the classical online model, and the case with $b = \lceil \log |\mathcal{A}| \rceil$, where $\mathcal{A}$ is the action space of the algorithm, always gives an optimal algorithm (since the particular answer may be encoded in the advice of any request). In the model from [4] (see also [3, 22]), the whole advice is given to the algorithm at the beginning in a form of a binary string. A trivial upper bound, apart from specifying each action in $n\lceil \log |\mathcal{A}| \rceil$ bits, is to encode the whole input using the advice of size that corresponds to the Kolmogorov complexity of the input instance. However, as it is shown in [3, 4, 22], these trivial bounds can be in many cases substantially improved.

Our choice for the second model is mainly due to the fact that it makes it possible to analyze sublinear advice. While for problems where there are at most two possible actions for each request (as is the case of MIS; every vertex is either included in the set or not), one bit of advice per request is already sufficient to achieve optimum, it is interesting to know what can be done with smaller advice. We show, e.g., that in the case of bipartite graphs, $O(\log \log n)$ bits of advice already bring the competitive ratio down from $\Omega(n)$ to $O(\log n)$.

In a related context of distributed algorithms, in [14, 16–18, 23] the advice is a function that assigns a binary string to each node of a communication network. The only information about the network known to each node is its local information, and the advice string. The parameter under consideration is either the sum or the maximum of the lengths of the strings in all nodes, and the main question is how much advice is needed to (efficiently) perform communication tasks such as broadcasting in (radio) networks, graph searching, computing a proper vertex coloring of the network, or computing the spanning tree of the network. In [10, 15], there is a mobile entity (agent) performing some task in the network (e.g., exploration), and the advice is a binary string given to the agent based on the topology.

Our Contribution. In this paper we focus on the online MIS in the unknown graph and known supergraph models. While the first intuition is that the knowledge of the supergraph may be significant additional information[4] , it is easy to see that it does not affect the worst-case performance of deterministic algorithms, even restricted to a class of forests (which are planar and bipartite). Indeed, let the underlying graph be a forest containing m stars with k leaves. First, centers of the stars $c_1, c_2, \ldots$ are presented to the algorithm one by one until it selects some c_i for the first time. Then the leaves of the i-th star are presented, and the input ends. Because the competitive ratio is parameterized by the size of the presented graph G (and not in the size of the supergraph H),

[4] Although without any restrictions on the structure or size of the supergraph, one can always construct a "universal" supergraph that fools any deterministic algorithm; with randomized algorithms, however, the situation is more subtle (see [2]).

it can be easily verified that, even in the milder definition involving the additive constant, the competitive ratio is $\Omega(n)$.

We are interested in the question to what extent the known supergraph helps in reducing the amount of advice needed to obtain some given solution quality. We use the model of advice from [3]: at the beginning, the algorithm can access the result of any function of the actual input that returns a binary string of length $s(n)$. Based on this advice, the algorithm then proceeds as a standard online algorithm. Note that this approach is similar to [19] and [20], however, in our case there are $2^{s(n)}$ partial solutions constructed, and the largest one is chosen as output.

We show that in the class of sparse graphs, the knowledge of the supergraph does not help substantially: even in the unknown graph model, advice of constant size is sufficient to obtain constant competitive ratio on graphs with $O(n)$ edges. The situation is, however, different in the class of bipartite graphs. With the known supergraph, a competitive ratio of 2 can be achieved with one bit just by specifying the bipartition with the majority of presented vertices. On the other hand, we show that in the unknown graph model the competitive ratio of any deterministic algorithm with $s(n)$ bits of advice is at least $\Omega(\log(n/s(n))/\log\log(n/s(n)))$. The bounds are complemented with the corresponding opposite bounds.

2 Sparse Graphs

First let us observe the simple fact that if there is a way to construct a proper vertex coloring (i.e. a coloring of vertices where each edge has the endpoints colored by different colors) in the respective online model, then the $s(n)$ bits of advice can specify the largest of the first $2^{s(n)}$ color classes.

Theorem 1. *Let $\mathcal{G}$ be a class of graphs such that each n-vertex graph $G \in \mathcal{G}$ is online colorable (in the respective model) in such a way that the union of the first $k(n)$ color classes contains at least $\alpha(n)$ vertices. Then there is an online MIS algorithm using $\lceil \log k(n) \rceil$ bits of advice with competitive ratio $n\,k(n)/\alpha(n)$.*

In the known supergraph model the presented graph is a subgraph of a known graph. Hence, if the supergraph is k-colorable, we immediately have results for the advice complexity; e.g., 2 bits of advice are sufficient to reach competitive ratio of 4 if the known supergraph is planar (since planar graphs are 4-colorable). There are numerous results on (online) vertex coloring that translate to the advice bounds in a similar way, see e.g., [8, 24, 26] and references therein.

Although sparse graphs (even trees) are not online colorable with constant number of colors, the first fit (FF) algorithm where a new vertex is assigned the smallest color different from the (assigned) colors of its neighbors, is good enough to apply Theorem 1 on sparse graphs in the unknown graph model.

Lemma 2. *Let G be a n-vertex graph with at most cn edges. The* FF *algorithm produces in the unknown graph model a coloring of G such that the union of color classes $1, 2, \ldots, 2c$ contains at least $n/2$ vertices.*

Proof. Let us call an edge e a *forcing edge* if there is a vertex v such that

- when v arrives, it is assigned color larger than $2c$, and
- e is an edge incident to v, leading to a vertex that has arrived before

Let S be the set of vertices given a color larger then $2c$ when FF is run on a graph with at most cn edges. As the sets of forcing edges of different vertices are disjoint and each vertex from S has at least $2c$ forcing edges, it must hold $2c|S| \leq cn$, hence $|S| \leq n/2$. This means that the number of vertices of color at most $2c$ is at least $n/2$. □

Corollary 3. *Online MIS can be solved in the unknown graph model on graphs with at most cn edges using advice of* $1 + \lceil \log c \rceil$ *bits and achieving competitive ratio* $4c$.

Hence, for c independent of n, with $O(\log c)$ bits of advice, it is possible to reach a competitive ratio of $O(c)$ on graphs with at most cn edges. However, improving the ratio to $o(c)$ requires $\Omega(n)$ bits:

Theorem 4. *Let* A *be an algorithm solving online MIS in the unknown graph model on graphs with at most cn edges, and with* $s(n)$ *bits of advice. Then the worst-case competitive ratio of* A *is at least* $\frac{c}{2c\frac{s(n)}{n} + \frac{2c}{n}(1+2\log c)+1}$.

3 Lower Bound for Bipartite Graphs

Recall that in the known supergraph model, one bit of advice is sufficient to achieve competitive ratio 2 on bipartite graphs. The main result of this section is the following theorem:

Theorem 5. *Consider any subadditive function* $s(n) < n$, *and any algorithm* A *solving online MIS in the unknown graph model on bipartite graphs with advice* $s(n)$. *Then the worst-case competitive ratio of* A *over n-vertex bipartite graphs is* $\Omega\left(\frac{\log(n/s(n))}{\log\log(n/s(n))}\right)$.

Hence, not only it is not possible to achieve a constant competitive ratio using constant advice, but $\Omega(n)$ bits of advice are needed to do so.

To prove the theorem we construct a class of bipartite graphs $\{G^{k,a}\}$, such that the number of vertices of $G^{k,a}$ is $n = a(k+1)2^{k+1}$, and the size of maximum independent set of $G^{k,a}$ is $a(k+1)2^k$. We consider a particular set of input instances $\mathcal{C}^{k,a}$ based on a graph $G^{k,a}$ that differ by the order in which the vertices are presented to the algorithm.

Consider an arbitrary algorithm A with at most a bits of advice, and competitive ratio r. Since the advice partitions the set of input instances into 2^a classes such that within each class the instances are processed deterministically, it follows that there is a deterministic algorithm $\mathtt{A}_{det}$, and a subset of $C \subseteq \mathcal{C}^{k,a}$ of size $|C| \geq |\mathcal{C}^{k,a}|/2^a$ such that $\mathtt{A}_{det}$ achieves competitive ratio r over instances from C. The main idea of the proof is to consider an arbitrary deterministic

algorithm $\mathtt{A}_{det}$, and any subset C of size at least $|\mathcal{C}^{k,a}|/2^a$, and to show that $\mathtt{A}_{det}$ constructs an independent set on size $O(a2^k \log k)$ on some instance from C.

The theorem the follows by chosing k to be the largest integer such that $\phi(k) \leq \sqrt{\frac{n}{s(n)+1}}$, where $\phi(x) := (x+1)2^{x+1}$, and letting $a = \left\lceil \frac{n}{\phi(k)} \right\rceil$. The graph $G^{k,a}$ then has $a\phi(k) \geq n$ vertices, and the competitive ratio is $\Omega\left(\frac{\log(\phi(k))}{\log\log(\phi(k))}\right)$. Due to space constraints, we omit the computations.

The lower bound graph $G^{k,a}$ (we shall omit the superscripts from now on) is a bipartite graph with partitions L and R, consisting of a copies of graph G_ϵ (ϵ denoting the empty string) $G_\epsilon(1), G_\epsilon(2), \ldots G_\epsilon(a)$. Each G_ϵ is constructed hierarchically from components G_α for $\alpha \in \{0,1\}^{*k}$ as follows (see Fig. 1):

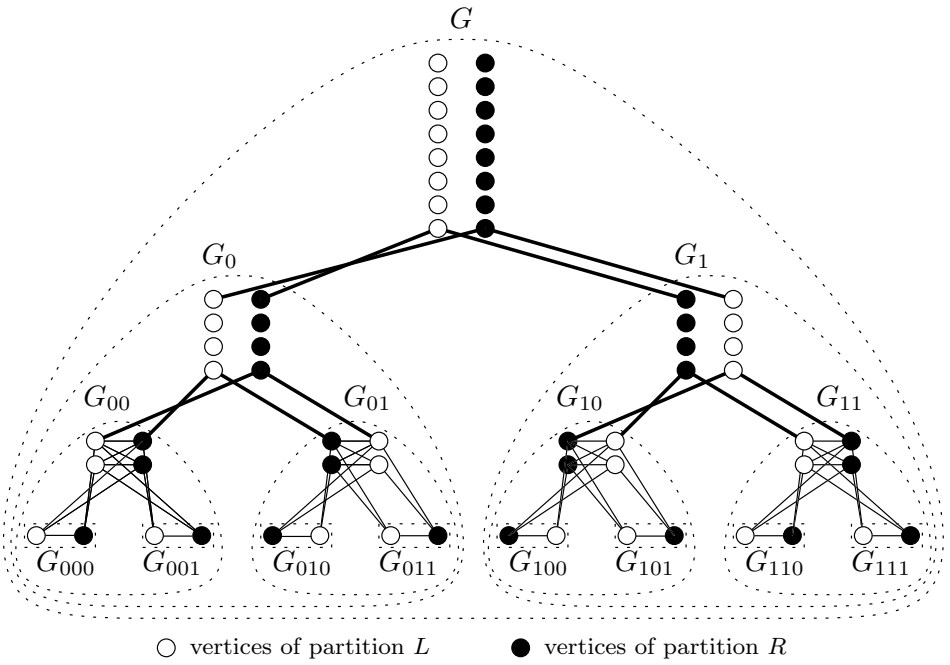


Fig. 1. The basic element G_ϵ of the lower bound graph. All edges are shown between vertices of level 1 and 0. At higher levels only schematic connections to subtrees are shown, the edges to all vertices of the opposite partition in the whole subtree are still present. The left vertex/column arrives before the corresponding right vertex/column.

- Level 0: G_α where $\alpha \in \{0,1\}^k$ consists of a single edge
- Level i: G_α where $\alpha \in \{0,1\}^{k-i}$ is obtained by
  - taking two level $i-1$ graphs $G_{\alpha 0}$ and $G_{\alpha 1}$
  - adding two sets of 2^i vertices $L_\alpha = \{l_j\}_{j=1}^{2^i} \subset L$ and $R_\alpha = \{r_j\}_{j=1}^{2^i} \subset R$
  - connecting l_j to all vertices from R in $G_{\alpha 0} \bigcup G_{\alpha 1} \bigcup R_\alpha$
  - connecting r_j to all vertices from L in $G_{\alpha 0} \bigcup G_{\alpha 1} \bigcup L_\alpha$

The construction is repeated for k levels. Note that at each level exactly 2^{k+1} vertices are added, resulting in $(k+1)2^{k+1}$ vertices in G_ϵ and $n = a(k+1)2^{k+1}$. Each graph G_ϵ can be represented by a complete binary tree H_ϵ of depth k, where the leaves correspond to G_α with $|\alpha| = k$, and each internal vertex $v_\alpha \in H_\epsilon$ represents the union of L_α and R_α. To distinguish between the vertices of the graph and vertices of the tree, we shall call the latter *nodes*. H_α will denote the subtree rooted at v_α. Let $H = \bigcup_{i=1}^{a} H_\epsilon(i)$. The *weight* $w(v)$ of each node $v \in H$ is the size of the maximum independent set of the set of vertices associated with v. Hence $w(v) = 2^i$ for nodes at level i. Clearly, $w(v)$ is an upper bound on how much the nodes of v can contribute to the independent set produced by the algorithm.

The instances $\mathcal{C}^{k,a}$ are defined by the order in which the vertices are presented to the algorithm. For each node $v \in H$, the associated vertices of G arrive as a block, not interleaved with vertices associated with other nodes of H. For each leaf v in H, there are two possible orders for the vertices associated with v to arrive. We assign label $l(v) \in \{0, 1\}$, based on whether the first vertex of v to arrive is from L or from R, respectively. Taken over all leaves, this represents 2^{a2^k} possible configurations. The label of an internal node $v_\alpha \in H$ is equal to the label of the leftmost leaf in H_α (i.e., the first vertex to arrive in H_α). The order of arrival of the vertices associated with an internal node v_α of H is fixed: First to arrive are the vertices of the same partition as was the first to arrive in $v_{\alpha 0}$. The nodes also arrive in fixed order, with the vertices of $H_\epsilon(i)$ arriving in postorder, and before the vertices of $H_\epsilon(i+1)$. The set $\mathcal{C}^{k,a}$ is hence fully described by a configuration of the leaf vertices. Since it is irrelevant which partition is L and which is R and the graphs $G_\epsilon(i)$ are disjoint, we have $|\mathcal{C}^{k,a}| = 2^{a(2^k-1)}$.

Let us now consider an arbitrary deterministic algorithm $\mathtt{A}_{det}$, and an arbitrary subset $C \subseteq \mathcal{C}^{k,a}$ of of size $|C| \geq |\mathcal{C}^{k,a}|/2^a = 2^{a(2^k-2)}$. We show that within C there exists an instance on which $\mathtt{A}_{det}$ selects an independent set of size $O(a2^k \log(k))$. We prove this by presenting Algorithm 1 constructing the configuration for which we can prove the desired bound.

In order to proceed with the arguments, we need to introduce the following notation:

- Node v_α is *dead* at time t if the algorithm has already accepted vertices from different partitions in G_α. A node which is not dead is *alive*.
- Node v_α (or subtree H_α) is *empty* at time t if the algorithm has not selected any vertex from G_α into the independent set. Otherwise v_α (H_α) is *full*.

Suppose first that $\mathtt{A}_{det}$ selects exactly one vertex from each leaf node of H, and C initially contains all instances $\mathcal{C}^{k,a}$. Consider now a leaf node v_α and its closest ancestor v_β such that $v_\alpha \in H_{\beta 1}$. Since $\mathtt{A}_{det}$ selects a vertex from every leaf, $v_{\beta 0}$ is full and in half of the instances the vertices selected in v_α and $v_{\beta 0}$ are in different partitions, making v_β and all its ancestors dead. Applying this argument over all $a(2^k - 1)$ such leaf nodes (there are only a leafs which have no such v_β as they are the leftmost ones in their respective H_ϵ) yields a configuration in which $\mathtt{A}_{det}$ has selected only $a2^k$ vertices.

In general,the situation is complicated by the fact that $\mathtt{A}_{det}$ may decide, in order to avoid killing a low-level node, not to select any vertex from a given node. Moreover, since the initial C is an arbitrary subset of $\mathcal{C}^{k,a}$, the fraction of instances that kill the parent after selecting some vertex from a given node may not be one half.

The finding of a bad instance can be viewed as a game between the algorithm and an adversary that selects the orientation of each presented node[5]. On one hand, the adversary tries to kill as many nodes as possible, as only alive nodes contribute to the independent set, on the other hand it must keep the working set of configurations sufficiently large. The adversary handles this by using a *threshold* t, whose distance (measured on logarithmic scale) from $1/2$ is directly proportional to the size of the node that would be killed: If the fraction of instances consistent with the orientation that kills the node is at least t, the adversary sets that orientation. Otherwise, it lets the node survive in order to retain enough instances in its working set. The analysis uses an accounting scheme, where the algorithm starts with a *credit* a, and gets credit when a vertex is killed, but must pay *cost* to ensure a vertex survives; at the end the account must be non-negative and the weight of the alive nodes determines the size of the independent set.

We shall now present the above ideas more formally. Let I denote the vertices accepted by the algorithm into the independent set. The next lemma shows that the algorithm cannot win too much by not selecting vertices from a presented node. In fact, for the analysis it is sufficient to consider the "decision" nodes W' that are alive and have both children full.

Lemma 6. *Let W denote the union of the sets of full alive nodes and full leaves and let W' be those nodes from W whose both children are full. Let $w(S)$ denote the sum of weights of the nodes in S. Then it holds $3w(W')+2^{k+1} \geq w(W) \geq |I|$.*

Proof. Consider a node $v \in W \setminus W'$. We charge the weight of v to its closest ancestor $u \in W'$. The nodes v form two paths (one in each child subtree of v), since branching at a node u' would mean that u' is a closer ancestor to v than u. As the weights double in each layer, the weight of each of these paths sums up to at most $w(u)$. This way, each node of W' is charged at most three times its weight.

Let W'' be the set of nodes from $W \setminus W'$ which do not have an ancestor in W'. We have $3w(W') + w(W'') > w(W)$. We show by induction on height h that for each $v \in W''$ of level h, $w(W'' \bigcap H_v) \leq 2^{h+1}$. In the base case $h = 0$ we have a single leaf of weight 2, hence the statement trivially follows. Consider now an internal node v at level h. There are two cases: either v is alive or it is dead. In the case v is alive, from the fact that $v \notin W'$ it follows that one of its children is empty. Hence $w(W'' \bigcap H_v) = w(v) + w(W'' \bigcap H_{v'})$,

[5] For a leaf node, this means selecting the order in which the vertices arrive. For an internal node, this means flipping the orientation of one child's subtree. Note that since the partitions L and R are symmetric, this can be done without affecting the algorithm's behavior.

Algorithm 1. Finding the bad instance

1: Let C be the set of configurations for which a given deterministic algorithm applies.
2: **for** all nodes $v_\alpha \in H_\epsilon(i)$ in the traversal order specified above **do**
3: **if** v_α is not a leaf **then**
4: Let C_0 and C_1 be the partitioning of C into sets of configurations consistent with
$l(v_{\alpha 0}) = l(v_{\alpha 1})$ and $l(v_{\alpha 0}) \neq l(v_{\alpha 1})$, respectively.
5: **if** v_α is dead, or ($v_{\alpha 0}$ or $v_{\alpha 1}$ is empty) **then**
6: Set C to be the larger of C_0 and C_1.
7: **else** ▷ hence v_α is alive and both of its children are full.
8: Let C_x be the configurations in which the algorithm has selected in G_α vertices from both partitions.
9: Let j be the level of the highest-level live ancestor of v_α.
10: Set $t(j)$ satisfying $\log(t_j) = -1 - 1/2^{k-j}$.
11: **if** $|C_x|/|C| > t(j)$ **then** ▷ note that C_x might be empty
12: Set $C \leftarrow C_x$. ▷ this kills v_α and all its ancestors
13: **else**
14: Set $C \leftarrow C_{1-x}$. ▷ v_α survives, but C has not decreased much
15: **end if**
16: **end if**
17: **end if**
18: Deliver the vertices of $G_\alpha(i)$ and observe algorithm's action.
19: **end for**
20: Output the configuration consistent with what has happened so far.

where v' is the full child of v'. By induction hypothesis $w(W'' \bigcap H_{v'}) \leq 2^h$. As $w(v) = 2^h$, the induction step follows. In the case of v being dead, let v' and v'' be its two children. Applying the induction hypothesis yields $w(W'' \bigcap H(v)) = w(W'' \bigcap H(v')) + w(W'' \bigcap H(v'')) \leq 2^h + 2^h = 2^{h+1}$.

Finally, note that in a dead internal node, the algorithm cannot accept any vertices into the independent set. An alive internal node v consists of two partitions, each containing $w(v)$ vertices, hence in v the algorithm can accept at most $w(v)$ vertices. Only in leaves can the algorithm accept vertices even if the node becomes dead – but such leaves are included in W. This yields $w(W) \geq |I|$. □

Let p (pass), d (dead) and l (live) denote the number of times the lines 6, 12 and 14 have been executed, respectively. Let $D = \{v \in H$: line 12 was applied when processing $v\}$ and let $L = \{v \in H$:line 14 was applied when processing $v\}$. Note that L is actually the set of decision nodes W' from Lemma 6, since a node v which remained alive after processing will never be killed later. This is because only v's children can kill v, and those have been processed before v. Let j_v for $v \in (D \cup L)$ be the level j from line 9 when processing v. For a node $v \in L$, let us define $cost(v) = \log(1 - t(j_v)) + 1$. Similarly, for a node $v \in D$, define $credit(v) = -\log(t(j_v)) - 1 = 2^{j-k}$. Analogously, for a set S, define $cost(S) = \sum_{v \in S} cost(v)$ and $credit(S) = \sum_{v \in S} credit(v)$. The next lemma bounds the overall cost that the algorithm may pay:

Lemma 7. $cost(L) < a(k+1)$

Let us now find an upper bound on the size of the independent set accepted by $\mathtt{A}_{det}$. From Lemma 6 we know that it is sufficient to bind $w(L)$. Hence, the problem now becomes "*Maximize* $w(L)$ *while satisfying* $cost(L) < a(k+1)$".

Note that if a node $v \in L$ has an alive parent u which is not in L, the set $L' = L \cup \{u\} \setminus \{v\}$ has the same cost but higher weight. Hence, it is sufficient to consider sets L in which $\forall u, v$ such that u is an ancestor of v, the whole path from v to u is in L. Therefore, L induces in H a set of r rooted trees $\{T_i\}_{i=1}^{r}$. Let v_i is the root of T_i. Then, for every node $u \in T_i$, the highest-level live ancestor of u at the moment u is processed is either v_i or its ancestor, hence $j(u) \geq level(v_i)$ and therefore $cost(T_i) \geq |T_i| cost(v_i)$. As T_i is a subtree of a binary tree and the weights of nodes halve as the levels decrease, we obtain $w(T_i) \leq \log(|T_i|+1)w(v_i)$ Since $cost(.)$ is a concave function, $w(L)$ is maximized when $r = a$ and all v_i are at the top level k, with $|T_i| = (k+1)/cost(v_i) = (k+1)/(\log(3)-1)$. Using $w(v_i) = 2^k$ gives $w(L) \leq a \log\left(\frac{k+1}{\log(3)-1} - 1\right) 2^k \in O(a2^k \log(k).)$ Applying Lemma 6 yields $|I| \in O(a2^k \log(k))$, and Theorem 5 follows.

4 Upper Bound for Bipartite Graphs

While Theorem 5 indicates that quite a lot of advice is needed to approach a competitive ratio one, it seems rather weak when relatively little advice is available. In this section we show that it is not weak at all, and very little advice is sufficient to bring the competitive ratio to $O(\log n)$.

For a fixed input graph G with bipartitions L and R, and a fixed arrival order, let us observe the connected components C_i formed by the vertices presented so far. Each of them is a connected bipartite graph, and the algorithm knows its bipartition. However, it does not know which partition of C_i corresponds to L. Let each C_i have a distinguished partition P_{C_i} maintained by the algorithm. Let us assign to each vertex v, and each respective component, a meta-level $m(v)$ using Algorithm 2.

Consider a component C of meta-level l and let v be the first vertex of meta-level l in C, i.e., the vertex that formed C as a component of meta-level l on line 10 of Algorithm 2. By construction the algorithm can, when creating a new component C, always choose the P_C in a consistent way. Let $ms(i)$ denote the minimal size of a component of meta-level i. By construction, we have recurrences $ms(1) = 1$ and $ms(i+1) \geq 2ms(i)+1$, yielding $ms(i) \geq 2^i - 1$. This immediately gives us:

Lemma 8. $\forall v \in G : ms(v) \leq \log(n+1)$

Let l be a fixed meta-level and let $\{C_1, C_2, \ldots, C_r\}$ be the set of connected components of meta-level l. Let V_i be the set of vertices of meta-level l in C_i (note that V_i might be disconnected) and let $V(l) = \bigcup_{i=1}^{r} V_i$. Let $I(l) = \bigcup_{i=1}^{r} V_i \bigcap P_{C_i}$ and $\bar{I}(l) = \bigcup_{i=1}^{r} V_i \bigcap \overline{P_{C_i}}$, where $\bar{P}$ denotes the opposite partition to P. Since the components C_i are disjoint, both $I(l)$ and $\bar{I}(l)$ are independent sets, even

Algorithm 2. Algorithm Meta-levels

if a newly arriving vertex v is singleton **then**
  Set $m(v) \leftarrow 1$, and $m(\{v\}) \leftarrow 1$
else if v is connected to a single connected component C **then**
  Set $m(v) \leftarrow m(C)$, and $m(C \bigcup \{v\}) \leftarrow m(C)$
else
  Let v be connected to components $C_1, C_2, \ldots, C_s$, ordered by meta-levels in non-increasing order.
  if $m(C_1) > m(C_2)$ **then**
    Set $m(v) \leftarrow m(C_1)$ and $m(\{v\} \bigcup C_1 \bigcup C_2 \bigcup \ldots C_s) \leftarrow m(C_1)$.
  else ▷ $m(C_1) = m(C_2)$
    Set $m(v) \leftarrow m(C_1) + 1$ and $m(\{v\} \bigcup C_1 \bigcup C_2 \bigcup \ldots C_s) \leftarrow m(C_1) + 1$.
  end if
end if

though P_{C_i}'s might belong to different partitions of G. Knowing l, a single bit of advice telling which of $I(l)$ and $\bar{I}(l)$ is bigger is sufficient for the algorithm to select an independent set of size $V(l)/2$.

Let m be the meta-level with the maximal number of vertices. From Lemma 8 we have $V(l) \geq n/\log(n+1)$. Therefore, the algorithm that uses $1 + \log\log n$ bits of advice to identify the meta-level m with the largest $V(m)$, and the larger of $I(m)$ and $\bar{I}(m)$, selects an independent set of size $O(n/\log n)$, yielding.

Theorem 9. *$O(\log\log n)$ bits of advice[6] are sufficient to achieve competitive ratio of $O(\log n)$ for the online independent set in bipartite graphs.*

Consider now the case that a bits of advice are available, with $a \geq \log\log n$. In such case, the algorithm can use this advice to learn the orientation of meta-components and improve the competitive ratio.

Theorem 10. *There is an algorithm that using $a \geq \log\log n$ bits of advice achieves competitive ratio $O(\log(n/a))$ for the online independent set problem in bipartite graphs.*

5 Conclusion

We were interested in the relation of the competitiveness of online MIS in unknown graph and known supergraph models in terms of advice complexity. Without any advice, the competitive ratio is $\Omega(n)$ in both models, even restricted to sparse bipartite graphs. We showed that in sparse graphs, constant advice is sufficient in both models to achieve a constant competitive ratio. In bipartite graphs, however, the models differ significantly since 1 bit of advice is sufficient to achieve competitive ratio 2 in known supergraph model, whereas $\Omega(n)$ bits are needed to achieve a constant competitive ratio in the unknown graph model.

[6] Since n is not known, a self-delimited encoding will be used, at a cost of small constant factor increase in the number of bits used.

The use of the advice as a general way to measure the relevant information about unknown input is an attractive alternative to ad-hoc solutions. Although not practical in its general form (e.g., no computability constraints), it may help in characterizing the key structural properties that affect the performance of online algorithms.

Acknowledgement. The authors acknowledge the support of the VEGA agency under the grants 1/0671/11, and 2/0136/12. The research originated at the "Mountains and Algorithms" workshop organized by Juraj Hromkovič in August, 2011.

References

1. Bartal, Y., Fiat, A., Leonardi, S.: Lower bounds for on-line graph problems with application to on-line circuit and optical routing. In: STOC 1996, pp. 531–540. ACM (1996)
2. Bartal, Y., Fiat, A., Leonardi, S.: Lower bounds for on-line graph problems with application to on-line circuit and optical routing. SIAM J. Comput. 36(2), 354–393 (2006)
3. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
4. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
5. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge Univ. Press (1998)
6. Caragiannis, I., Fishkin, A.V., Kaklamanis, C., Papaioannou, E.: Randomized on-line algorithms and lower bounds for computing large independent sets in disk graphs. Discrete Applied Mathematics 155(2), 119–136 (2007)
7. Christodoulou, G., Zissimopoulos, V.: On-line maximum independent set in chordal graphs. Journal of Foundations of Computing and Decision Sciences 30(4), 283–296 (2005)
8. Cieślik, I.: On-line Graph Coloring. PhD thesis, Jagiellonian University, Krakow, Poland (2004)
9. Demange, M., Paradon, X., Paschos, V.T.: On-line maximum-order induced hereditary subgraph problems. In: Jeffery, K., Hlaváč, V., Wiedermann, J. (eds.) SOFSEM 2000. LNCS, vol. 1963, pp. 327–335. Springer, Heidelberg (2000)
10. Dereniowski, D., Pelc, A.: Drawing maps with advice. J. Par. Distrib. Comput. 72(2), 132–143 (2012)
11. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem-relevant information in input. ITA 43(3), 585–613 (2009)
12. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theor. Comput. Sci. 412(24), 2642–2656 (2011)
13. Escoffier, B., Paschos, V.T.: On-line models and algorithms for max independent set. RAIRO - Operations Research 40(2), 129–142 (2006)
14. Fraigniaud, P., Gavoille, C., Ilcinkas, D., Pelc, A.: Distributed computing with advice: information sensitivity of graph coloring. Distributed Computing 21(6), 395–403 (2009)

15. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Tree exploration with advice. Inf. Comput. 206(11), 1276–1287 (2008)
16. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Communication algorithms with advice. J. Comput. Syst. Sci. 76(3-4), 222–232 (2010)
17. Fraigniaud, P., Korman, A., Lebhar, E.: Local mst computation with short advice. Theory Comput. Syst. 47(4), 920–933 (2010)
18. Fusco, E.G., Pelc, A.: Trade-offs between the size of advice and broadcasting time in trees. Algorithmica 60(4), 719–734 (2011)
19. Halldórsson, M.M.: Online coloring known graphs. In: SODA 1999, pp. 917–918. ACM/SIAM (1999)
20. Halldórsson, M.M., Iwama, K., Miyazaki, S., Taketomi, S.: Online independent sets. Theor. Comput. Sci. 289(2), 953–962 (2002)
21. Håstad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Mathematica 182, 105–142 (1999)
22. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
23. Ilcinkas, D., Kowalski, D.R., Pelc, A.: Fast radio broadcasting with advice. Theor. Comput. Sci. 411(14-15), 1544–1557 (2010)
24. Kubale, M.: Graph colorings. Contemporary Mathematics, vol. 352. AMS (2004)
25. Lipton, R.J., Tomkins, A.: Online interval scheduling. In: SODA 1994, pp. 302–311 (1994)
26. Randerath, B., Schiermeyer, I.: Vertex colouring and forbidden subgraphs - a survey. Graphs and Combinatorics 20(1), 1–40 (2004)

Online Multi-Commodity Flow with High Demands*

Guy Even and Moti Medina**

School of Electrical Engineering,Tel-Aviv Univ., Tel-Aviv 69978, Israel
{guy,medinamo}@eng.tau.ac.il

Abstract. This paper deals with the problem of computing, in an online fashion, a maximum benefit multi-commodity flow (ONMCF), where the flow demands may be bigger than the edge capacities of the network.

We present an online, deterministic, centralized, all-or-nothing, bi-criteria algorithm. The competitive ratio of the algorithm is constant, and the algorithm augments the capacities by at most a logarithmic factor.

The algorithm can handle two types of flow requests: (i) low demand requests that must be routed along a path, and (ii) high demand requests that may be routed using a multi-path flow.

Two extensions are discussed: requests with known durations and machine scheduling.

Keywords: Online algorithms, primal-dual scheme, multi-commodity flow.

1 Introduction

We study the problem of computing a multi-commodity flow in an online setting (ONMCF). The network is fixed and consists of n nodes and m directed edges with capacities. The adversary introduces flow requests in an online fashion.

A flow request r_j is specified by the source node s_j, the target node t_j, the demand d_j, i.e., the amount of flow that is required, and the benefit b_j, i.e., the credit that is given for a served request.

We focus on an *all-or-nothing* scenario, where a credit b_j is given only if a request r_j is fully served, otherwise, the credit is zero. Given a sequence of flow requests, the goal is to compute a multi-commodity flow (MCF) that maximizes the total benefit of *fully served* requests. Our algorithm can deal with high demands d_j. In particular, the demand d_j may be bigger than the maximum capacity.

Our Contribution. We present a centralized, deterministic, all-or-nothing, non-preemptive online algorithm for the ONMCF problem with high demands. The

* The full version of this paper can be found in http://arxiv.org/abs/1201.5030.
** M.M was partially funded by the Israeli ministry of Science and Technology.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 16–29, 2013.

algorithm is $O(1)$-competitive. The algorithm violates edges capacities by an $O(\log n)$ factor.

We show how to extend the algorithm so that it handles two types of flow requests: (i) low demand requests that must be routed along a path, and (ii) high demand requests that may be routed using a multi-path flow.

Finally, two extensions are discussed: requests with known durations and machine scheduling.

1.1 Previous Work

Online multi-commodity flow was mostly studied in the context of *single path routing*. The *load* of an edge e in a network is the ratio between the flow that traverses e and its capacity.

Online routing was studied in two settings: (1) throughput maximization, i.e., maximizing the total benefit gained by flow requests that are served [1–3], and (2) load minimization, i.e., routing all requests while minimizing the maximum load of the edges [4, 5, 2, 6].

In these two settings the following variants are considered: (1) permanent routing [4, 2, 3], (2) unknown durations [5], and (3) known durations [1, 3, 6].

Load Minimization. In the case of permanent routing, Aspnes et. al [4] designed an algorithm that augments the edge capacities by a factor of at most $O(\log n)$ w.r.t. a feasible optimal routing. Buchbinder and Naor [2] obtained the same result by applying a primal-dual scheme. This result can be extended to requests with high demands. The extension is based on a min-cost flow oracle that replaces the shortest path oracle.

Aspnes et. al [4] also showed how to use approximated oracles to allocate Steiner trees in the context of multicast virtual circuit routing. They obtained a competitive ratio of $O(\log n)$. Recently Bansal et.al [6] extended this result to bi-criteria oracles and showed how to embed d-depth trees and cliques in the context of resource allocation in cloud computing. In the case of cliques, they required that the pairwise demands are uniform and smaller than the edge capacities. For the case of clique embedding they obtained a competitive ratio of $O(\log^3 n \cdot \log(nT))$ w.r.t. a feasible optimal solution, where T is the ratio between the maximum duration to the minimum duration of a request.

Throughput Maximization. For the case of known durations, Awerbuch et. al [1] designed an $O(\log(nT))$ competitive algorithm, where T is the maximum request duration. This algorithm requires that the demands are smaller than the edge capacity by a logarithmic factor. Buchbinder and Naor [2, 7] introduced the primal-dual scheme in the online setting and designed a bi-criteria algorithm that is 1-competitive while augmenting edge capacities by a factor of $O(\log n)$ for the case of unit demands and unit benefits.

Recently Even et. al [3] showed how to apply the primal-dual scheme to embed a variety of traffic patterns in the context of Virtual Networks (VNETs). Their goal is to maximize the profit of the served VNET requests. Some of the results in [3] require solving the ONMCF problem with high demands.

1.2 Approaches for Online MCF with High Demands

We briefly discuss the weaknesses of approaches for solving the ONMCF problem that rely directly on previous algorithms.

The algorithms in [1, 2] route each request along a single path. They require that the demand is smaller than the capacities. In order to apply these methods one should augment the capacities in advance so that the requested demand is bounded by the bottleneck along each path from the source to the destination. This augmentation might be polynomial compared to the logarithmic augmentation requirement by our algorithm.

Another option is to split the requests into subrequests of small demand so that each subdemand is smaller than the minimum capacity. After that, a single-path online routing algorithm [1, 2] can be used to route each of these subrequests. In this case, some of the subrequests might be rejected, hence violating our all-or-nothing requirement.

We define the *granularity* of a flow as the smallest positive flow along an edge in the network. Let ε denote the granularity of a flow. One can formulate the multi-commodity problem as a packing linear problem and apply the methods in [7, 8]. The edge capacity augmentation of these algorithms depends on the $\log(\frac{1}{\varepsilon})$, which might be unbounded. For example, consider the following network: (1) The set of nodes is $V = \{u, v\}$, (2) there are two unit capacity parallel edges (u, v). Consider a request with demand $d_j = 1 + \varepsilon$, for $\varepsilon < 1$. If the flow oracle computes an all-or-nothing flow that routes flow of size 1 on one edge and ε on the other, then the granularity is ε.

In order to solve this granularity problem, one can apply [7, 8] and apply randomized rounding to obtain an all-or-nothing solution with unit granularity. Even in the unit-demand case, this technique increases the competitive ratio from $O(1)$ to $O(\log n)$ while the edge capacity augmentation is $O(\log n)$. Our result shows that an $O(1)$-competitive ratio is achievable.

1.3 Techniques

Our algorithm is based on the *online primal-dual scheme*. The online primal-dual scheme by Buchbinder and Naor [8, 7, 2] invokes an "min-weight" path oracle. The oracles considered in [8, 7, 2] are either exact oracles or approximate oracles. Bansal et al. [6] use bi-criteria oracles. Namely, the oracles they considered are approximated and augment the edge capacities. We need tri-criteria oracles.

We extend the online primal-dual scheme so it supports *tri-criteria* oracles. In the context of MCF, the oracles compute min-cost flow. The three criteria of these oracles are: (1) the approximation ratio, (2) the capacity augmentation of the edges, and (3) the granularity of the computed flow.

Multiple criteria oracles were studied by Kolliopoulos and Young [9]. They presented bi-criteria approximation algorithms for covering and packing integer programs. Their algorithm finds an approximate solution while violating the packing constraints. The granularity property is used in [9] to mitigate this violation.

2 Problem Definition

Online multi-commodity flow (ONMCF) is defined as follows.

The Network. Let $G = (V, E)$ denote a directed graph, where V is the set of nodes and E is the set of directed edges of the network. Let $n \triangleq |V|$, and $m \triangleq |E|$. Each edge $e \in E$ has a capacity $c_e \geq 1$.

The Input. The online input is a sequence of requests σ, i.e., $\sigma = \{r_j\}_{j \in \mathbb{N}^+}$. Each flow request is a 4-tuple $r_j = (s_j, t_j, d_j, b_j)$. Let $s_j, t_j \in V$ denote, respectively, the source node and the target node of the jth request. Let $d_j \geq 1$ denote the flow demand for the jth request. Let $b_j \geq 1$ denote the benefit for the jth request. We consider an online setting, namely, the requests arrive one-by-one, and no information is known about a request r_j before its arrival.

The Output. The output is a multi-commodity flow $F = (f_1, f_2, \ldots)$. For each request r_j, f_j is a flow from s_j to t_j.

Terminology. Let $|f_j|$ denote the amount of flow of f_j. Let $f_j(e)$ denote the jth flow along the edge $e \in E$. Finally, for every $e \in E$, $F^{(j)}(e) \triangleq \sum_{k=1}^{j} f_k(e)$, that is, the accumulated flow along an edge e after request r_j is processed. We say that an MCF $F = (f_1, f_2, \ldots)$ *fully serves* a request r_j if $|f_j| = d_j$. We say that an MCF F *rejects* a request r_j if $|f_j| = 0$. We say that an MCF is *all-or-nothing* if each request is either fully served or rejected. An all-or-nothing MCF is credited b_j for each fully served request r_j. We say that an online MCF (ONMCF) algorithm is *monotone* if flow is never retracted. We say that an online MCF (ONMCF) algorithm is *preemptive* if the flow f_j of a fully served request r_j is retracted entirely, i.e., $|f_j| = 0$. A monotone ONMCF algorithm is, in particular, *non-preemptive*.

The Objective. The goal is to compute an all-or-nothing ONMCF that maximizes the total benefit of the served requests.

2.1 The Main Result

We present an online algorithm for the ONMCF problem that satisfies the following properties: (i) The algorithm is centralized and deterministic. (ii) There is no limitation on demands. In particular, $\min_j d_j$ may exceed $\max_e c_e$. (iii) The algorithm is all-or-nothing. (iv) The online algorithm ALG competes with an all-or-nothing offline optimal algorithm. (v) The algorithm is $(1 + \delta)$-competitive, for a constant $\delta \in (0, 1]$. (vi) The algorithm violates the capacity constraints by an $O(\log n)$ factor. (vii) The algorithm is non-preemptive and monotone.

For a vector $x = (x_1, \ldots, x_k)$, let $x_{\min} \triangleq \min_i x_i$. Similarly, $x_{\max} \triangleq \max_i x_i$. The main result is formalized in Theorem 1.

Theorem 1 (Main Result). *Let γ denote a constant. Assume that: (i) $1 \leq b_{\min} \leq b_{\max} \leq O(n^{\gamma})$, (ii) $1 \leq c_{\min} \leq c_{\max} \leq O(n^{\gamma})$, (iii) $1 \leq d_{\min}$. Then, Algorithm 1 is a non-preemptive, monotone, online algorithm for the* ONMCF *problem that computes an all-or-nothing multi-commodity flow that is $(O(1), O(\log n))$-competitive*[1]*, that is, the computed flow is $O(1)$-competitive and it violates the capacity constraints by an $O(\log n)$ factor.*

3 Online Packing and Covering Formulation

In this section we present a sequence of packing linear programs (LPs) that correspond to the ONMCF problem. We also present covering linear programs. We refer to the covering programs as the primal LPs and to the packing programs as the dual LPs.

3.1 Flow Polytopes

We define polytopes of flows that correspond to the requests $\{r_j\}_{j \in \mathbb{N}^+}$ as follows.

Definition 1. *For every $r_k = (s_k, t_k, d_k, b_k)$, let $\Pi_k(\mu)$ denote the polytope of* unit *flows f (i.e., $|f| = 1$) from s_k to t_k in G that satisfy: $\forall e \in E : f(e) \leq \mu \cdot \frac{c_e}{d_k}$.*

We refer to $\Pi_k(1)$ simply by Π_k. Let $V(\Pi_k(\mu))$ denote the set of extreme points of $\Pi_k(\mu)$.

Definition 2. *We say that request r_k is* μ-feasible *if $\Pi_k(\mu) \neq \emptyset$. We say that request r_k is* feasible *if $\Pi_k \neq \emptyset$.*

Note that a request r_j is μ-feasible if and only if the capacity of the minimum cut that separates s_k from t_k is at least $\frac{d_k}{\mu}$. In particular, a request r_j may be feasible even if $d_j > \max_e c_e$.

3.2 Packing and Covering Formulation

For every prefix of requests $\{r_k\}_{k=1}^{j}$ we define a primal linear program P-LP(j) and a dual linear program D-LP(j). The LP's appear in Figure 1.

The packing program D-LP(j) has a variable y_f for every flow $f \in \bigcup_k V(\Pi_k)$ and two types of constraints: *demand constraints* and *capacity constraints*. The capacity constraints require that the load on every edge e is at most c_e. The demand constraints require that the conical combination of unit flows in $V(\Pi_k)$ is a flow of size at most d_k.

The covering program P-LP(j) has a variable x_e for every edge $e \in E$, and a variable z_k for every request r_k, where $k \leq j$. It is useful to view x_e as the cost of a unit flow along e.

[1] Actually, we prove a competitive ratio of 3/2.

P-LP(j)	D-LP(j)
$\min \sum_{k=1}^{j} d_k \cdot z_k + \sum_{e \in E} c_e \cdot x_e$ s.t. $\forall k \in [1,j] \forall f \in V(\Pi_k) : z_k + \sum_{e \in E} x_e \cdot f(e) \geq \frac{b_k}{d_k}$ $x, z \geq \mathbf{0}$	$\max \sum_{k=1}^{j} \sum_{f \in V(\Pi_k)} \frac{b_k}{d_k} \cdot y_f$ s.t. $\forall e \in E : \sum_{k=1}^{j} \sum_{f \in V(\Pi_k)} f(e) \cdot y_f \leq c_e$ $\forall k \in [1,j] : \sum_{f \in V(\Pi_k)} y_f \leq d_k$ $y \geq \mathbf{0}$
(I)	(II)

Fig. 1. (I) The primal LP P-LP(j). (II) The dual LP D-LP(j): The first type of constraints are the capacity constraints. The second type of constraints are the demand constraints.

4 The Online Algorithm ALG

In this section we present the online algorithm ALG.

4.1 Preliminaries

The algorithm maintains the following variables: (1) For every edge e the primal variable x_e, (2) for every request r_j the primal variable z_j, and (3) the multi-commodity flow F. The primal variables x, z are initialized to zero. The MCF F is initialized to zero as well.

Notation. Let $x_e^{(j)}$ denote the value of the primal variable x_e after request r_j is processed by ALG.

For every request r_j, let $cost_j(f)$ denote the x-cost of a flow f, formally:

$$cost_j(f) \triangleq \sum_e x_e^{(j-1)} \cdot f(e) \,.$$

For every flow f, let $w(f)$ denote the sum of the flows along the edges, formally:

$$w(f) \triangleq \sum_e f(e) \,.$$

Let $F^{(k)}$ denote the MCF F after request r_k is processed. Let $benefit_j(F)$ denote the benefit of MCF F after request r_j is processed, formally:

$$benefit_j(F) \triangleq \sum \{b_i \mid i \leq j, r_i \text{ is fully served by } F^{(j)}\} \,.$$

Let $value_j(x,z)$ denote the objective function's value of P-LP(j) for a given x and z, formally:

$$value_j(x,z) \triangleq \sum_{k=1}^{j} d_k \cdot z_k + \sum_{e \in E} c_e \cdot x_e^{(j)} \,.$$

Let F^* denote an all-or-nothing offline optimal MCF w.r.t input sequence $\sigma = \{r_j\}_j$.

Definition 3. *An* MCF $F = (f_1, f_2, \ldots)$ *is* (α, β)-competitive *with respect to a sequence* $\{r_j\}_j$ *of requests if for every* j*: (i)* F *is* α*-competitive:* $\text{benefit}_j(F) \geq \frac{1}{\alpha} \cdot \text{benefit}_j(F^*)$*. (ii)* F *is* β*-feasible: for every* $e \in E$*,* $F^{(j)}(e) \leq \beta \cdot c_e$*.*

Definition 4. *An* MCF $F = (f_1, f_2, \ldots)$ *is* all-or-nothing *if each request* r_j *is either fully served by* F *or it is rejected by* F *(i.e.,* $|f_j| \in \{0, d_j\}$*).*

4.2 Description

Upon arrival of a request r_j, if the request is not feasible, then the algorithm rejects it upfront. Otherwise, if the request is feasible, then ALG invokes a tri-criteria oracle. The oracle returns a unit-flow f_j for r_j.

If the cost of the oracles's flow is "small enough", then the request is accepted as follows: (1) the flow F is updated by adding the oracle's unit-flow f_j times the required demand d_j, (2) the primal variables x_e, for every edge e that the flow f_j traverses, are updated.

If the flow is "too expensive", then the request is rejected and no updates are made to the primal variables and to the MCF F.

The listing of the online algorithm ALG appears in Algorithm 1.

Algorithm 1 ALG: Online multi-commodity flow algorithm. The algorithm receives a sequence of requests and outputs a multi-commodity flow F.

Initialize: $z \leftarrow 0, x \leftarrow 0, F \leftarrow 0$.
Upon arrival of request $r_j = (s_j, t_j, d_j, b_j)$, for $j \geq 1$:

1) If r_j is not feasible (i.e., $\Pi_j = \emptyset$), then **reject** r_j and skip the remaining lines.
2) $f_j \leftarrow \mathit{oracle}(x, r_j)$ {The oracle is a $(\lambda, \mu, \varepsilon)$-criteria.}
3) If $d_j \cdot \mathit{cost}_j(f_j) < \lambda \cdot b_j$,
   4) then **accept** r_j
      5) $F \leftarrow F + d_j \cdot f_j$ {Updating the multi-commodity flow.}
      6) $z_j \leftarrow \frac{b_j}{d_j} - \frac{\mathit{cost}_j(f_j)}{\max\{\lambda, \mu\}}$
      7) $\forall e : f_j(e) > 0$:

$$L_j(e) \triangleq \frac{d_j \cdot f_j(e)}{\max\{\lambda, \mu\} \cdot c_e}, \quad x_e \leftarrow x_e \cdot 2^{L_j(e)} + \frac{1}{d_j \cdot w(f_j)} \cdot \left(2^{L_j(e)} - 1\right)$$

8) Else **reject** r_j

4.3 The Oracle

The oracle description is as follows:

(i) **Input:** Request r_j, edge capacities $\frac{c_e}{d_j}$, and edge costs $x^{(j-1)} : E \to \mathbb{R}^{\geq 0}$.
(ii) **Output:** A unit-flow f from s_j to t_j.

Let MIN-COST$_j$ denote the min-cost flow in Π_j w.r.t. the costs x_e, formally:

$$\text{MIN-COST}_j \triangleq \arg\min\{cost_j(f) : f \in \Pi_j\} .$$

Note that: (1) MIN-COST$_j$ is well defined because $\Pi_j \neq \emptyset$, and (2) the edge capacities in Π_j are $\frac{c_e}{d_j}$.

The oracles in our context are tri-criteria, as formalized in the following definition.

Definition 5 (Oracle Criteria). *We say that an oracle is* $(\lambda, \mu, \varepsilon)$-criteria, *if the oracle outputs a flow* f *that satisfies the following properties: (i)* ***(λ-Approximation.)*** $\text{cost}_j(f) \leq \lambda \cdot \text{cost}_j(\text{MIN-COST}_j)$. *(ii)* ***($\mu$-Augmentation.)*** $f \in \Pi_j(\mu)$. *(iii)* ***(ε-Granular.)*** $f(e) > 0 \Rightarrow f(e) \geq \varepsilon$.

A Tri-criteria Oracle for Minimum Cost Flow. The oracle's listing is as follows.

The Oracle Outline.

1. Let $f \leftarrow$ MIN-COST$_j$.
2. Decompose f to at most m flow paths $\{f_1, \ldots, f_m\}$.
3. Remove each flow path f_ℓ such that $|f_\ell| < \frac{1}{2m^2}$.
4. Let g denote the removed flow from f.
5. Scale every remaining flow path f_ℓ (i.e., $|f_\ell| \geq \frac{1}{2m^2}$): $f_\ell \leftarrow f_\ell \cdot \left(1 + \frac{|g|}{|f|-|g|}\right)$

The proof of the following lemma appears in Appendix A.

Lemma 1. *The oracle is* $(2, 2, \frac{1}{2m^2})$*-criteria algorithm.*

Lemma 1 justifies using the following parameters: $\lambda = \mu = 2$, and $\varepsilon = \frac{1}{2m^2}$.

5 Analysis

The following observation is proved by the fact that $\frac{1}{z} \cdot (2^z - 1)$ is monotone increasing for $z > 0$.

Observation 1. *Let* $c \in \mathbb{R}^{>0}$, *then* $\forall x \in [0, c] : c \cdot (2^{x/c} - 1) \leq x$.

Observation 2. *If* $L_j(e) \leq 1$, *then* $\left(2^{L_j(e)} - 1\right) \cdot c_e \leq \frac{1}{\max\{\lambda,\mu\}} \cdot d_j \cdot f_j(e)$.

Proof. By Observation 1 and since $L_j(e) \leq 1$, it follows that $\left(2^{L_j(e)} - 1\right) \cdot \max\{\lambda, \mu\} \cdot c_e \leq d_j \cdot f_j(e)$, and the observation follows.

Notation. Let

$$\alpha \triangleq 1 + \frac{1}{\max\{\lambda,\mu\}} \leq 2, \quad \beta \triangleq \max\{\lambda,\mu\} \cdot \log_2\left(1 + m^2 \cdot \frac{3 \cdot \lambda \cdot c_{\max} \cdot b_{\max}}{\varepsilon}\right).$$

In the following theorem we prove that ALG is an all-or-nothing (α,β)-competitive, non-preemptive and monotone online algorithm.

Theorem 2. *Assume that: (i) $b_{\min}, c_{\min}, d_{\min} \geq 1$. (ii) The oracle is $(\lambda,\mu,\varepsilon)$-criteria. Then* ALG *is non-preemptive, monotone, online algorithm for the* ON-MCF *problem that computes an all-or-nothing multi-commodity flow that is (α,β)-competitive.*

Proof. The algorithm ALG rejects upfront requests that are not feasible. These requests are also rejected by F^*, hence it suffices to prove (α,β)-competitiveness w.r.t fractional offline optimal algorithm over the feasible requests. We now prove α-competitiveness and β-feasibility.

α-competitiveness. First, we prove α-competitiveness. Let $\Delta_j P \triangleq \mathit{value}_j(x,z) - \mathit{value}_{j-1}(x,z)$, and $\Delta_j F \triangleq \mathit{benefit}_j(F) - \mathit{benefit}_{j-1}(F)$. We begin by proving that $\Delta_j P \leq \alpha \cdot \Delta_j F$ for every request r_j.

Recall that $x_e^{(j)}$ denotes the value of the primal variable x_e after r_j is processed. If r_j is rejected then $\Delta_j P = \Delta_j F = 0$ and the claim holds. If r_j is accepted, then $\Delta_j F = b_j$ and $\Delta_j P = \sum_e (x_e^{(j)} - x_e^{(j-1)}) \cdot c_e + d_j \cdot z_j$. Let f_j denote the output of the oracle when dealing with request r_j, i.e., $f_j \leftarrow \mathit{oracle}(x^{(j-1)}, r_j)$. Indeed,

$$\begin{aligned}
\sum_e \left(x_e^{(j)} - x_e^{(j-1)}\right) \cdot c_e &= \sum_e \left[x_e^{(j-1)} \cdot \left(2^{L_j(e)} - 1\right) + \frac{1}{d_j \cdot w(f_j)} \cdot \left(2^{L_j(e)} - 1\right)\right] \cdot c_e \\
&= \sum_e \left(x_e^{(j-1)} + \frac{1}{d_j \cdot w(f_j)}\right) \cdot \left(2^{L_j(e)} - 1\right) \cdot c_e \\
&\leq \sum_e \left(x_e^{(j-1)} + \frac{1}{d_j \cdot w(f_j)}\right) \cdot \frac{d_j \cdot f_j(e)}{\max\{\lambda,\mu\}} \\
&= \frac{d_j \cdot \mathit{cost}_j(f_j)}{\max\{\lambda,\mu\}} + \frac{1}{\max\{\lambda,\mu\}}, \qquad (1)
\end{aligned}$$

where the third inequality holds since the oracle is μ-augmented and by Observation 2. Hence, Equation 1 and Step 6 of ALG imply that:

$$\begin{aligned}
\Delta_j P &\leq \frac{d_j \cdot \mathit{cost}_j(f_j)}{\max\{\lambda,\mu\}} + \frac{1}{\max\{\lambda,\mu\}} + d_j \cdot z_j \\
&= \frac{d_j \cdot \mathit{cost}_j(f_j)}{\max\{\lambda,\mu\}} + \frac{1}{\max\{\lambda,\mu\}} + d_j \cdot \left(\frac{b_j}{d_j} - \frac{\mathit{cost}_j(f_j)}{\max\{\lambda,\mu\}}\right) \\
&= \frac{1}{\max\{\lambda,\mu\}} + b_j \leq \alpha \cdot b_j \,,
\end{aligned}$$

where the last inequality holds since $b_j \geq 1$.

Since $\Delta_j F = b_j$ it follows that

$$\Delta_j P \leq \alpha \cdot \Delta_j F \,, \tag{2}$$

as required.

Initially, the primal variables and the flow F equal zero. Hence, Equation 2 implies that:

$$value_j(x,z) \leq \alpha \cdot benefit_j(F) \,. \tag{3}$$

We now prove that, the primal variables $\{x_e^{(j)}\}_e \cup \{z_i\}_{i \leq j}$ constitute a feasible solution for P-LP(j):

1. If r_j is rejected, then $cost_j(f_j) \geq \lambda \cdot \frac{b_j}{d_j}$. Since the oracle is λ-approximate it follows that for every $f' \in V(\Pi_j)$: $cost_j(f') \geq cost_j(\text{MIN-COST}_j) \geq cost_j(f_j)/\lambda \geq \frac{b_j}{d_j}$. It follows that the primal constraints are satisfied in this case.
2. If r_j is accepted, then $cost_j(f_j) < \lambda \cdot \frac{b_j}{d_j}$. Since $z_j = \frac{b_j}{d_j} - \frac{cost_j(f_j)}{\max\{\lambda,\mu\}}$ it follows that for every $f' \in V(\Pi_j)$: $z_j + cost_j(f') \geq \frac{b_j}{d_j} - \frac{cost_j(f_j)}{\max\{\lambda,\mu\}} + \frac{cost_j(f_j)}{\lambda} \geq \frac{b_j}{d_j}$. We conclude that the primal constraints are satisfied in this case as well.

The first j flows of the optimal offline multi-commodity flow F^* are clearly a feasible solution to D-LP(j). The value of this solution equals $benefit_j(F^*)$. Since the primal variables constitute a feasible primal solution, weak duality implies that: $benefit_j(F^*) \leq value_j(x,z)$. Hence, by Equation 3, it follows that: $benefit_j(F) \geq \frac{1}{\alpha} \cdot benefit_j(F^*)$, which proves that ALG is α-competitive.

β-feasibility. We now prove β-feasibility, i.e., for every r_i and for every $e \in E$, $F^{(i)}(e) \leq \beta \cdot c_e$.

We prove a lower bound and an upper bound on x_e in the next two lemmas. The proofs of the next two lemmas appear in Appendices B, C. Let r_j denote the index of the last request. Let

$$W \triangleq \max\{d_k \cdot w(f_k) : 0 \leq k \leq j\}.$$

Lemma 2. *For every edge e, $x_e \geq \frac{1}{W} \cdot \left(2^{F(e)/\max\{\lambda,\mu\} \cdot c_e} - 1\right)$.*

Lemma 3. *For every accepted request r_k, if $f_k(e) > 0$, then $x_e^{(k)} \leq \frac{3 \cdot \lambda \cdot b_k}{\varepsilon \cdot d_k}$.*

Lemma 2 and Lemma 3 imply that: $\frac{1}{W} \cdot \left(2^{F^{(k)}(e)/\max\{\lambda,\mu\} \cdot c_e} - 1\right) \leq \max_{k \leq j} \frac{3 \cdot \lambda \cdot b_k}{\varepsilon \cdot d_k}$.

Hence,

$$F^{(k)}(e) \leq c_e \cdot \max\{\lambda, \mu\} \cdot \log_2 \left(1 + W \cdot \max_k \frac{3 \cdot \lambda \cdot b_k}{\varepsilon \cdot d_k}\right) . \tag{4}$$

Since (i) $W \leq m \cdot d_{\max}$, (ii) $d_{\max} \leq m \cdot c_{\max}$, and (iii) $d_{\min} \geq 1$, it follows that, $F^{(k)}(e) \leq \beta \cdot c_e$, for every k, as required.

This concludes the proof of Theorem 2. Theorem 1 follows directly from Theorem 2 and Lemma 1.

Remark 1. *Let bpb_k denote the benefit-per-bit of request r_k, i.e., $bpb_k \triangleq \frac{b_k}{d_k}$. Let $bpb_{\max} \triangleq \max_k bpb_k$. Instead of β, the augmentation can be also bounded by:*

$$\max\{\lambda, \mu\} \cdot \log_2 \left(1 + W \cdot \frac{3 \cdot \lambda}{\varepsilon} \cdot bpb_{\max}\right) .$$

6 Mixed Demands

One may consider a mixed case of low and high demands. A flow request with high demand has to be split into multiple paths. Splitting a stream of packets along multiple paths should be avoided, if possible, because it complicates implementation in nodes where flow is split, may cause packets to arrive out-of-order, etc. Thus, one may require not to split requests with low demand. Formally, a request has low demand if $d_j \leq c_{\min}$; otherwise, it has a high demand.

An online algorithm for mixed demands can be obtained by employing two oracles: (1) A tri-criteria oracle for the high demands. This oracle may serve a flow request by multiple paths. (2) An exact (shortest path) oracle for low demands. This oracle must serve a flow request by a single path.

Theorem 3. *There exists a non-preemptive, monotone, online algorithm for the* ONMCF *problem with mixed demands that computes an all-or-nothing multicommodity flow that is $(O(1), O(\log n))$-competitive.*

Proof (Proof sketch). The proof is based on the feasibility of the primal LP and on the bounded gap between $\Delta_j F$ and $\Delta_j P$. These two invariants are maintained regardless of the oracle that is invoked. The proof for the case of small demands appears in [8]. The augmentation of the capacities are determined by the oracle with the "worst" parameters. Because the exact oracle is $(1, 1, 1)$-criteria, it is also (λ, μ, ϵ)-criteria. Thus, the augmentation factor β is determined by the approximate oracle.

7 Further Extensions

Requests with known durations. The algorithm can be extended to deal with flow requests with known durations. For the sake of simplicity, the flow requests

in this paper are permanent, namely, after arrival, a request stays forever. Using previous techniques [1–3], our algorithm can be adapted to deal also with the important variant of known durations. In this variant, each request, upon arrival, also has an end-time. The competitive ratio for known durations when the requests are a logarithmic fraction of the capacities is $O(\log(nT))$, where T denotes the longest duration [1]. In fact, the primal-dual method in [2] can be extended to the case of routing requests with known durations (see [3]). Thus, for known durations, if the demands are bounded by the minimum capacity, then the primal-dual method yields an online algorithm, the competitive ratio of which is $(O(1), O(\log(nT)))$. One can apply a tri-criteria oracle with granularity $O(n^{-2})$, to obtain an $(O(1), O(\log(nT)))$-competitive ratio for known durations even with high demands.

All-or-Nothing Machine Scheduling. A simple application of our algorithm is the case of maximizing throughput in an online job all-or-nothing scheduling problem on unrelated machines. The variant in which the objective is to minimize the load was studied by Aspnes et al. [4]. We, on the other hand, focus on maximizing the throughput.

Jobs arrive online, and may be assigned to multiple machines immediately upon arrival. Moreover, a job may require specific subset of machines, i.e., restricted assignment. The increase in the load of a machine when a job is assigned to it is a function of the machine and the fraction of the job that is assigned to it. Formally, Let $\tau_j(e) \in [0, 1]$ denote the "speed up" of machine e when processing job j, that is, one unit of job e on machine j incurs an additional load of $\tau_j(e)$ on machine e. The reduction is to network of m parallel edges, one edge per machine. The capacity of each edge equals the capacity of the corresponding machine.

Large jobs need to be assigned to multiple machines, while small jobs may be assigned to a single machine (as in [4]). In this case our algorithm is $(O(1), O(\frac{\log m}{\min_{j,e} \tau_j(e)}))$-competitive, where m is the number of machines.

References

1. Awerbuch, B., Azar, Y., Plotkin, S.: Throughput-competitive on-line routing. In: FOCS 1993: Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science, pp. 32–40. IEEE Computer Society, Washington (1993)
2. Buchbinder, N., Naor, J.S.: Improved bounds for online routing and packing via a primal-dual approach. In: Annual IEEE Symposium on Foundations of Computer Science, pp. 293–304 (2006)
3. Even, G., Medina, M., Schaffrath, G., Schmid, S.: Competitive and deterministic embeddings of virtual networks. In: Bononi, L., Datta, A.K., Devismes, S., Misra, A. (eds.) ICDCN 2012. LNCS, vol. 7129, pp. 106–121. Springer, Heidelberg (2012); CoRR abs/1101.5221 (January 2011)
4. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. Journal of the ACM (JACM) 44(3), 486–504 (1997)

5. Awerbuch, B., Azar, Y., Plotkin, S., Waarts, O.: Competitive routing of virtual circuits with unknown duration. Journal of Computer and System Sciences 62(3), 385–397 (2001)
6. Bansal, N., Lee, K.W., Nagarajan, V., Zafer, M.: Minimum congestion mapping in a cloud. In: PODC, pp. 267–276 (2011)
7. Buchbinder, N., Naor, J.S.: Online primal-dual algorithms for covering and packing. Math. Oper. Res. 34(2), 270–286 (2009)
8. Buchbinder, N., Naor, J.S.: The design of competitive online algorithms via a primal-dual approach. Foundations and Trends in Theoretical Computer Science 3(2-3), 99–263 (2009)
9. Kolliopoulos, S., Young, N.: Approximation algorithms for covering/packing integer programs. Journal of Computer and System Sciences 71(4), 495–505 (2005)

A Proof of Lemma 1

In this section we prove the following lemma.

Lemma 1 The oracle is $(2, 2, \frac{1}{2m^2})$*-criteria algorithm.*

Proof. Throughout this proof we refer to flow path that are not removed, simply by 'flow paths'.

First, we prove that the oracle outputs a unit flow. For every flow path f_ℓ such that $|f_\ell| \geq \frac{1}{2m^2}$, let $f_\ell^{(s)}$ denote the scaled flow along it. The sum of the flows, along the scaled flow paths equals:

$$\sum_\ell f_\ell^{(s)} = \sum_\ell f_\ell \cdot \left(1 + \frac{|g|}{|f| - |g|}\right) = |f| - |g| + |g| = |f| \,.$$

Hence, the oracle outputs a unit flow as required.

The oracle is $\frac{1}{2m^2}$-granular by construction.

We prove that the oracle is 2-augmented. Note that $|g| \leq \frac{m}{2m^2} = \frac{1}{2m}$. Hence,

$$\frac{|g|}{|f| - |g|} \leq \frac{\frac{1}{2m}}{1 - \frac{1}{2m}} = \frac{1}{2m - 1} \,.$$

It follows that the flow along every edge is augmented by at most

$$\left(1 + \frac{|g|}{|f| - |g|}\right) \leq \left(1 + \frac{1}{2m - 1}\right) < 2 \,.$$

Hence, $f \in \Pi_j(2)$, as required.

Moreover, the flow along every scaled flow path $f_\ell^{(s)}$ satisfies for every $e \in E$:

$$f_\ell^{(s)}(e) \leq f_\ell(e) \cdot \left(1 + \frac{1}{2m - 1}\right) \,,$$

Hence,

$$cost_j(f_\ell^{(s)}) \leq cost_j(f_\ell) \cdot \left(1 + \frac{1}{2m - 1}\right) \,,$$

which proves 2-approximation of the oracle.

B Proof of Lemma 2

In this section we prove the following lemma.

Lemma 2 For every edge e,

$$x_e \geq \frac{1}{W} \cdot \left(2^{F(e)/\max\{\lambda,\mu\}\cdot c_e} - 1\right) .$$

Proof. Recall that $x^{(k)}$ (resp. $F^{(k)}$) denote the value of x (resp. F) after request r_k is processed. We prove by induction on $k \leq j$ that

$$x_e^{(k)} \geq \frac{1}{W} \cdot \left(2^{F^{(k)}(e)/\max\{\lambda,\mu\}\cdot c_e} - 1\right). \tag{5}$$

The induction basis, for $k = 0$, holds because both sides equal zero.

Induction step: Note that if r_k is rejected, then both sides of Equation 5 remain unchanged, and hence Equation 5 holds by the induction hypothesis. We now consider the case that r_k is accepted.

The update rule in Step 7 of ALG implies that

$$\begin{aligned} x_e^{(k)} &= x_e^{(k-1)} \cdot 2^{\frac{d_k \cdot f_k(e)}{\max\{\lambda,\mu\}\cdot c_e}} + \frac{1}{d_j \cdot w(f_k)} \cdot \left(2^{\frac{d_k \cdot f_k(e)}{\max\{\lambda,\mu\}\cdot c_e}} - 1\right) \\ &\geq \frac{1}{W} \cdot \left(2^{\frac{F^{(k-1)}(e)}{\max\{\lambda,\mu\}\cdot c_e}} - 1\right) \cdot 2^{\frac{d_k \cdot f_k(e)}{\max\{\lambda,\mu\}\cdot c_e}} + \frac{1}{d_j \cdot w(f_k)} \cdot \left(2^{\frac{d_k \cdot f_k(e)}{\max\{\lambda,\mu\}\cdot c_e}} - 1\right) \\ &\geq \frac{1}{W} \cdot \left(2^{\frac{F^{(k)}(e)}{\max\{\lambda,\mu\}\cdot c_e}} - 1\right). \end{aligned}$$

The lemma follows.

C Proof of Lemma 3

In this section we prove the following lemma.

Lemma 3 For every accepted request r_k, if $f_k(e) > 0$, then $x_e^{(k)} \leq \frac{3\cdot\lambda\cdot b_k}{\varepsilon\cdot d_k}$.

Proof. Since r_k is accepted, we have $d_k \cdot cost_k(f_k) < \lambda \cdot b_k$. By ε-granularity of the oracle, $cost_k(f_k) \geq x_e^{(k-1)} \cdot \varepsilon$. It follows that $x_e^{(k-1)} \leq \frac{\lambda\cdot b_k}{d_k\cdot\varepsilon}$. By the update rule for x_e, we have:

$$x_e^{(k)} \leq \frac{\lambda \cdot b_k}{d_k \cdot \varepsilon} \cdot 2^{L_k(e)} + \frac{1}{d_k \cdot w(f_k)} \cdot \left(2^{L_k(e)} - 1\right) .$$

Since the oracle is μ-augmented, $L_k(e) \leq 1$. In addition, since the oracle is ε-granular, $w(f_j) \geq \varepsilon$ it follows:

$$x_e^{(k)} \leq \frac{\lambda \cdot b_k}{d_k \cdot \varepsilon} \cdot 2 + \frac{1}{d_k \cdot \varepsilon} \cdot (2 - 1) \leq \frac{3 \cdot \lambda \cdot b_k}{d_k \cdot \varepsilon} ,$$

as required.

Approximating Spanning Trees with Few Branches*

Markus Chimani[1,**] and Joachim Spoerhase[2]

[1] Inst. of Computer Science, Friedrich-Schiller-University Jena, Germany
markus.chimani@uni-jena.de
[2] Inst. of Computer Science, University of Würzburg, Germany
joachim.spoerhase@uni-wuerzburg.de

Abstract. Given an undirected, connected graph, the aim of the *minimum branch-node spanning tree* problem is to find a spanning tree with the minimum number of nodes with degree larger than 2. The problem is motivated by network design problems where junctions are significantly more expensive than simple end- or through-nodes, and are thus to be avoided. Unfortunately, it is NP-hard to recognize instances that admit an objective value of zero, rendering the search for guaranteed approximation ratios futile.

We suggest to investigate a *complementary* formulation, called *maximum path-node spanning tree*, where the goal is to find a spanning tree that maximizes the number of nodes with degree at most two. While the optimal solutions (and the practical applications) of both formulations coincide, our formulation proves more suitable for approximation. In fact, it admits a trivial 1/2-approximation algorithm. Our main contribution is a local search algorithm that guarantees a ratio of 6/11.

1 Introduction

Let G be an undirected, connected graph. Finding a spanning tree T of G with the smallest number of *branch nodes* (nodes of degree at least 3) is known as the *minimum branch-node spanning tree* problem (MBST). It was introduced by Gargano et al. [6], where it is pointed out that MBST can also be seen as a natural optimization version of the Hamiltonian path problem since any spanning tree without branch nodes is a Hamiltonian path and vice versa.

Practical applications of MBST can be found in the design of optical networks. Gargano et al. [6] consider *light trees* as a means to realize all-optical multicast (sending information from one source to multiple sinks). Classical wavelength division multiplexing technology (WDM) supports only unicast connections (i.e., light paths). In a WDM network, multicast can therefore only be achieved by establishing multiple unicast connections leading to a decreased performance.

* This research started from an open problem posed at Dagstuhl Seminar 11191. The authors are grateful for Dagstuhl's support.
** Markus Chimani was funded by a Carl-Zeiss-Foundation juniorprofessorship.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 30–41, 2013.

Light trees rely on a new technology—light-splitting switches—capable of replicating light signals by splitting light. These sophisticated and thus expensive switches have to be placed at exactly the branch nodes of the light tree, which naturally leads to MBST.

Gargano et al. [6] give bounds on the minimum number of branch nodes depending on other density conditions and on graph parameters such as connectivity, independence number, and the length of a longest paths. In later works [4,5], sufficient density conditions for the existence of *spanning spiders*—spanning trees with at most one branch node—are considered.

Salamon [14] gives a different practical motivation for MBST, which is also based on optical networks. Moreover, he provides logarithmic upper bounds on the required number of branch nodes depending on the total number of nodes and the density of the input graph. Cerulli et al. [2] develop ILP formulations for MBST which allows them to solve small instance to optimality. For larger instances they propose three different heuristic approaches. Silva et al. [15] propose heuristic algorithms for MBST based on an iterative refinement idea.

In this paper, we are interested in algorithms for MBST with provable quality guarantees. Unfortunately, the above problem formulation is not suitable for analyzing approximation algorithms, due to its close relation to the NP-hard Hamilton path problem. It is already NP-hard to decide whether a given instance allows a "perfect" solution, i.e., a solution without any branch nodes. The main problem arises from the observation that such a solution, a Hamiltonian path, corresponds to an objective value $OPT = 0$. These facts not only show that approximating MBST within a bounded ratio is NP-hard [6], but that if $OPT = 0$ then the ratio of *every* suboptimal solution to the optimum one is unbounded and therefore equally bad. It would, however, be more reasonable to assign spanning trees with fewer branch nodes a better approximative performance than ones with more branch nodes.

We consider the *complementary* formulation of MBST whose objective function is the number of *path nodes*. Put formally, we aim at finding a spanning tree of the input graph that *maximizes* the number of nodes with degree at most 2. We denote such low-degree nodes as *path nodes*, as they are exactly the nodes that arise in paths, and consequently call this reformulation *maximum path-node spanning tree* (MPST). Although the original and our formulation of the problem lead to the same optimum solutions (and have the same practical applications), ours is more appropriate for analyzing approximation algorithms. In fact, every spanning tree is a 1/2-approximation for MPST since at least half the nodes have degree 1 or 2.

Notice that complementary problem formulations have also been successfully considered in the context of similar optimization problems. For example, Lu an Ravi [10] show that there is no constant-factor approximation algorithm for the problem of finding a spanning tree with a minimum number of leaves. Later, the complementary problem of maximizing the number of internal nodes has been suggested in [11] and lead to a series of improved constant-factor approximation algorithms [12,13,9] as well as a large body of literature on fixed-parameter

algorithms. Two other examples of similar complementary problem pairs that have received significant attention in the algorithmic literature are the *maximum leaf spanning tree* [16] and *minimum connected dominating set* [7] problems, as well as the *full-degree spanning tree* [1] and its complementary problem [8].

There are several local search approximation algorithms for spanning tree problems known in the literature [8,10,9]. One of the most influential works in this context is the additive 1-approximation of the maximum degree spanning tree problem by Fürer and Raghavachari [3].

Our Contribution and Organization of the Paper. We propose an approximation algorithm for the maximum path-node spanning tree problem. The algorithm is based on local search and has a ratio of 6/11. This beats the (trivial) approximation factor of 1/2, which is often a critical bound for combinatorial optimization problems (e. g., vertex cover, k-center, maximum leaf spanning tree, full-degree spanning tree, etc.).

The paper is organized as follows. In Section 2 we describe our local search approximation algorithm. The analysis of the performance guarantee of the algorithm consitutes the main body of this work and is presented in Section 3. Finally, we make some observations concerning the tightness of our analysis in Section 4.

2 Algorithm

As pointed out above every spanning tree of the input graph is already a 1/2-approximation since at least one half of the nodes in a tree have degree 1 or 2. The worst case of ratio 1/2 occurs when the considered spanning tree consists of only leaves and cubic nodes (nodes with degree 3) while the optimum is a Hamiltonian path or a star with a single high-degree node. We will improve this ratio following the idea that we can obtain a ratio better than 1/2 if we can ensure that a certain fraction of the nodes that are non-cubic in the optimum are also non-cubic in our solution.

A *feasible k-flip* replaces k edges of T with the same number of edges in $E(G) - E(T)$ such that the resulting graph T' is a spanning tree of G. A k-flip is *improving* if it either increases the number of path nodes (degree 1 or 2), or if the number of path nodes remains constant while the number of cubic nodes (degree 3) decreases. For simplicity, a $\bar{k}$-flip denotes any ℓ-flip with $1 \leq \ell \leq k$.

Algorithm. Starting with an arbitrary spanning tree, perform improving $\bar{2}$-flips as long as there exist any. ◇

The main part of this paper is devoted to show that this algorithms guarantees a solution with at least 6/11 of the optimum solution's path nodes. Before we do so, we show:

Lemma 1. *The above algorithm runs in polynomial time. In particular, it only requires $O(|V(G)|)$ flips.*

Proof. It is clear that checking for improving $\bar{2}$-flips can be done in polynomial time (e.g., by evaluating all $O(|E(G)|^4)$ possible $\bar{2}$-flips). Furthermore, the definition of improving flips gives rise to a lexicographical objective function of two parts: the most significant part is n_P, the number of path nodes where more is better; to compare solutions with equal n_P, we consider the number of cubic nodes n_C where less is better. Any improving flip either increases n_P (while possibly also increasing n_C) or leaves n_P unchanged and decreases n_C. Since both measures are non-negative and bounded by $|V(G)|$ from above, a trivial upper bound of $O(|V(G)|^2)$ iterations follows. We can improve on this by realizing that a flip increasing n_P cannot increase n_C arbitrarily: Since any $\bar{2}$-flip touches at most $8 = O(1)$ nodes, the sum of all increases of n_C is bounded by $O(|V(G)|)$, and hence our algorithm performs at most $O(|V(G)|)$ flips overall. □

3 Analysis of Approximation

In the following, T always denotes a locally optimal spanning tree for G obtained by our algorithm and we partition the nodes into subsets based on their degree in T: L denotes the set of leaves in T (T-*leaves*), F the set of degree-2-nodes in T (*forward nodes*[1]), $P := L \cup F$ the set of path nodes, C the set of degree-3 (*cubic*) nodes, and H the set of nodes with degree ≥ 4 in T (*high-degree nodes*). Generally, we may use graph theoretic terms with preceding graph specification to specify whether we consider whole G or only T. E.g., we use the term T-*neighbors* to denote neighbors of a node in T. Let T^* be a spanning tree in G with a maximum number of path nodes, and $C' \subseteq C$ be the set of cubic nodes in T that are *not* cubic in T^*. We observe that the set $C' \cup H$ contains all nodes (if any) that are path nodes in T^* but not in T.

3.1 Overview

The nodes $C \cup H$ are the branch nodes in T. Observe that $|L| = 2 + \sum_{v \in C \cup H}(\deg(v) - 2)$. To see this, perform a breadth-first search in T. Observe that whenever a node v is expanded the number of leaves increases by $\deg(v) - 2$ if v is a branch node. If v is a path node the number of leaves does not change. Hence we can think of each branch node v as contributing $\deg(v) - 2$ leaves to the tree. Rearranging the above equation yields

$$|L| = 2 + |C \cup H| + \sum_{v \in H}(\deg(v) - 3) \geq |C' \cup H| + \sum_{v \in H}(\deg(v) - 3)\,.$$

We interpret this inequality so that each branch node in $C' \cup H$ is guaranteed to contribute one *regular* leaf to T and that each node in H contributes $\deg(v) - 3$ *supplementary* leaves to T. For the proof, we neither require an explicit assignment of leaves to nodes in $C' \cup H$, nor an explicit designation of the leaves as

[1] The term *forward node* is motivated by the technical background of WDM networks, and, e.g., consistent with the definition of maximum forward-node spanning trees [14].

regular or supplementary. We will only argue over their respective number. Yet, the reader may find it helpful to consider any arbitrary but fixed assignment and designation. Assume we can show that T contains at least $1/5 \cdot |C' \cup H|$ supplementary leaves or forward nodes. Together with the $|C' \cup H|$ regular leaves this implies the existence of at least $6/5 \cdot |C' \cup H|$ path nodes in T. Since the nodes in $C \setminus C'$ are cubic in T^* by definition of C' we have that $\mathrm{OPT} \leq |P| + |C' \cup H|$. Hence the performance guarantee of our algorithm is bounded by

$$\frac{|P|}{\mathrm{OPT}} \geq \frac{|P|}{|P| + |C' \cup H|} \geq \frac{|P|}{|P| + 5/6|P|} = \frac{6}{11}\,. \tag{1}$$

It remains how to show the existence of at least $1/5 \cdot |C' \cup H|$ supplementary leaves or forward nodes in T. We consider the following coin transfer scheme: Initially, each node in $C' \cup H$ receives one *coin*. In three stages, described in detail below, each node in C' transfers its coin either to a forward node or to a high-degree node in T. In doing so, we ensure that the following conditions hold after the transfer scheme:

(P1) Only nodes in $F \cup H$ hold coins,
(P2) each node in F holds at most five coins, and
(P3) each node $v \in H$ holds at most $\deg(v) + 1$ coins.

Property (P3) guarantees that $v \in H$ holds at most $(\deg(v)+1)/(\deg(v)-3) \leq 5$ coins for each of its $\deg(v) - 3$ supplementary leaves. Clearly, if there exists a transfer scheme with these properties, there must exist at least $1/5 \cdot |C' \cup H|$ supplementary leaves or forward nodes.

3.2 Transfer Scheme – Stage I

For each $u \in C'$ consider its T-neighbors. If any T-neighbor lies in $F \cup H$ then u transfers its coin to this neighbor (breaking ties arbitrarily). It is clear that any node in $v \in F \cup H$ receives at most $\deg(v)$ coins in Stage I.

3.3 Transfer Scheme – Stage II

Let $u \in C'$ be a node that still owns its coin after Stage I. This implies that all T-neighbors of u are in $L \cup C$. We can show the following.

Lemma 2. *Assume $u \in C'$ has only T-neighbors in $L \cup C$ and there exists an edge $(u, v) \in E(G) \setminus E(T)$. Then, $v \in F$.*

Proof. Assume $v \notin F$. Consider the 1-flip where we add (u, v) to T and remove the unique other edge (u, w) incident to u in T that lies on the so-established cycle. By this operation, $\deg(u)$ does not change, $\deg(v)$ is increased by one, and $\deg(w)$ is decreased by one. Clearly, w was not in L before and hence was a cubic node that becomes a forward node. Since $v \notin F$, v does not become a cubic node but is in P or H after the 1-flip. Overall, our flip would increase the number of path nodes, which contradicts the local optimality of T. □

In Stage II we process all nodes u that satisfy the condition of Lemma 2. For each such node we transfer the coin of u to a node $v \in F$ that is G-adjacent to u. According to Lemma 2 such a node always exists.

Lemma 3. *No two nodes that are processed in Stage II transfer their coins to the same node.*

Proof. Assume there are two distinct cubic nodes u_1, u_2 processed in stage B which assign their coin to some node $v \in F$. Consider adding the two edges (u_1, v) and (u_2, v) to T. The introduction of each of these edges (without introducing the other one) establishes a cycle: let (u_1, w_1) and (u_2, w_2) the unique other incident edges to u_1 and u_2, respectively, that lie on the respective cycle. Now, consider the 2-flip removing (u_1, w_1) and (u_2, w_2) and instead inserting (u_1, v) and (u_2, v). The remaining structure is clearly still a tree, and we can investigate how the node degrees change. The nodes w_1 and w_2 are cubic nodes, as they cannot be leaves while lying on a cycle, and their neighboring u_1, u_2 were not able to transfer a coin to them in Stage I. Obviously, $\deg(v)$ is increased by 2 and becomes a degree-4 node. By case distinction we show that this 2-flip would always either increase the number of path nodes, or—when this number does not change—decrease the number of cubic nodes; both contradicts the local optimality of T.

- $w_1 \neq w_2$, $w_2 \neq u_1$ and $w_1 \neq u_2$: The nodes u_1, u_2 remain cubic, but the cubic nodes w_1 and w_2 become path nodes.
- $w_1 = w_2$: The nodes u_1, u_2 remain cubic, but the cubic node w_1 becomes a leaf.
- $w_2 = u_1$ or $w_1 = u_2$: Assume, w.l.o.g. $u_1 = w_2$. Then, u_2 remains cubic, but the cubic nodes u_1 and w_1 become path nodes. □

Applying this lemma iteratively, we obtain:

Corollary 1. *Any forward node gains at most one coin in Stage II.*

3.4 Transfer Scheme – Stage III

A node $u \in C'$ is called *unprocessed* if it has not been processed in Stage I or II. This implies that u is T-adjacent only to nodes in $L \cup C$ and moreover, there is no edge $e \in E(G) \setminus E(T)$ incident on u. Since u lies in C' there must be an edge $e = (u, v) \in E(T) \setminus E(T^*)$, for, otherwise u would be cubic in T^*, too. If there are multiple (at most 2) such edges, we pick one of these as e arbitrarily. We call edge e *critical*. Note that v itself may also be unprocessed, and e therefore critical due to both.

Let V_u, V_v be the node sets of the two connected components obtained by removing a critical edge (u, v) from T (let $u \in V_u$). Since T^* is connected there must be an edge $e^* = (x, y) \in E(T^*)$ with $x \in V_u$ and $y \in V_v$. Of course $x \neq u$ since u has not been processed in Stage II. Note that $T + e^* - e$ is a tree, i.e., the corresponding 1-flip is feasible.

Lemma 4. *At least one of the nodes x, y is a forward node in T.*

Proof. Assume both x, y are not in F, and consider a 1-flip removing (u, v) and inserting (x, y). If $v = y$, $\deg(v)$ does not change, the cubic node u becomes a path node, and $\deg(x)$ increases by one. Since $x \notin F$ does not become cubic, the flip would be improving.

If $v \neq y$, then v cannot have been a T-leaf. Consequently, the cubic nodes u and v become path nodes; the non-path-nodes x, y increase their degree by one and cannot become cubic. Again, the flip would be improving, which is a contradiction to T's local optimality. □

In Stage III we process all yet unprocessed nodes in C'. Let u be one of these nodes. The idea is that there is at least one node $y \in F$ associated with u according to Lemma 4. We transfer the coin of u to this node. If this is done in an arbitrary manner we cannot rule out that one forward node gains a large number of coins in Stage III. In what follows we develop a refined transfer scheme that ensures that each forward node in T gains *at most two* coins in Stage III.

First, we transfer all coins located at unprocessed nodes to their respective critical edges. This gives at most two coins per critical edge. Then we identify for each critical edge e a *replacement edge* $r(e) \in E(T^*) - E(T)$ such that the 1-flip replacing e with $r(e)$ leads to a tree. According to Lemma 4 one of the endpoints of $r(e)$ is a forward node in T. We will construct the mapping $r(\cdot)$ so that the number of forward nodes incident on replacement edges is greater than or equal to the number of critical edges. This allows us to transfer the coins from the critical edges to forward nodes incident to their respective replacement edges such that each forward node receives at most two coins.

It remains to show the existence of such a mapping $r(\cdot)$. We first show:

Lemma 5. *There exists an injective mapping $r(\cdot)$ from critical edges to replacement edges, i.e., $r(e) \neq r(e')$ for distinct critical edges e, e'.*

Proof. Let $e = (u, v)$ be a critical edge and let R_e be the set of edges in $E(T^*) \setminus E(T)$ that cross the cut (V_u, V_v). The edges in R_e are exactly the set of edges that are candidates for $r(e)$. Now consider an arbitrary subset E' of critical edges and let $\{V_1, \ldots, V_{|E'|+1}\}$ be the connected components of the forest $T - E'$. Since T^* is connected there are at least $|E'|$ edges in T^* connecting distinct components $V_i \neq V_j$. Let E'' be the set of such edges. Note that each edge in E'' lies in R_e for some $e \in E'$ since nodes in different components $V_i \neq V_j$ are separated in T by some critical edge in E'. Hence $|\bigcup_{e \in E'} R_e| \geq |E''| \geq |E'|$. By Hall's marriage theorem there is an injective mapping r as required above. □

In what follows we show that the set of endpoints of replacement edges contains at least as many forward nodes as there are critical edges, whenever we have an injective mapping $r(\cdot)$. We know by Lemma 4 that each replacement edge has a forward node in T as one of its endpoints. We face, however, the problem that two replacement edges might be adjacent leading to multiple countings of forward nodes.

The following lemma shows that two replacement edges can only be adjacent at a forward node if they do not allow a feasible 2-flip.

Lemma 6. *Let e, e' be distinct critical edges. Assume that the 2-flip replacing e, e' with $r(e), r(e')$ leads to a tree. Then $r(e)$ and $r(e')$ are not incident on a common forward node.*

Proof. Let $e = (u, v)$ and $e' = (u', v')$ be distinct critical edges with $u \neq u'$ as unprocessed nodes and let $r(e) = (x, y)$ and $r(e') = (x', y')$ be their replacement edges. By way of contradiction assume that both replacement edges are incident on forward node $x = x'$.

Now consider the 2-flip replacing e, e' with $r(e)$, $r(e')$. Due to our assumption this 2-flip leads to a tree T'. We show below that this is an improving 2-flip, contradicting the local optimality of T.

To this end let $D = \{u, u', v, v'\}$ be the set of nodes incident on the edges e, e' and let $I = \{x, y, y'\}$ be the set of nodes incident on the edge $r(e), r(e')$. Note that $x \notin D$ since x is a forward node and D contains only leaves and cubic nodes. Furthermore, the 2-flip changes the degree of x from 2 to 4. We distinguish two cases and show that in both cases the flip would be improving.

- D contains at least three cubic nodes: The removal of e and e' transforms at least three cubic nodes in D into path nodes. By the addition of $r(e)$ and $r(e')$, at most three path nodes are destroyed; yet, at most two of these latter nodes (namely y, y') can become cubic.
- D contains less then three cubic nodes: Then $u \neq u'$ are cubic and v, v' are distinct T-leaves. Hence $y = v$ and $y' = v'$ since otherwise one of these nodes would be disconnected by the 2-flip. The removal of e, e' creates two forward nodes (namely u, u'). The addition of $r(e)$, $r(e')$ destroys only x's path node property, as the nodes v, v' remain leaves after the flip. □

The following two lemmas show that two critical edges can only share a common forward node in T if all of their endpoints are forward nodes. In some sense this compensates the negative effect of "multiple counting".

Lemma 7. *Let e be a critical edge and let e^* be an edge in T^* that is not adjacent to e. If the 1-flip replacing e with e^* is feasible then both endpoints of e^* are forward nodes in T.*

Proof. Let $e = (u, v)$ and let $e^* = (x, y)$. Since e and e^* are not adjacent and the 1-flip replacing e with e^* is feasible the T-path connecting x and y contains u, v as interior nodes. Hence u and v are not leaves in T. Assume w.l.o.g. that u is unprocessed and hence cubic. Then v must also be cubic since u is only adjacent to leaves or cubic nodes. Now, if one of the nodes x, y were not a forward node then the above 1-flip would be improving contradicting the local optimality of T. □

Lemma 8. *Let e, e' be distinct critical edges. Assume that $r(e)$ and $r(e')$ are incident on the same forward node. Then all three endpoints of $r(e)$ and $r(e')$ are forward nodes.*

Proof. According to Lemma 6 the 2-flip replacing e, e' with $r(e), r(e')$ in T does not lead to a tree.

If we delete e, e' from T we obtain three connected components X, Y, Z. W.l.o.g. assume that e connects X, Y and e' connects Y, Z. Since the 1-flips replacing e and e' with $r(e)$ and $r(e')$, respectively, are feasible but the above 2-flip is *not* feasible we conclude that both $r(e)$ and $r(e')$ must connect X and Z.

Let x be the forward node shared by $r(e)$ and $r(e')$. Assume w.l.o.g. that x lies in X. Since x is a forward node it is not incident on the critical edge e. Since the other endpoints of $r(e)$ and $r(e')$ both lie in Z but e connects X, Y none of the edges $r(e)$ or $r(e')$ can be adjacent to e.

Now we observe that the 1-flip replacing e with $r(e)$ and the 1-flip replacing e (instead of e') with $r(e')$ are both feasible and satisfy the requirements of Lemma 7. Hence all three endpoints of $r(e)$ and $r(e')$ are forward nodes. □

Now consider the forest $\mathcal{F}$ consisting of all replacement edges. We split $\mathcal{F}$ at all nodes that are not forward nodes in T. As a result of this, $\mathcal{F}$ consists of isolated edges and non-trivial components containing at least two edges. Two adjacent edges can only share a node that is a forward node in T.

According to Lemma 8 non-trivial components consist exclusively of edges both of whose endpoints are forward nodes. Hence non-trivial components contain only nodes that are forward nodes in T. Lemma 4 shows that each isolated edge contains at least one forward node of T. We conclude that there are at least as many forward nodes in T as there are replacement edges, which, in turn, equals the number of critical edges. As any critical edge is incident to at most two unprocessed nodes, each unprocessed node can transfer its coin to a forward node so that no forward node gains more than two coins in Stage III.

Corollary 2. *No forward node in T gains more than two coins in Stage III.* □

Observation 1. *After Stage III, all coins from C' are transferred to some nodes of $F \cup H$.*

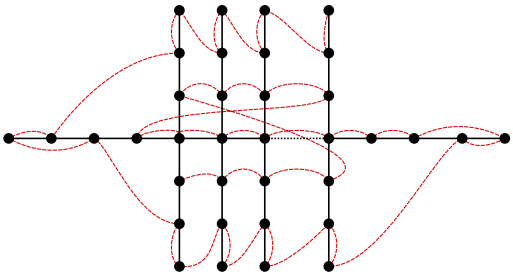
3.5 Transfer Scheme – Result

We now consider the result of our three-stage transfer scheme and summarize our findings. Observation 1 above established property (P1).

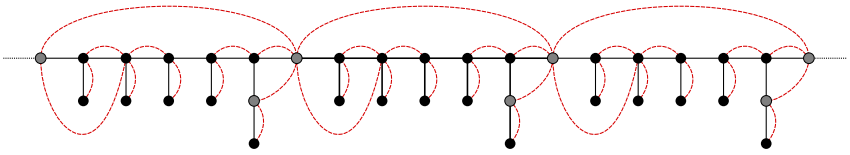
In Stage I we transferred some of the coins located at nodes in C' to nodes in $F \cup H$. Each node in $F \cup H$ gains at most $\deg(v)$ additional coins in Stage I. In particular, each node v in H owns at most $\deg(v)+1$ coins after Stage I and will not recieve further coins in the subsequent stages. This establishes property (P3).

In Stages II and III, coins are only transferred from nodes in C' to nodes in F. Since each forward node gains at most 2 coins in Stage I and, according to Corollaries 1 and 2, gains at most three further coins in Stages II and III we conclude that each forward node owns at most 5 coins after the execution of all three stages. This establishes property (P2).

To summarize, properties (P1)–(P3) hold after the execution of our transfer scheme. As described in Section 3.1, we can therefore distribute the $|C' \cup H|$



(a) The optimal solution is a Hamilton path. In general, we consider this graph to have k many vertical paths.



(b) The optimal solution has high-degree nodes. In general, we will repeat the central pattern k times; the figure shows 3 repetitions.

Fig. 1. Bad cases. The solid black edges form a locally optimal solution found by the algorithm, while the dashed red edges form the optimum solution. The graph itself consists of exactly the union of these two edge sets (disregarding multiplicity of edges).

coins in $C' \cup H$ among path nodes distinct from regular leaves such that each of them receives at most 5 coins. Hence there must exist at least $1/5 \cdot |C' \cup H|$ supplementary leaves or forward nodes. Together with the $|C' \cup H|$ regular leaves, we thus have $|P| \geq 6/5 \cdot |C' \cup H|$ path nodes overall. By means of Inequality (1), we can therefore derive our main result.

Theorem 1. *Our algorithm approximates the problem of finding a spanning tree with the largest possible number of non-branching nodes within a ratio of* $\frac{6}{11}$. □

4 Tightness of Approximation

It remains an open problem, whether the ratio proved above is tight. The following two examples show certain structural properties which may be interesting in the future hunt for an answer to this question.

Figure 1(a) shows an instance where the optimal solution is $|V(G)|$, i.e., the graph allows a Hamiltonian path (dashed red lines) and hence requires no branch nodes at all. Yet, our algorithm may terminate with a tree (solid black lines) containing multiple branch nodes. Observe the repetitive pattern (vertical structures) that may be repeated, say, k times. The depicted solution then has k branch nodes (of degree 4, rather than the degree 3 nodes presumably required for a tight example). Since the overall graph has $7k + 8$ nodes, this example would only suggest a ratio of (asymptotically) at least $6/7$.

Figure 1(b) shows an instance (in fact, a repetitive pattern) which requires branch nodes. Each pattern has 12 nodes, the figure shows three repetitions. A final instance would consist of k repetitions, with some constant size "caps" on the left and right end of the graph structure. The optimum solution requires 2 branch nodes and hence allows 10 path nodes per pattern, whereas the algorithm may terminate with only 7 path nodes (5 branch nodes) per pattern. Hence, this example would result in a ratio of (asymptotically) 7/10.

5 Conclusions

We proposed the *maximum path-node spanning tree* problem as a complementary formulation to the established but approximation-wise impractical *minimum branch-node spanning tree*. Both formulations have the very same practical applications. We showed a conceptually and implementation-wise simple algorithm to tackle this NP-hard problem. The paper's core lies in the proof that this algorithm in fact guarantees an approximation ratio of (at least) 6/11, beating the critical barrier of ratio 1/2. It remains open whether this bound is tight or if an even better ratio can be shown for this algorithm. It would also be interesting to investigate whether we can do better with k-flips for a constant $k > 2$.

References

1. Bhatia, R., Khuller, S., Pless, R., Sussmann, Y.J.: The full degree spanning tree problem. In: Proc. of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), pp. 864–865 (1999)
2. Cerulli, R., Gentili, M., Iossa, A.: Bounded-degree spanning tree problems: models and new algorithms. Computational Optimization and Applications 42(3), 353–370 (2009)
3. Fürer, M., Raghavachari, B.: Approximating the minimum degree spanning tree to within one from the optimal degree. In: Proc. of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA 1992), pp. 317–324 (1992)
4. Gargano, L., Hammar, M.: There are spanning spiders in dense graphs (and we know how to find them). In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 802–816. Springer, Heidelberg (2003)
5. Gargano, L., Hammar, M., Hell, P., Stacho, L., Vaccaro, U.: Spanning spiders and light-splitting switches. Discrete Mathematics 285(1-3), 83–95 (2004)
6. Gargano, L., Hell, P., Stacho, L., Vaccaro, U.: Spanning trees with bounded number of branch vertices. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 355–365. Springer, Heidelberg (2002)
7. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica 20(4), 374–387 (1998)
8. Khuller, S., Bhatia, R., Pless, R.: On local search and placement of meters in networks. In: Proc. of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), pp. 319–328 (2000)
9. Knauer, M., Spoerhase, J.: Better approximation algorithms for the maximum internal spanning tree problem. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 459–470. Springer, Heidelberg (2009)

10. Lu, H.I., Ravi, R.: The power of local optimization: Approximation algorithms for maximum-leaf spanning tree. In: Proceedings of the Thirtieth Annual Allerton Conference on Communication, Control and Computing, pp. 533–542 (1992)
11. Prieto, E., Sloper, C.: Reducing to independent set structure – the case of k-internal spanning tree. Nordic Journal of Computing 12(3), 308–318 (2005)
12. Salamon, G., Wiener, G.: On finding spanning trees with few leaves. Information Processing Letters 105, 164–169 (2008)
13. Salamon, G.: Approximating the maximum internal spanning tree problem. Theoretical Computer Science 410(50), 5273–5284 (2009)
14. Salamon, G.: Degree-Based Spanning Tree Optimization. PhD thesis, Department of Computer Science and Information Theory, Budapest University of Technology and Economics, Hungary (2010)
15. Silva, D.M., Silva, R.M.A., Mateus, G.R., Gonçalves, J.F., Resende, M.G.C., Festa, P.: An iterative refinement algorithm for the minimum branch vertices problem. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 421–433. Springer, Heidelberg (2011)
16. Solis-Oba, R.: 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 441–452. Springer, Heidelberg (1998)

On the Complexity of the Regenerator Location Problem - Treewidth and Other Parameters
(Extended Abstract)

Itamar Hartstein[1], Mordechai Shalom[2], and Shmuel Zaks[1]

[1] Department of Computer Science, Technion, Haifa, Israel
{hitamar,zaks}@cs.technion.ac.il
[2] TelHai College, Upper Galilee, 12210, Israel
cmshalom@telhai.ac.il

Abstract. We deal with the Regenerator Location Problem in optical networks. We are given a network $G = (V, E)$, and a set $\mathcal{Q}$ of communication requests between pairs of terminals in V. We investigate two variations: one in which we are given a routing $\mathcal{P}$ of the requests in $\mathcal{Q}$, and one in which we are required to find also the routing. In both cases, each path in $\mathcal{P}$ must contain a regenerator after every d edges in order to deal with loss of signal quality for some $d > 0$. The goal is to minimize the number of vertices that contain regenerators used by the solution. Both variations of the problem are NP-HARD in the general case. In this work we investigate the parameterized complexity of the problem. We introduce several fixed parameter tractability results and polynomial algorithms for fixed parameter values, as well as several NP-HARDNESS results. The parameters under consideration are the treewidth of the input graph, the sizes d and $|\mathcal{Q}|$ and the vertex load, i.e. the maximum number of paths passing through any vertex.

1 Introduction

1.1 Background

One of the main problems in optical network design is the deterioration in the signal quality after it travels a given distance d. To overcome this problem, regenerators are located at the vertices of the network. For two clients to be able to communicate in the network, the regenerators must be placed such that lightpaths can be formed between them such that there is at least one regenerator in every d consecutive vertices of each path. A regenerator can serve only one lightpath.

Regenerators are rather expensive equipment, and much research has been conducted, concerning minimizing their usage while satisfying all or most of the communication requirements posed by clients.

The cost of regenerators in a network is measured in two main ways:

- The number of regenerators placed in the network.
- The number of locations (that is, vertices) in which regenerators are placed.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 42–55, 2013.

In this article we deal with the second measure. The motivation behind the number of locations measure, is that a considerable part of the required cost of a regenerator is shared by other regenerators (e.g. by the use of common equipment or manpower).

We deal with two types of connectivity requirements. In the first, we are given paths between certain terminal vertices, and we need to find a placement of regenerators such that the given paths are satisfied. In the second, we are only given pairs of terminals, and we are required to find lightpaths which connect the given terminal pairs, and a regenerator placement minimizing the number of vertices hosting regenerators used by these lightpaths. We denote these problems as RLP_{path} and RLP_{req} , respectively.

1.2 Related Work

Regenerator placement has been extensively studied from an engineering point of view (see [2,3,13,15,16]). Such works usually offer mainly simulations and heuristics.

[2] contained the first theoretical result concerning regenerator placement, showing that the regenerator location problem with connection requests (i.e. without a given routing) is NP-HARD in the all to all case, i.e. when there is a request between every two vertices in the network. [4] was the first paper completely devoted to theoretical analysis of regenerator placement, offering exact algorithms, NP-Hardness results, approximation algorithms and hardness of approximation results for the regenerator *location* problem. [11] provides a first theoretical analysis of the *number of regenerators* objective. The articles [5,6,7] deal with minimizing the number of regenerators in optical networks with traffic grooming, i.e. networks which allow sharing of regenerators by at most g different paths with g being the *grooming factor*. Finally, [12] and [14] consider regenerator placement in an online setting.

1.3 Our Contribution

Both problems under consideration were shown to be NP-HARD in [4]. Our goal is to better understand the parameters making these problems hard, and to separate the polynomial cases, from the NP-HARD cases depending on these parameters, and to show fixed parameter tractability where possible.

We first show a result regarding ring networks. It was proven in [4] that RLP_{path} is polynomial-time solvable in tree and ring networks. The problems RLP_{path} and RLP_{req} are identical in trees, and therefore RLP_{req} is also polynomial-time solvable in this topology. We show that RLP_{req} is polynomial-time solvable in ring networks.

We consider the treewidth of the network as one main parameter. The treewidth $tw(G)$ of a network G is a measure for its structure, resembling the network's level of similarity to a tree. We show that: (1) RLP_{req} is NP-HARD for any fixed value of d and for any fixed value of $tw(G)$ at least 3, (2) RLP_{path} is fixed parameter tractable for $d = 2$ with the treewidth $tw(G)$ of the graph as

the parameter, and (3) RLP_{path} is NP-HARD for any fixed value $d \geq 5$ and for graphs of any fixed treewidth at least 2.

Next, we introduce the vertex load parameter for the problem RLP_{path} . The vertex load of a path set $\mathcal{P}$, $L_v(\mathcal{P})$, is the maximum number of paths which share the same common vertex. We show that RLP_{path} is polynomially solvable if both the treewidth and the vertex load are fixed.

Finally, we consider the number of connections that need to be made. We show that (1) RLP_{path} is APX-HARD for every two fixed values $|\mathcal{P}| \geq 2, d \geq 2$, and (2) RLP_{req} is fixed parameter tractable for $d = 1$ and $d = 2$.[1]

The results are summarized in Table 1.

Table 1. Summary of results

RLP_{path} $(G, d, \mathcal{P})$	RLP_{req} $(G, d, \mathcal{Q})$
NP-HARD when $tw(G) \geq 2$ and $d \geq 5$	NP-HARD when $tw(G) \leq 3$ and $d \geq 1$.
FPT in $tw(G)$ when $d = 2$	
Polynomial when $tw(G)$ and $L_v(\mathcal{P})$ are constants for any d	
APX-HARD when $\|\mathcal{P}\| \geq 2, d \geq 2$	FPT in $\|Q\|$ for $d = 1$

The structure of the article is as follows. In Section 2 we give definitions of the terms used in this paper. In Section 3 we discuss RLP_{req} on rings. In Section 4 we prove a hardness result concerning RLP_{req} for graphs of treewidth at most 3. In Section 5 we provide results concerning RLP_{path} on bounded treewidth and bounded vertex load (see Table 1). In Section 6 we deal with instances of RLP_{path} and RLP_{req} with limited number of paths and requests, respectively. Finally, in Section 7 we conclude by presenting several open problems and possible extensions to this work.

Due to space limitations, some proofs and figures have been removed and can be found in our full paper [10].

2 Preliminaries

Given an undirected underlying graph $G = (V(G), E(G))$ that corresponds to the network topology, a *lightpath* is a simple path in G. $\mathcal{P} = \{\pi_1, \pi_2, \ldots, \pi_n\}$ is a set of paths in G representing the lightpaths. $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\}$ is a set of communication requests between the vertices of G, i.e. $q_i = \{s_i, t_i\}$ for some $s_i, t_i \in V$, for every $1 \leq i \leq n$. Given a subset $U \subseteq V(G)$, $G[U]$ denotes the subgraph of G induced by U. For any subset $Q \subseteq \mathcal{Q}$ of requests, $term(\mathcal{Q}) \stackrel{def}{=} \cup\mathcal{Q}$ denotes the set of terminals of $\mathcal{Q}$. The *length* $\ell(\pi)$ of a lightpath π is the number of its edges. The *internal vertices* (resp. *edges*) of a path π are the vertices (resp.

[1] All the hardness results presented in this work actually consider the minimum values of the mentioned parameters, (i.e. $d, tw(G)$). However the generalization to higher values is simple.

edges) in π except the first and the last ones. Given a set $\mathcal{P}$ of paths, a vertex $v \in V$ and an edge $e \in E$, $\mathcal{P}_v$ denotes the set of paths of $\mathcal{P}$ traversing v (i.e. having v as internal vertex), and $\mathcal{P}_e$ is the set of paths of $\mathcal{P}$ containing e. The load induced by $\mathcal{P}$ on e and v are $L(\mathcal{P}, v) \stackrel{def}{=} |\mathcal{P}_v|$ and $L(\mathcal{P}, e) \stackrel{def}{=} |\mathcal{P}_e|$. Finally $L_v(\mathcal{P}) \stackrel{def}{=} \max\{L(\mathcal{P}, v)|v \in V\}$ and $L(\mathcal{P}) \stackrel{def}{=} \max\{L(\mathcal{P}, e)|e \in E\}$.

Given two vertices $u, v \in V$, we define $dist(u, v)$ as the distance of a shortest path between u and v in G. If u and v are vertices of a common path $\pi \in \mathcal{P}$, we define $dist_\pi(u, v)$ as the number of edges between u and v on π.

A routing of $\mathcal{Q}$ is a set of paths $\mathcal{P}$ such that for every request $q_i = \{s_i, t_i\} \in \mathcal{Q}$, there is a path $\pi_i \in \mathcal{P}$ between s_i and t_i. A regenerator location assignment is a set $R \subseteq V$.

Given an integer d, a lightpath π is *d-satisfied* by a set of regenerator locations R if for any d consecutive internal vertices $v_1, v_2, \cdots, v_d$ of π, $v_i \in R$ for some $1 \leq i \leq d$. When there is no ambiguity about the set of regenerator locations under consideration we simply say that π is *d-satisfied*.

A set of lightpaths is *d-satisfied* if every lightpath in it is d-satisfied. Note that a path with at most d edges is d-satisfied regardless of R, therefore we assume without loss of generality that every path $\pi \in \mathcal{P}$ has at least $d + 1$ edges. We assume, without loss of generality, that every edge of the graph is used by at least one path $\pi \in \mathcal{P}$.

We consider two variants of the Regenerator Location Problem: (a) given a graph $G = (V, E)$, a set $\mathcal{P}$ of paths in G, and a distance $d \geq 1$, find a regenerator location assignment $R \subseteq V$ of minimum cardinality such that all the paths in $\mathcal{P}$ are d-satisfied. (b) given a graph $G = (V, E)$, a set $\mathcal{Q}$ of requests in G, and a distance $d \geq 1$ find a routing $\mathcal{P}$ of $\mathcal{Q}$ and a regenerator location assignment $R \subseteq V$ of minimum cardinality such that all the paths in $\mathcal{P}$ are d-satisfied. Formally:

REGENERATOR LOCATION PROBLEM(RLP_{path})

Input: An undirected graph $G = (V, E)$, a set $\mathcal{P}$ of paths in G, $d \geq 1$
Output: A regenerator location assignment $R \subseteq V$ such that every path $P \in \mathcal{P}$ is d-satisfied.
Objective: Minimize $|R|$.

ROUTING AND REGENERATOR LOCATION PROBLEM(RLP_{req})

Input: An undirected graph $G = (V, E)$, a set $\mathcal{Q}$ of requests in G, $d \geq 1$
Output: A routing $\mathcal{P}$ of $\mathcal{Q}$, and a regenerator location assignment $R \subseteq V$ such that every path $P \in \mathcal{P}$ is d-satisfied.
Objective: Minimize $|R|$.

We assume the reader is familiar with the notions of treewidth, parameterized complexity, fixed parameter tractability (FPT), polynomial-time approximation scheme (PTAS), and APX-HARD. Nevertheless, definitions of these terms can be found in our full paper [10].

Two Fundamental Problems. We mention here two fundamental problems used in this work: the 3SAT problem and the Vertex Cover problem.

An instance (X, ϕ) of 3SAT (or any of its variants that we use in this work) is a set $\phi = \{\phi_i |\ 1 \leq i \leq m\}$ of clauses over a set of boolean variables $X = \{x_j |\ 1 \leq j \leq n\}$. $\bar{X} = \{\bar{x_j} |\ 1 \leq j \leq n\}$ is the set of all negative literals over X. Each clause $\phi_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ is a conjunction of 3 literals from $X \cup \bar{X}$. The output for such an instance is a satisfying assignment $f : X \mapsto \{0, 1\}$ such that all the clauses ϕ_i of ϕ are satisfied, i.e. at for least one literal $\ell_{i,k} (k \in \{1, 2, 3\})$ either $\ell_{i,k} = x_j \wedge f(x_j) = 1$ for some $1 \leq j \leq n$ or $\ell_{i,k} = \bar{x_j} \wedge f(x_j) = 0$ for some $1 \leq j \leq n$.

Given an undirected graph $G = (V, E)$, $C \subseteq V$ is a vertex cover of G if for every $\{u, v\} \in E$, either $\{u, v\} \cap C \neq \emptyset$. The problem MINVERTEXCOVER is the problem of finding a vertex cover of minimum cardinality.

3 RLP_{req} in Rings

It was proven in [4] that RLP_{path} is polynomially solvable on rings. Note, that given a ring $G = (V, E)$ and a request set $\mathcal{Q}$, there are exactly two routings for every $q \in \mathcal{Q}$. Therefore, given a request set $\mathcal{Q}$, by considering all the possible routings of $\mathcal{Q}$, and finding the optimal solution for RLP_{path} for each one of them, we can optimally solve RLP_{req} $(G, d, \mathcal{Q})$ for every given positive integer d. Unfortunately, there are $2^{|\mathcal{Q}|}$ such possible routings. We show that it is actually sufficient to consider only $O(|V|^3)$ routings in order to find an optimal solution. Namely we prove:

Theorem 1. *RLP_{req} $(G, d, \mathcal{Q})$ can be solved in polynomial time.*

We assume an arbitrary planar embedding of the graph G, such that for every $u, v \in V$ there is one *clockwise* path and one *counterclockwise* path which connects u and v. We denote by $\rho(u, v)$ the clockwise path from u to v, and by $\ell(\rho(u, v))$ the length of $\rho(u, v)$. For convenience, we will abuse notation and also refer to $\rho(u, v)$ as the set of its vertices. Finally, given a solution R, and two regenerator locations $r_1, r_2 \in R$, we say that r_1 and r_2 are *consecutive in* R if $\rho(r_1, r_2) \cap R = \{r_1, r_2\}$.

According to [4], the problem RLP_{path} is polynomially solvable on rings. Let ALGRLPRTRINGS be an algorithm which accepts an instance $(G, d, \mathcal{P})$ of RLP_{path} , and obtains an optimal regenerator placement R d-satisfying $\mathcal{P}$.

Denote the vertices of V as $v_0, v_1, \ldots, v_{n-1}$ such that $v_0, v_1, \ldots, v_{n-1}$ are consecutive vertices in clockwise direction. The algorithm below solves optimally RLP_{req} $(G, d, \mathcal{Q})$ when G is a ring. The correctness of the algorithm is implied by the proof of Theorem 1 in [10].

Algorithm 1. ALGRLPREQRINGS $(G, d, \mathcal{Q})$

$R_D \leftarrow \{v_i \mid 0 \leq i \leq n-1 \wedge i \equiv 0 \pmod d\}$ ▷ $\{v_0, v_d, v_{2d}, ...\}$
$\mathcal{P}_D \leftarrow \{\rho(v_i, v_j) \mid \{v_i, v_j\} \in \mathcal{Q} \wedge i < j\}$
$Sol \leftarrow \{(\mathcal{P}_D, R_D)\}$

for $r_1, r_2 \in V$ s.t $\ell(\rho(r_1, r_2)) > d$ **do**
    Let $\mathcal{Q}' = \{\{s_1, t_1\}, \{s_2, t_2\}, ..., \{s_m, t_m\}\}$ be the set of requests in $\mathcal{Q}$ such that:
        – For every $1 \leq i \leq m$: $s_i \in \rho(r_1, r_2)$, $t_i \in \rho(r_2, r_1)$, $\ell(\rho(r_1, s_i)) \leq d$ and $\ell(\rho(s_i, r_2)) \leq d$
        – $\ell(\rho(r_2, t_1)) < \ell(\rho(r_2, t_2)) < ... < \ell(\rho(r_2, t_m))$.
    $\hat{\mathcal{P}} \leftarrow \emptyset$
    for $\{s, t\} \in \mathcal{Q} \setminus \mathcal{Q}'$ **do**
        if $\rho(s, t) \subseteq \rho(r_1, r_2)$[2] **or** $\rho(r_1, r_2) \subseteq \rho(s, t)$ **or** $s \in \rho(r_1, r_2) \wedge t \in \rho(r_2, r_1) \wedge \ell(\rho(s_i, r_2)) > d$ **then**
            $\hat{\mathcal{P}} \leftarrow \hat{\mathcal{P}} \cup \{\rho(t, s)\}$
        else
            $\hat{\mathcal{P}} \leftarrow \hat{\mathcal{P}} \cup \{\rho(s, t)\}$
    for $1 \leq i \leq m$ **do**
        $\mathcal{P}_{r_1, r_2, t_i} \leftarrow \hat{\mathcal{P}} \cup \{\rho(s_j, t_j) | j \leq i\} \cup \{\rho(t_j, s_j) | j > i\}$
        $R_{r_1, r_2, t_i} \leftarrow ALGRLPRTRINGS(G, d, \mathcal{P}_{r_1, r_2, t_i})$
        $Sol \leftarrow Sol \cup \{(\mathcal{P}_{r_1, r_2, t_i}, R_{r_1, r_2, t_i})\}$

return $\underset{(\mathcal{P}, R) \in Sol}{\operatorname{argmin}} |R|$

4 Hardness of RLP_{req} in Bounded Treewidth Graphs

In this section we show that the problem RLP_{req} $(G, d, \mathcal{Q})$ is hard even when $tw(G) = 3$ and $d = 1$. The proof follows a technique used in [1] to show that the Steiner Forest problem is hard even for graphs of treewidth 3.

First, we define the problem R-SAT. Given three boolean variables a_1, a_2, a_3, an *R-clause* $R(a_1, a_2, a_3)$ stands for $(a_1 = a_3) \vee (a_2 = a_3)$. Given a set of boolean variables $X = \{x_1, ..., x_n\}$, we say that a formula ϕ is an *R-formula* over X if $R = R_1 \wedge R_2 \wedge ... \wedge R_m$ where m is a non-negative integer and for every $1 \leq i \leq m$, R_i is an R-clause composed of literals over $X \cup \{0, 1\}$. We assume that for every $1 \leq j \leq n$, x_j appears in some clause of ϕ. For example, the R-formula $R(x, y, 1) \wedge R(x, 0, z)$ stands for $((x = 1) \vee (y = 1)) \wedge ((x = z) \vee (0 = z))$. The problem of deciding whether a given R-formula ϕ is satisfiable is called *R-SAT* and is proven in [1] to be NP-COMPLETE.

A formula ϕ is non-trivial if it contains both constants 0 and 1. The problem *Non-trivial R-SAT* is the problem of deciding whether a non-trivial R-SAT formula is satisfiable. It is easy to see that non-trivial R-SAT is also NP-HARD by reduction from regular R-SAT. Indeed, any R-SAT formula ϕ can be transformed into an equivalent non-trivial R-formula $\phi' = \phi \wedge R(0, 1, x)$, where $x \in X$.

[2] Recall that we assume every path contains at least $d + 1$ edges!

In [10] we prove the main result of this section by a reduction from the non-trivial R-SAT problem.

Theorem 2. *RLP_{req} $(G, d, \mathcal{Q})$ is* NP-HARD *even when $tw(G) \leq 3$ and $d = 1$.*

5 RLP_{path} with Treewidth and Vertex Load Parameters

We first show that the problem RLP_{path} is FPT for $d = 2$ when parameterized by the treewidth of the input graph (note that RLP_{path} is trivial when $d = 1$ even for general graphs).

Theorem 3. *RLP_{path} $(G, 2, \mathcal{P})$ is* FPT *when parameterized by $tw(G)$.*

By the above theorem, RLP_{path} is polynomial time solvable for any fixed treewidth when $d = 2$. We next show that this does not hold for every value of d. Specifically, we show that:

Theorem 4. *RLP_{path} $(G, 5, \mathcal{P})$ is* NP-HARD*, even when $tw(G) = 2$.*

We have seen that when the treewidth of the input graph is fixed, the problem RLP_{path} is NP-HARD. By Corollary 61 the problem is also NP-HARD when the vertex load of the path set is fixed. We now show that the problem is solvable in polynomial time when *both* the treewidth and the vertex load are fixed.

Theorem 5. *An optimal solution for RLP_{path} $(G, d, \mathcal{P})$ can be computed in polynomial time, if both $tw(G)$ and $L_v(\mathcal{P})$ are fixed.*

We start by introducing definitions used in the algorithm and its analysis, and then present an optimal algorithm for $RLP_{path}(G, d, \mathcal{P})$. In [10] we prove the correctness of the algorithm, and show that it returns an optimal solution in polynomial time when $tw(G) \leq w$ and $L_v(\mathcal{P}) \leq \ell$ for every two constants ω and ℓ.

Definitions and Basic Observations. In this section we assume that every path $\pi \in \mathcal{P}$ is assigned an arbitrary and fixed direction, designating one endpoint of π as its *left* endpoint, denoted as $left(\pi)$, and the other as its *right* endpoint. Given a path π, and two vertices $u, v \in \pi$, we say that u is on the left of v in π if u is in the sub-path of π connecting $left(\pi)$ and v, and denote this by $u \leq_\pi v$. If $u \leq_\pi v$ and u is adjacent to v in π, we say that u is the *left neighbor* of v in π and denote u as $left_\pi(v)$.

Given a set $R \subseteq V$ of regenerator locations, the function a_R indicates for each path and each internal node of it, the distance of the closest regenerator location when moving towards the left of the path. Formally, for a set $U \subseteq V$ we define the set $\mathcal{D}_U \stackrel{def}{=} \bigcup_{u \in U}(\{u\} \times \mathcal{P}_u)$, and $a_R : \mathcal{D}_V \mapsto \mathbb{N}$ is an integer function over $\mathcal{D}_U$, such that:

$$a_R(v, \pi) = \begin{cases} 0 & \text{if } v \in R \\ 1 & \text{if } v \notin R \text{ and } left_\pi(v) = left(\pi) \\ a_R(left_\pi(v), \pi) + 1 & \text{otherwise} \end{cases}$$

Clearly, if R d-satisfies $\mathcal{P}$, then $a_R(v,\pi) \in \mathbb{Z}_d = \{0,1,\ldots,d-1\}$. We denote by $\mathcal{F}_U$ the set of all functions from $\mathcal{D}_U$ into $\mathbb{Z}_d$, i.e. $\mathcal{F}_U \stackrel{def}{=} \mathbb{Z}_d^{\mathcal{D}_U}$. Clearly, $|\mathcal{D}_U| = \sum_{u\in U} |\mathcal{P}_u| \leq \ell \cdot |U|$, thus $|\mathcal{F}_U| \leq d^{\ell\cdot|U|}$.

A function $f \in \mathcal{F}_V$ is a *legal* d-assignment if $f = a_R$ for some $R \subseteq V$ d-satisfying $\mathcal{P}$. Define for every $f \in \mathcal{F}_V$: $R_f \stackrel{def}{=} \left\{v \in V \middle|\ \sum_{\pi\in\mathcal{P}_v} f(v,\pi) = 0\right\}$.

Claim. $f \in \mathcal{F}_V$ is a legal d-assignment if and only if for every $(v,\pi) \in \mathcal{D}_V$

$$f(v,\pi) = 0 \Rightarrow \forall \pi' \in \mathcal{P}_v, f(v,\pi') = 0 \tag{1}$$

$$left_\pi(v) = left(\pi) \Rightarrow f(v,\pi) \in \{0,1\} \tag{2}$$

$$left_\pi(v) \neq left(\pi) \Rightarrow f(v,\pi) \in \{0, f(left_\pi(v),\pi) + 1\} \tag{3}$$

If f is a legal d-assignment such that $f = a_R$, then clearly $R = R_f$.

Corollary 51. *Given a function $f \in \mathcal{F}_V$*

- *It can be tested in polynomial time whether f is a legal d-assignment.*
- *R_f can be computed in polynomial time.*

Given a $f \in \mathcal{F}_V$ and $U \subseteq V$, the *restriction* $f\big|_U$ of f to U is the function $f\big|_U \in \mathcal{F}_U$ where $f\big|_U(v,\pi) = f(v,\pi)$ for every $(v,\pi) \in \mathcal{D}_U$. We say that $f, f' \in \mathcal{F}_V$ *agree* on $U \subseteq V$ (and denote by $f \cong_U f'$) if $f\big|_U = f'\big|_U$. When $f \in \mathcal{F}_U$ and $f' \in \mathcal{F}_{U'}$ agree on $U \cap U'$ we say simply that f and f' agree and denote this by $f \cong f'$. When these conditions do not hold, we write $f \ncong_U f'$ and $f \ncong f'$.

$g \in \mathcal{F}_U$ is a *partial legal* d-assignment over $U \subset V$ if $g = f\big|_U$ for some legal d-assignment $f \in \mathcal{F}_V$. We denote by $\mathcal{L}_U$ the set of all partial legal d-assignments over U. An optimal partial legal d-assignment over $U \subseteq V$ is a function $f^* \in \mathcal{L}_U$ minimizing $|R_{f^*}|$, i.e. $|R_{f^*}| = \min_{f\in\mathcal{L}_U} |R_f|$.

Claim. Let $U \subseteq V$ and $f \in \mathcal{F}_U$. $f \in \mathcal{L}_U$ if and only if the following conditions hold for every $u, v \in U$:

$$f(v,\pi) = 0 \Rightarrow \forall \pi' \in \mathcal{P}_v, f(v,\pi') = 0 \tag{4}$$

$$f(v,\pi) \leq dist_\pi(left(\pi), v) \tag{5}$$

$$u \leq_\pi v \Rightarrow f(v,\pi) < dist_\pi(u,v) \vee f(v,\pi) = f(u,\pi) + dist_\pi(u,v) \tag{6}$$

Corollary 52. *Given a function $f \in \mathcal{F}_U$*

- *it can be verified in polynomial time whether $f \in \mathcal{L}_U$,*
- *R_f can be computed in polynomial time.*

Claim. Let $f \in \mathcal{F}_U$ and $u \leq_\pi v \leq_\pi w$. If both pairs u, v and v, w satisfy (6), then the pair u, w satisfies (6) too.

Proof. If $f(w,\pi) < dist_\pi(v,w)$ then clearly $f(w,\pi) < dist_\pi(u,w)$ and the condition holds for u, w. Otherwise $f(w,\pi) = f(v,\pi) + dist_\pi(v,w)$. If $f(v,\pi) < dist_\pi(u,v)$ then $f(w,\pi) = f(v,\pi) + dist_\pi(v,w) < dist_\pi(u,v) + dist_\pi(v,w) = dist_\pi(u,v)$ and the condition holds for u, v, otherwise $f(v,\pi) = f(u,\pi) + dist_\pi(u,v)$, therefore $f(w,\pi) = f(u,\pi) + dist_\pi(u,v) + dist_\pi(v,w) = f(u,\pi) + dist_\pi(u,w)$ and the condition holds.

Given $U, U' \subseteq V$, $f \in \mathcal{L}_U$, $f' \in \mathcal{L}_{U'}$, such that $f \cong f'$, recall that their union $f \cup f'$ is a function $g \in \mathcal{F}_{U \cup U'}$ such that

$$g(v,\pi) = \begin{cases} f(v,\pi) & \text{if } v \in U \\ f'(v,\pi) & \text{otherwise.} \end{cases}$$

Claim. If $f \in \mathcal{L}_U$ and $f' \in \mathcal{L}_{U'}$, then $g = f \cup f'$ satisfies conditions (4) and (5).

Proof. In order to show that (4) is satisfied, let $g(v,\pi) = 0$ and assume by contradiction that there is some $\pi' \in \mathcal{P}_v$ such that $g(v,\pi') \neq 0$. Assume without loss of generality that $v \in U$. Then $f(v,\pi) = g(v,\pi) = 0$ and $f(v,\pi') = g(v,\pi') \neq 0$, contradicting the assumption that $f \in \mathcal{L}_U$, and, in particular, that it satisfies (4). In order to show that (5) is satisfied, let $(v,\pi) \in \mathcal{D}_{U \cup U'}$. If $v \in U$, then $g(v,\pi) = f(v,\pi) < dist_\pi(left(\pi), v)$ since f satisfies (5). Otherwise, $g(v,\pi) = f'(v,\pi) < dist_\pi(left(\pi), v)$ since f' satisfies (5). Therefore (5) is also satisfied in any case.

Note that $f \cup f'$ is not necessarily a partial legal d-assignment as it might not satisfy (6).

Observation 51. *Let $U \subseteq V$, and f, f' be partial legal d-assignments over two sets that both include U, and $f \cong f'$. Then*

$$R_{f \cup f'} = R_f \cup R_{f'}$$

$$R_{f\big|_U} = R_f \cap U.$$

We end up this section by stating a claim about tree decompositions.

Claim. Let B_i and B_k two adjacent bags in a tree decomposition $\mathcal{T}$ of a graph $G = (V, E)$, and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the subtrees obtained by the removal of the edge $\{B_i, B_k\}$ from $\mathcal{T}$. Let $V_i \stackrel{def}{=} \cup \mathcal{T}_i$ be the set of all nodes that reside in some bag of $\mathcal{T}_i$. Let $v_1 \in \mathcal{T}_1 \setminus \mathcal{T}_2$ and $v_2 \in \mathcal{T}_2 \setminus \mathcal{T}_1$ two non-adjacent nodes of G. Then every path π connecting v_1 and v_2 contains at least one node $w \in B_i \cap B_k$.

A Dynamic Programming Algorithm for RLP_{path} . We now present a polynomial time dynamic programming algorithm solving optimally RLP_{path} $(G, d, \mathcal{P})$.

The algorithm works in three phases. In the initialization phase, we compute an optimal tree decomposition $\mathcal{T} = (\mathcal{B}, F)$ of G, and for every bag $B_i \in \mathcal{B}$ we set A_i to be the set of all possible partial legal d-assignments over B_i. During the dynamic programming phase, we update A_i such that eventually it contains a set of legal d-assignments over V_i, where V_i is the set of vertices appearing in some bag in the subtree of $\mathcal{T}$ rooted at B_i (when $\mathcal{T}$, without loss of generality, is assumed to be rooted at B_1). One of these, as we will see, is a minimum-size d-assignment over V_i. In the final phase, we obtain the minimum size assignment in A_1, f_{min}, and declare $R_{f_{min}}$ as an optimal solution for RLP_{path} $(G, d, \mathcal{P})$.

We prove in [10] three lemmata showing that the algorithm returns a feasible solution, runs in polynomial time and is optimal, respectively.

Algorithm 2. ALGRLP $(G, d, \mathcal{P})$

Phase 1: Initialization

Direct arbitrarily the paths in π.
Compute a tree decomposition of G of width $tw(G)$, $\mathcal{T} = (\mathcal{B}, F)$, and direct $\mathcal{T}$ such that B_1 is its root.
Compute $\mathcal{L}_{B_i}$ for every $B_i \in \mathcal{B}$.

Phase 2: Dynamic Programming

for every bag B_i, traversed in postorder fashion **do**
    $A_i \leftarrow \emptyset$
    for every $\hat{f} \in \mathcal{L}_{B_i}$ **do**
        Denote $Ch_i = \{B_{k_1}, B_{k_2}, \ldots, B_{k_t}\}$ as the set of children of B_i
        for every $1 \leq j \leq t$ **do**
            $M \leftarrow \left\{ f' \in A_{k_j} \mid f' \cong \hat{f} \right\}$
            $f_{k_j} \leftarrow \underset{f' \in M}{\operatorname{argmin}} |R_{f'}|$
        $\alpha_i(\hat{f}) \leftarrow \hat{f} \cup \bigcup_{j=1}^{t} f_{k_j}$
        $A_i \leftarrow A_i \cup \left\{ \alpha_i(\hat{f}) \right\}$

Phase 3: Obtain an Optimal Solution

$f_{min} \leftarrow \underset{f' \in A_1}{\operatorname{argmin}} |R_{f'}|$
return $R_{f_{min}}$

6 Fixed Number of Connections

In this section we consider another important parameter relevant for both problems RLP_{path} and RLP_{req} , namely the number of connection requests. More precisely the parameter under consideration is the number $|\mathcal{P}|$ of paths for RLP_{path} , and the number $|\mathcal{Q}|$ of requests for RLP_{req} .

We first show that RLP_{path} is NP-HARD even when $d = |\mathcal{P}| = 2$. To see that this result draws an exact hardness boundary, we note that when $d = 1$ or $|\mathcal{P}| = 1$, RLP_{path} can be solved in linear time. Indeed when $d = 1$, every internal vertex of a path in $\mathcal{P}$ must host a regenerator, and this constitutes a feasible solution, therefore optimal. When $|\mathcal{P}| = 1$, let $\mathcal{P} = \{\pi\}$. Clearly any solution contains at least $\lceil \frac{|\pi|}{d} \rceil$ locations, and such a solution can be easily found by letting R be the vertices $d, 2d, ...$ of π.

In fact, we prove a stronger result: we show that RLP_{path} does not admit a polynomial-time approximation scheme unless $P = NP$.

Theorem 6. *RLP_{path} $(G, d, \mathcal{P})$ is* APX-HARD *when $|\mathcal{P}| = 2$ and $d = 2$.*

We note that for every path set $\mathcal{P}$, $L_v(\mathcal{P}) \leq |\mathcal{P}|$. Therefore the following corollary follows immediately.

Corollary 61. *RLP_{path} $(G, d, \mathcal{P})$ is* APX-HARD *when $L_v(\mathcal{P}) \leq 2$ and $d = 2$.*

We next show that RLP_{req} is FPT in the number $|\mathcal{Q}|$ of connection requests, when $d = 1$. We start by introducing two problems.

MINIMUM DIRECTED STEINER TREE PROBLEM(DST)

Input: An instance (G, w, S, r) where $G = (V, A)$ is a digraph with a non-negative weight function $w : E \mapsto \mathbb{R}^+$ on its arcs, $S \subseteq V$ is a set of *terminal* vertices, and $r \in V$ is the root vertex.
Output: A subgraph T of G such that: (a) T is a directed tree rooted at r, (b) $S \subseteq V(T)$.
Objective: Minimize $w(T) \stackrel{def}{=} \sum_{e \in E(T)} w(e)$.

Given a graph $G = (V, E)$ and a subset $S \subseteq V$ of its vertices, a subset $D \subseteq V$ is said to *dominate* S in G if every vertex s of S is either in D or adjacent to a vertex of D.

MINIMUM STEINER CONNECTED DOMINATING SET PROBLEM($SCDS$)

Input: An instance (G, w, S) where $G = (V, E)$ is a graph with a non-negative weight function $w : V \mapsto \mathbb{R}^+$ on its vertices, and $S \subseteq V$ is a subset of vertices to be dominated.
Output: A set of vertices $D \subseteq V$ such that: (a) $G[D]$ is connected, (b) D dominates S in G.
Objective: Minimize $w(D) \stackrel{def}{=} \sum_{w \in D} w(v)$.

The $SCDS$ problem is a generalization of the well-known problem of finding a minimum connected dominating set. It was first defined in [8] in the context of approximation algorithms.

We denote by $OPT_{DST}(G, w, S, r)$ (resp. $OPT_{SCDS}(G, w, S)$, $OPT_{RLP_{req}}(G, d, \mathcal{Q})$) the optimum value of the instance (G, w, S, r) (resp. (G, w, S), $(G, d, \mathcal{Q})$) of problem DST(resp. $SCDS$, RLP_{req}).

In [9], it is noted that DST is FPT in the number of terminals. We will use this result to show that (a) $SCDS$ is FPT in the size of the dominated set, and (b) RLP_{req} is FPT in the number $|\mathcal{Q}|$ of requests when $d = 1$.

Lemma 1. *$SCDS$ is* FPT *in the size of the dominated set.*

Proof. By reduction to DST. Let (G, w, S) be an instance of $SCDS$. We build a digraph $G' = (V', A')$ where

$$\begin{aligned} V' &= \{v_0, v_1 | v \in V\} \\ A' &= \{(v_0, v_1) | v \in V\} \cup \{(u_1, v_0), (v_1, u_0) | \{u, v\} \in E\}. \end{aligned} \tag{7}$$

The weight function w' on the arcs of A' is

$$\begin{aligned} w'(u_0, u_1) &= w(u) \qquad\qquad \forall u \in V \\ w'(u_1, v_0) &= w'(v_1, u_0) = 0 \; \forall \{u, v\} \in E \end{aligned}$$

Let $S_0 \stackrel{def}{=} \{v_0 | v \in S\}$. In [10] we prove the following claim implying an FPT algorithm for DST:

Claim. $OPT_{SCDS}(G, w, S) = \min_{r \in V} OPT_{DST}(G', w', S_0, r_0)$. Moreover, given a solution T of (G', w', S_0, r_0) a solution D of (G, w, S) with $w(D) = w'(T)$ can be calculated in polynomial time.

The lemma concludes easily from the claim: Indeed, let $ALGDST(G', w', S_0, r_0)$ be an FPT algorithm calculating an optimal solution in time $f(|S_0|) \cdot p(n)$. Algorithm 3 calculates an optimal solution D^* of (G, w, S) in time $|V| \cdot f(|S_0|) \cdot p(q_2(n)) + q_1(n) + q_3(n) = |V| \cdot f(|S|) \cdot p(q_2(n)) + q_1(n) + q_3(n) \leq f(|S|) \cdot (n \cdot p(q_2(n)) + q_1(n) + q_3(n))$, therefore is an FPT algorithm for DST.

Algorithm 3. ALGSCDS (G, w, S)

1: Build G', w' and S_0 from G, w, S. ▷ running time $q_1(n)$
2: $r_0 \leftarrow \underset{r_0 \in V}{\operatorname{argmin}}\ w(ALGDST(G', w', S_0, r_0))$. ▷ running time $|V| \cdot f(S_0) \cdot p(q_2(n))$
3: $T^* \leftarrow ALGDST(G', w', S_0, r_0)$.
4: Calculate D^* from T^*. ▷ running time $q_3(n)$
5: **return** D^*.

Theorem 7. *RLP_{req} is* FPT *in the number $|\mathcal{Q}|$ of the requests when $d = 1$.*

Proof. Consider an instance $(G, 1, \mathcal{Q})$ of RLP_{req} . Without loss of generality, we assume that $\mathcal{Q}$ does not contain edges of G, i.e., for every $\{s_i, t_i\} \in \mathcal{Q}$, $\{s_i, t_i\} \notin E$, because otherwise this request is d-satisfied in every solution routing it through this edge. We denote by $\Omega_{\mathcal{Q}}$ the set of all partitions of the request set $\mathcal{Q}$.

In [10] we prove the following claim implying an FPT algorithm for RLP_{req} :

Claim. $OPT_{RLP_{req}}(G, 1, \mathcal{Q}) = \min_{\Pi \in \Omega_{\mathcal{Q}}} \sum_{\mathcal{Q}_i \in \Pi} OPT_{SCDS}(G, \mathbf{1}, term(\mathcal{Q}_i))$. [3] Moreover, given optimal solutions D_i of instances $(G, \mathbf{1}, term(\mathcal{Q}_i))$ for every $\mathcal{Q}_i \in \Pi$, a solution $\mathcal{P}, R$ of $(G, 1, \mathcal{Q})$ with $|R| = \sum w(D_i) = \sum |D_i|$ can be calculated in polynomial time.

The theorem concludes easily from the claim: Indeed, let $ALGSCDS(G, \mathbf{1}, term(\mathcal{Q}_i))$ be an FPT algorithm calculating an optimal solution in time $f(|term(\mathcal{Q}_i)|) \cdot p(n)$. Algorithm 4 calculates an optimal solution R^* of $(G, 1, \mathcal{Q})$. The dominant term of its running time is $\sum_{\Pi \in \Omega_{\mathcal{Q}}} \sum_{\mathcal{Q}_i \in \Pi} (f(|term(\mathcal{Q}_i)|) \cdot p(n)) \leq \sum_{\Pi \in \Omega_{\mathcal{Q}}} \sum_{\mathcal{Q}_i \in \Pi} (f(2\,|\mathcal{Q}|) \cdot p(n)) \leq |\Omega_{\mathcal{Q}}| \cdot |\mathcal{Q}| \cdot f(2\,|\mathcal{Q}|) \cdot p(n)$. As the first three factors depend solely on $|\mathcal{Q}|$, RLP_{req} is FPT in $|\mathcal{Q}|$ when $d = 1$.

7 Conclusion and Open Problems

We have considered the role of several parameters in the RLP_{path} and RLP_{req} problems. For RLP_{path} we considered the treewidth, the vertex load,

[3] $\mathbf{1}$ denotes a constant weight function with value 1 on the vertices of V.

Algorithm 4. ALGRLPREQ $(G, 1, \mathcal{Q})$

$\Pi \leftarrow \underset{\Pi \in \Omega_{\mathcal{Q}}}{\operatorname{argmin}} \sum_{\mathcal{Q}_i \in \Pi} |ALGSCDS(G, \mathbf{1}, term(\mathcal{Q}_i))|.$ ▷ running time $\sum_{\Pi \in \Omega_{\mathcal{Q}}} \sum_{\mathcal{Q}_i \in \Pi} (f(|term(\mathcal{Q}_i)|) \cdot p(n))$
for $\mathcal{Q}_i \in \Pi$ **do**
  $R_i \leftarrow ALGSCDS(G, \mathbf{1}, term(\mathcal{Q}_i))$
  for $(s,t) \in \mathcal{Q}_i$ **do**
    $s' \leftarrow$ an arbitrary neighbor of s in R_i
    $t' \leftarrow$ an arbitrary neighbor of t in R_i
    route (s,t) through:
      - the edge $\{s, s'\}$
      - a path from s' to t' in $G[R_i]$
      - the edge $\{t', t\}$
$R^* \leftarrow \cup_{\mathcal{Q}_i \in \Pi} R_i$.

and the number of paths as parameters. For RLP_{req} we considered the treewidth and the number of requests as parameters. In each case, our goal was to determine whether: (1) the problem is fixed parameter tractable for that parameter, (2) the problem is polynomial for all or some fixed values of that parameter (but possibly not FPT in that parameter), or (3) the problem is hard for that parameter.

We have several remaining open cases:

- RLP_{path} $(G, d, \mathcal{P})$ parameterized by $tw(G)$ when $d = 3, 4$.
- RLP_{req} $(G, d, \mathcal{Q})$ parameterized by $|\mathcal{Q}|$ when $d \geq 2$
- We have seen that RLP_{path} $(G, d, \mathcal{P})$ is polynomially solvable when $tw(G)$ and $L_v(P)$ are fixed. Is RLP_{path} parameterized by $tw(G) + L_v(P)$ fixed parameter tractable? (our algorithm does not guarantee this in case d is not constant)

This work can be extended in many ways, including: (a) a complete analysis of the above cases, (b) considering new parameters (e.g. cliquewidth, pathwidth, local treewidth) and their combinations, (c) considering special families of graphs, (d) providing approximation algorithms or proving approximation hardness for the NP-HARD cases.

References

1. Bateni, M., Hajiaghayi, M., Marx, D.: Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. J. ACM 58(5), 21:1–21:37 (2011)
2. Chen, S., Raghavan, S.: The regenerator location problem. In: Proceedings of the International Network Optimization Conference (INOC), Spa, Belgium (April 2007)
3. Fedrizzi, R., Galimberti, G.M., Gerstel, O., Martinelli, G., Salvadori, E., Saradhi, C.V., Tanzi, A., Zanardi, A.: Traffic Independent Heuristics for Regenerator Site Selection for Providing Any-to-Any Optical Connectivity. In: 2010 Conference on Optical Fiber Communication (OFC), Collocated National Fiber Optic Engineers Conference (OFC/NFOEC), pp. 1–3 (2010)

4. Flammini, M., Marchetti-Spaccamela, A., Monaco, G., Moscardelli, L., Zaks, S.: On the complexity of the regenerator placement problem in optical networks. IEEE/ACM Transactions on Networking 19(2), 498–511 (2011)
5. Flammini, M., Monaco, G., Moscardelli, L., Shachnai, H., Shalom, M., Tamir, T., Zaks, S.: Minimizing total busy time in parallel scheduling with application to optical networks. Theor. Comput. Sci. 411(40-42), 3553–3562 (2010)
6. Flammini, M., Monaco, G., Moscardelli, L., Shalom, M., Zaks, S.: Optimizing regenerator cost in traffic grooming. In: Lu, C., Masuzawa, T., Mosbah, M. (eds.) OPODIS 2010. LNCS, vol. 6490, pp. 443–458. Springer, Heidelberg (2010)
7. Flammini, M., Monaco, G., Moscardelli, L., Shalom, M., Zaks, S.: Optimizing regenerator cost in traffic grooming. Theoretical Computer Science 412(52), 7109–7121 (2011)
8. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica 20(4), 374–387 (1998)
9. Guo, J., Niedermeier, R., Suchý, O.: Parameterized complexity of arc-weighted directed steiner problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 544–553. Springer, Heidelberg (2009)
10. Hartstein, I., Shalom, M., Zaks, S.: On the complexity of the regenerator location problem - treewidth and other parameters. Technical Report CS-2012-08, Department of Computer Science, Technion, Haifa, Israel (October 2012)
11. Mertzios, G.B., Sau, I., Shalom, M., Zaks, S.: Placing regenerators in optical networks to satisfy multiple sets of requests. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 333–344. Springer, Heidelberg (2010)
12. Mertzios, G.B., Shalom, M., Wong, P.W.H., Zaks, S.: Online regenerator placement. In: Fernàndez Anta, A., Lipari, G., Roy, M. (eds.) OPODIS 2011. LNCS, vol. 7109, pp. 4–17. Springer, Heidelberg (2011)
13. Pachnicke, S., Paschenda, T., Krummrich, P.M.: Physical impairment based regenerator placement and routing in translucent optical networks. In: Optical Fiber Communication Conference and Exposition and The National Fiber Optic Engineers Conference (OFC/NFOEC), OWA2, San Diego, CA, USA (February 2008)
14. Shalom, M., Voloshin, A., Wong, P.W.H., Yung, F.C.C., Zaks, S.: Online optimization of busy time on parallel machines. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 448–460. Springer, Heidelberg (2012)
15. Sriram, K., Griffith, D., Su, R., Golmie, N.: Static vs. dynamic regenerator assignment in optical switches: models and cost trade-offs. In: Workshop on High Performance Switching and Routing (HPSR), pp. 151–155 (2004)
16. Yang, X., Ramamurthy, B.: Sparse regeneration in translucent wavelength-routed optical networks: Architecture, network design and wavelength routing. Photonic Network Communications 10(1), 39–53 (2005)

Online Exploration of Polygons with Holes

Robert Georges, Frank Hoffmann, and Klaus Kriegel

Freie Universität Berlin, Institut für Informatik, 14195 Berlin, Germany
{georges,hoffmann,kriegel}@mi.fu-berlin.de

Abstract. We study online strategies for autonomous mobile robots with vision to explore unknown polygons with at most h holes. Our main contribution is an $(h+c_0)!$–competitive strategy for such polygons under the assumption that each hole is marked with a special color, where c_0 is a universal constant. The strategy is based on a new hybrid approach. Furthermore, we give new lower bounds for $h = 1$.

Keywords: Polygons with holes, online exploration, competitive analysis.

1 Introduction

A classical basic task [2,14] for an autonomous mobile robot is to explore an unknown environment modeled by a polygon, possibly with polygonal holes. We assume the robot to be modeled by a point and to start from a given point, s, on the polygon's outer boundary. It is equipped with an unlimited 360° vision system that continuously provides the complete visibility polygon for its current position. When the robot has observed every point of the polygon and, therefore, knows the map of $\mathcal{P}$, it returns to s.

Considering a known polygon, an optimal tour $\mathcal{T}_{\text{opt}}$ through s can be computed offline. The robot's performance exploring the unknown polygon online is evaluated through competitive analysis. Therefore we compare the length of the tour generated by the robot with the length of $\mathcal{T}_{\text{opt}}$. If this ratio is bounded from above by a constant $\mathcal{C}$ for any problem instance, we call the strategy $\mathcal{C}$–competitive.

Over the last two decades, the problem of designing competitive online exploration strategies for certain polygon classes has received a lot of attention. A simple greedy strategy is almost optimal for simple orthogonal polygons as shown in a seminal paper by Deng et al. [5], see also [11]. Later, Hoffmann et al. [12] came up with a 26.5–competitive strategy for general simple polygons (in the following called HIKK–strategy). On the other hand, there is a lower bound for the competitive ratio of 1.28 in this case [10]. If one allows polygons with h holes there is a lower bound of $\Omega(\sqrt{h})$, even for orthogonal polygons [1], and computing the optimal offline tour becomes NP–hard [6].

The only positive result in the presence of holes we are aware of is a $O(h)$–competitive strategy for orthogonal polygons with h holes [5]. This result yields a 14–competitive strategy (L_1–metric) for the case of one hole [7].

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 56–69, 2013.

Surprisingly, there are no competitive strategies known for general polygons with at most h holes, even in the case $h = 1$. Such strategies were conjectured to exist for each h in [4]. One of the main differences between exploring orthogonal and general polygons is the following. An optimal tour that learns a single hole in an orthogonal polygon (i.e., explores its boundary) can always afford to encircle the hole. In contrast, in a general polygon the hole could have the shape of a thin long triangle and the path length needed to learn it is not necessarily related to its perimeter. Such a hole could be learned from a distance with minimal effort.

We make the following contributions to the problem of exploring polygons with holes. In Section 2 we recall some basic definitions and give for $h = 1$ a rather simple lower bound of 2 for the orthogonal case and a lower bound of 2.618 for the competitive ratio in the general case. This latter bound also holds for a modified model, where the hole is specially colored and the robot can therefore distinguish between outer boundary edges and edges of the hole. Undoubtedly, this should be of great advantage for the robot to fulfill its task. Nevertheless, it seems to be nontrivial to come up with a competitive strategy under this assumption. Subsequently we describe our strategy h–CPEX, which stands for *Colored Polygon EXploration* in presence of at most h holes. We prove it to have a competitive factor that depends on h only.

We start with describing strategy 1–CPEX in Section 3. It proceeds in two phases. In Phase 1 it follows the HIKK–strategy until the hole H is visible for the first time. Then it learns, based on a doubling strategy, the shortest tour R encircling $H \cup \{s\}$. In Phase 2 a novel hybrid approach is implemented to explore the remaining "caves" inside and outside of R. It is based on the knowledge of the length $|R|$. As soon as our strategy knows that $|R|$ is less than $c \cdot |\mathcal{T}_{\text{opt}}|$, for a suitable universal constant c, the hole is classified *safe*, meaning that the strategy can encircle it without loosing competitiveness. To this end we connect the hole with s by introducing a *barrier* (that becomes part of the boundary) and invoking the HIKK–strategy (Lemma 1) for the modified polygon. Otherwise, the hole H has the status of being currently *critical*. In this case, we subdivide the polygon by building a *fence line* f that connects the farthest (w.r.t. s) point of H with the outer boundary. We get two simple polygons, the *front yard* $\mathcal{F}$ containing s and the *backyard* $\mathcal{B}$. Again, the front yard is explored using HIKK but as soon as its path exceeds a certain length bound, we interrupt since H becomes safe and we proceed as before. Otherwise, we are left with the task of exploring the backyard. This is done by doubling arguments.

In Section 4 we generalize these ideas in a straightforward way to an arbitrary number h of colored holes. Again, h–CPEX first uses 0–CPEX until the first hole is found and classifies discovered holes H to be safe or critical. As before, the decisions are based on the knowledge of the length of the shortest tour encircling $H \cup \{s\}$. In the case of a safe hole we can invoke a recursive call of $(h-1)$–CPEX. Even the strategy for critical holes can be adopted. A main difference is the necessity to use a generalized doubling strategy, which is known as m–star search [13,14].

We show that, due to its recursive structure, the competitive ratio $\mathcal{C}_h$ of h–CPEX is bounded by $(h + c_0)!$ for a universal constant c_0. We have not tried to optimize it, our main goal is to show that its bound depends only on h.

We assume that the reader is familiar with the HIKK-strategy [12]. It serves as the base case 0–CPEX for the recursive part of the h–CPEX. Recall that its competitive ratio is $\mathcal{C}_0 = 26.5$.

2 Preliminaries and Lower Bounds

2.1 Basic Notions

Let $\mathcal{P}$ be a polygon with h pairwise disjoint holes. Its boundary bd $\mathcal{P}$ consists of the outer boundary and the boundaries of the holes. A point $y \in \mathcal{P}$ is visible from $x \in \mathcal{P}$, if the line segment $\overline{xy}$ lies completely in $\mathcal{P}$ (with bd $\mathcal{P}$ included). $\mathrm{Vis}(x) = \{y \in \mathcal{P} \mid x \text{ sees } y\}$ is the visibility polygon of $x \in \mathcal{P}$. Remark: To avoid degenerate cases we assume general position for the vertices of $\mathcal{P}$.

In our online scenario the task for the robot is to create a closed continuous path $T : [0, 1] \to \mathcal{P}$, with $T(0) = T(1) = s$, s being the starting point on the outer boundary. Such a T is a valid watchman tour if $\bigcup_{0 \le t \le 1} \mathrm{Vis}\,(T\,(t)) = \mathcal{P}$. The model assumes that for computing $T(t)$ the robot has full knowledge of $\bigcup_{t' < t} \mathrm{Vis}\,(T\,(t'))$ (including distances). In particular, if this is only a partial map of $\mathcal{P}$, its boundary contains *windows*. These are edges that are not part of bd $\mathcal{P}$ and hide regions not seen by the robot so far.

A polygon vertex is *discovered* if it has been seen by the robot. It is *explored* (*learned*) if initial segments of both incident polygon edges have been seen as well. If the internal angle of a vertex exceeds 180°, it is called a *reflex vertex*. While learning such a reflex vertex v, the robot is in the following generic situation: It knows v and one incident edge. To learn the other edge e, it has to cross the prolongation of e into the polygon's interior. This is the so called *cut* of v.

2.2 Lower Bounds

Theorem 1. *Any deterministic online strategy $\mathcal{S}_1$ that computes valid watchman routes in polygons with at most one hole has a competitive ratio ≥ 2 in the orthogonal case and $\ge \frac{3+\sqrt{5}}{2} \approx 2.618$ in the general case.*

Proof. The proof for the orthogonal polygon can be found in [8]. In the general case we use a standard fooling trick [1] to construct the lower bound. An adversary specifies the polygon completely only after observing the strategy's initial behavior.

Consider the polygon in Fig. 1(a) with a fixed but arbitrarily small constant ϵ. After an initial phase (traveling $\ge 3\epsilon$), the strategy $\mathcal{S}_1$ knows all of the polygon except the region behind reflex vertex l. Especially, it knows that the cut of l can hit the hole on the left triangle side or pass by on the right beyond line L, say at distance ≥ 1 from s. Now there are two possible ways to continue:

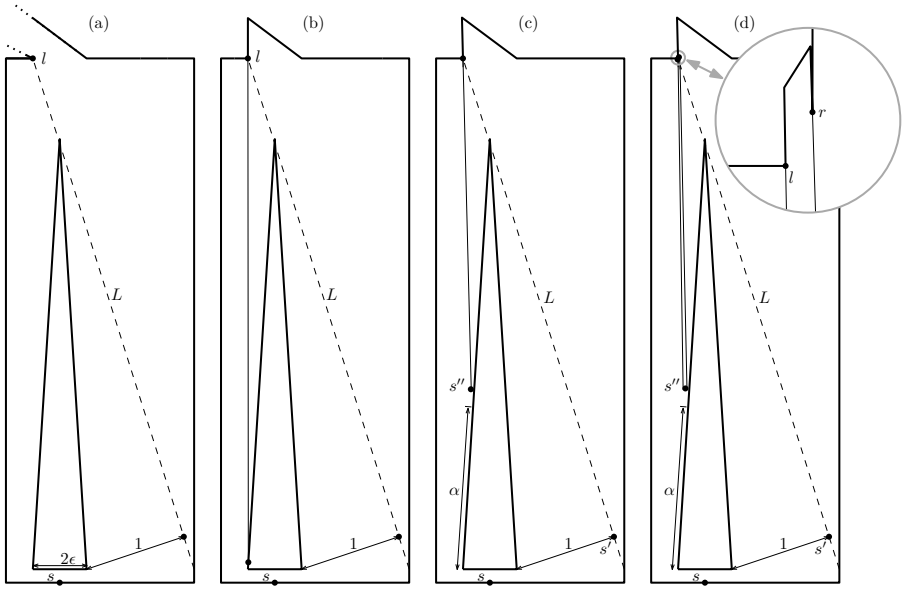


Fig. 1. Lower bound example: Placing a reflex vertex behind a vision-blocking hole

Case 1: $\mathcal{S}_1$ travels distance 1 to line L first. Then it is not competitive at all if the cut is very close to the triangle base on the left, Fig. 1(b).

Case 2: $\mathcal{S}_1$ looks for the cut on the left side first. If it moves to the top, it fails to be competitive again (e.g. the cut could be identical to L).

That's why $\mathcal{S}_1$ has to start with case 2 and, after traveling some distance α unsuccessfully, to return to explore the cut from the right side. Because of the malicious adversary, it will miss the cut by a very small distance (Fig. 1(c)). Ignoring ϵ (it can be chosen arbitrarily small), the strategy travels at least a total distance of $2\alpha+2$ to reach point s' and return to s. Now either the polygon can be explored completely or $\mathcal{S}_1$ could also learn the existence of another vertex r, hidden behind l, see Fig. 1(d). (Hint: r is very close to l and can be explored only from the left side in a competitive way.) Then $\mathcal{S}_1$ has to return once again to reach s''. This yields a total path length of $4\alpha + 2$.

In both cases the quotient of the tour length generated by $\mathcal{S}_1$ over the optimal tour length is a function in α. If the cut of l is learned in s', the competitive ratio is described by the monotonically increasing function $g(\alpha) = \frac{2\alpha+2}{2}$. If the cut points to s'' on the left side of the hole, we get the monotonically decreasing function $f(\alpha) = \frac{4\alpha+2}{2\alpha}$. Comparing both functions to determine the optimal value for α results in $\alpha = \frac{1+\sqrt{5}}{2}$, the golden ratio. We obtain $\frac{3+\sqrt{5}}{2} \approx 2.618$ as lower bound for the competitive ratio. □

In this context a colored hole would make no difference. The lower bound of $\Omega(\sqrt{h})$ [1] for polygons with h holes holds for the colored case, too. The problem

of learning a reflex vertex in the presence of vision-blocking holes turns out to be a fundamental issue for any strategy that wants to explore arbitrary polygons.

Remarks:

1. The lower bound of 2 in the orthogonal case holds for uncolored holes only.
2. The construction in the general case given above can be extended to $h > 1$ holes, but does not lead to much better results, see [8].

3 1–CPEX: Polygons with One Colored Hole

As usual, we assume that the starting point s is on the outer boundary of $\mathcal{P}$ and $\mathcal{T}_{\text{opt}}$ denotes a shortest closed watchman tour. The design of our strategy follows the basic principle that in each phase and subphase the generated path should have a length that compares to $|\mathcal{T}_{\text{opt}}|$ in a competitive way.

3.1 How to Explore a Bicolored Corridor

A basic task during the exploration is to learn the structure of the hole. Because of the coloring, each edge of the polygon can easily be associated with the hole or the outer boundary. This motivates the problem of exploring a bicolored corridor, which can be seen as a natural extension of the Cow-Path problem, see [3,14].

The corridor may have several branchings. Assume the task is to find a target t, that sees walls of both colors (Fig. 2(a)). This additional constraint guarantees that t cannot be located in a unicolored region of the polygon. At any time, the visibility polygon contains only two windows connecting two walls of different colors. These two so-called *main windows* arise from two vision blocking vertices, which can be explored on a semicircle, see [12]. The doubling paradigm [3,13] is used to link both exploration directions.

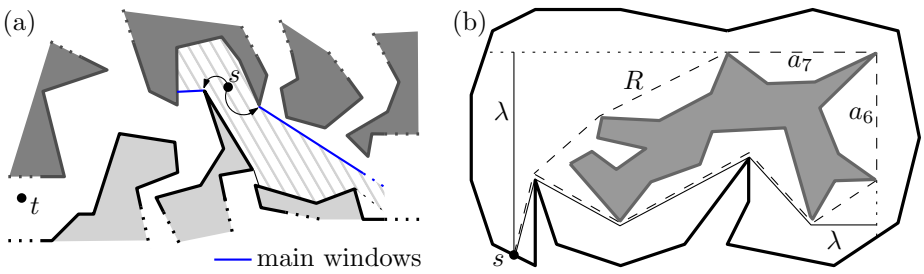


Fig. 2. (a) The bicolored corridor problem, (b) Learning the shortest path R encircling the hole

3.2 Phase 1: Learning the Shortest Tour Encircling the Hole

If no point of the hole H is visible from s, we start the 26.5-competitive HIKK-strategy (0–CPEX) until H becomes visible. The next goal is to look once around the hole, more precisely to learn the shortest tour R around it. R equals the boundary of the relative convex hull (RCH) of $H \cup \{s\}$.[1] Thus, one can imagine $R = \partial\,(\mathrm{RCH}\,(H \cup \{s\}))$ as the shape of a rubber band spanned around H and the starting point inside $\mathcal{P}$, see Fig. 2(b).

Any strategy that tries to learn R circling H in a fixed orientation will fail to be competitive. For example, consider the situation in Fig. 1(a). A strategy that explores R in cw–orientation has to walk up to the top vertex of H on the left side and down again on the right side of H. This can exceed $c \cdot |\mathcal{T}_{\mathrm{opt}}|$ for any constant c.

Thus, the situation resembles the bicolored corridor problem. We explore R in rounds via doubling, approaching the vertices corresponding to the main windows alternately on semicircles: In an odd/even round k we move in cw–/ccw–orientation 2^{k-1} length units. In each round there is a last known segment of R corresponding to a part of the bicolored corridor. It is ending at a reflex vertex that is associated to the main window and hides the next segment.

Combining this with the factor 2 of the semicircle strategy [12] and the factor 9 of the doubling approach we can show that our strategy learns R with total path length $\leq 36\,|\mathcal{T}_{\mathrm{opt}}|$.

After having learned R the strategy can derive the following lower bound λ on $|\mathcal{T}_{\mathrm{opt}}|$.

Let $a_1, a_2, \ldots, a_n$ be the ccw-oriented chain of line segments defining R, starting from s. Any strategy that learns R has to see each vertex $p_i \in R$, incident with a_i and a_{i+1}, both from the right half-plane of a_i and from the right half-plane of a_{i+1}. The path length to fulfill this task for p_i, maximized over all vertices of R, defines a lower bound λ to learn R and therefore a lower bound on $|\mathcal{T}_{\mathrm{opt}}|$.[2]

In Fig. 2(b) the lower bound λ is realized by the effort to learn (a_6, a_7).

3.3 Phase 2: The Hybrid Approach

The hole H in $\mathcal{P}$ is called *c–safe* (for a fixed constant c) if $|R| \leq c\,|\mathcal{T}_{\mathrm{opt}}|$ holds. As long as the strategy does not know whether the hole is c–safe, the hole is called *c–critical*. While learning more about $\mathcal{P}$, the strategy can also learn more about lower bounds on $|\mathcal{T}_{\mathrm{opt}}|$. This way the status of a hole can change from *c–critical* to *c–safe*.

Observation: If $|R| \leq c\lambda$, then the hole is c–safe.

[1] A set M is relatively convex in $\mathcal{P}$ if for each pair of points in M the geodesics (shortest paths) connecting them are included in M.

[2] This definition of λ is equivalent with that given in [9] for $h = 1$ and it easily generalizes to the case $h > 1$.

Lemma 1. *Any polygon $\mathcal{P}$ with a c–safe hole H can be explored online with total tour length $\leq (4c+2) \cdot \mathcal{C}_0 \cdot |\mathcal{T}_{\text{opt}}|$.*

Proof. Consider a shortest path b from s to H. If it contains reflex vertices of $\mathcal{P}$, we slightly shift b into the polygon's interior. b is treated as an additional boundary that transforms $\mathcal{P}$ into a simple polygon $\mathcal{P}'$ (Fig. 3(a)). We show that 0–CPEX (i.e., HIKK) applied to $\mathcal{P}'$ generates a tour that meets the claimed bound. To this end it is sufficient to show that there is a closed watchman tour $\mathcal{T}_{\text{e}}$ in $\mathcal{P}'$ of length $|\mathcal{T}_{\text{e}}| \leq (4c+2) \cdot |\mathcal{T}_{\text{opt}}|$.

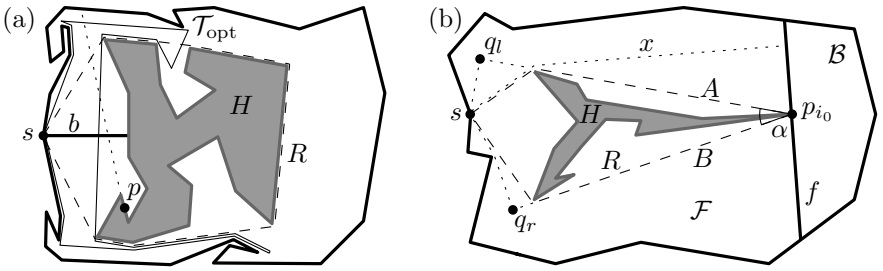


Fig. 3. Exploring a polygon with (a) a c–safe and (b) a c–critical hole

Barrier b can cut the original $\mathcal{T}_{\text{opt}}$ into several pieces. Moreover, it can reduce the visibility from $\mathcal{T}_{\text{opt}}$. E.g., point p in Fig. 3(a) is not visible from the original $\mathcal{T}_{\text{opt}}$ after inserting b. We use R plus two copies of b (one on the left, the other one on the right side of the barrier) to link together all pieces of $\mathcal{T}_{\text{opt}}$ and to restore full vision. E.g., point p is now visible from the right side copy of b. Doubling this structure we get a Eulerian graph and a Eulerian tour $\mathcal{T}_{\text{e}}$. Finally:

$$|\mathcal{T}_{\text{e}}| = 2\,(|R| + 2|b| + |\mathcal{T}_{\text{opt}}|) \leq 4|R| + 2\,|\mathcal{T}_{\text{opt}}| \leq (4c+2)\,|\mathcal{T}_{\text{opt}}| \quad .$$

□

The hybrid approach consists in implementing the following rule: As soon as the strategy learns that the hole is c–safe for a suitable chosen constant c, it switches to the simple polygon mode using Lemma 1. The following lemmas expose some important properties of a hole that appears to be c-critical after learning R, because of $|R| > c\lambda$. Especially, it turns out that $c = 6$ is an appropriate choice for the constant in the hybrid approach.

Lemma 2. *Assume that H is a hole with $|R| > 6\lambda$. Let a_i and a_{i+1} be the two segments of R that define λ, p_i their common apex and α the enclosed angle. Then $\alpha < \frac{\pi}{6}$.*

Proof. Let q_l and q_r be the two endpoints of the λ–path that see p_i from the left and right side (Fig. 3(b) . Denoting by $A = |\overline{q_l p_i}|$ and $B = |\overline{q_r p_i}|$ the distances to the apex, we have $A + B + \lambda \geq |R| > 6\lambda$. Consequently, $A \geq 2.5\lambda$ or $B \geq 2.5\lambda$ and the claim follows from the sine rule in the triangle $\Delta(p_i, q_l, q_r)$. □

Using the notations of Lemma 2 we define a *fence* f, subdividing $\mathcal{P}$ into two simple polygons $\mathcal{F}$ (the *front yard*), and $\mathcal{B}$ (the *backyard*), see Fig. 3(b). f is chosen to be the line segment perpendicular to the angular bisector of α through p_i.

Lemma 3. *Assume that H is a hole with $|R| > 6\lambda$ and f the fence of H. Let x denote twice the shortest path length from s to f. If $|\mathcal{T}_{\text{opt}}| \geq x$, the hole H is 4–safe.*

Proof. We use the notations from the proof of Lemma 2. It is sufficient to show that $x \geq 0.25\,|R|$. Combining $A + B + \lambda \geq |R| > 6\lambda$ with the triangle inequality $A + \lambda \geq B$ we obtain $2A + 2\lambda \geq |R|$ and $A \geq (\frac{1}{2} - \frac{1}{6})\,|R| = \frac{1}{3}\,|R|$. From Lemma 2 we know that the shortest path length from q_l to f has length $\geq A \cos\frac{\pi}{12} \geq 0.96A$. Since the distance from s to q_l in $\mathcal{P}$ is at most $\lambda \leq \frac{1}{6}\,|R|$ we end with

$$\frac{x}{2} \geq 0.96A - \lambda \geq \left(\frac{0.96}{3} - \frac{1}{6}\right)|R| > 0.15\,|R| \quad .$$

□

Lemma 4. *Let H be a hole with $|R| > 6\lambda$ and x be twice the shortest path length from s to fence f of H in $\mathcal{P}$. Then the front yard $\mathcal{F}$ can be learned by 0–CPEX with tour length $\leq \mathcal{C}_0 \cdot x$ or it turns out that H is 4–safe.*

Proof. 0–CPEX explores $\mathcal{F}$ from s and stops either if
(1) the exploration path length l reaches $\mathcal{C}_0 x$ with $\mathcal{F}$ being still unexplored or if
(2) $\mathcal{F}$ gets explored with tour length $l \leq \mathcal{C}_0 x$.

We will prove, that this procedure is $\mathcal{C}_0$–competitive in both cases, i.e. $l \leq \mathcal{C}_0\,|\mathcal{T}_{\text{opt}}|$, and that in case (1) the hole H becomes 4–safe. We remind that $\mathcal{T}_{\text{opt}}$ is the optimal exploration tour for the whole polygon $\mathcal{P}$, whereas $\mathcal{T}_{\text{opt}}(\mathcal{F})$ denotes an optimal tour for the front yard. Now, we combine cases (1) and (2) with another case distinction: (a) $|\mathcal{T}_{\text{opt}}| \leq x$ or (b) $|\mathcal{T}_{\text{opt}}| > x$. Remark that in case (a) $\mathcal{T}_{\text{opt}}$ does not leave $\mathcal{F}$, and consequently $|\mathcal{T}_{\text{opt}}(\mathcal{F})| \leq |\mathcal{T}_{\text{opt}}|$. In case (b) the hole becomes 4-safe by Lemma 3.

Case (1): Since the exploration is not finished and 0–CPEX is $\mathcal{C}_0$–competitive, in case (a) we have $l = \mathcal{C}_0 x < \mathcal{C}_0\,|\mathcal{T}_{\text{opt}}(\mathcal{F})| \leq \mathcal{C}_0\,|\mathcal{T}_{\text{opt}}|$, and therefore $x < |\mathcal{T}_{\text{opt}}|$, a contradiction. Otherwise, (b) holds and thus $l = \mathcal{C}_0 x < \mathcal{C}_0\,|\mathcal{T}_{\text{opt}}|$. Here H is 4–safe by Lemma 3.

Case (2): If (a) holds then $l \leq \mathcal{C}_0\,|\mathcal{T}_{\text{opt}}(\mathcal{F})| \leq \mathcal{C}_0\,|\mathcal{T}_{\text{opt}}|$. Otherwise $l \leq \mathcal{C}_0 x \leq \mathcal{C}_0\,|\mathcal{T}_{\text{opt}}|$.

□

Now, we are ready to describe the behavior of 1-CPEX if H is a 6-critical hole: It starts 0-CPEX in the front yard $\mathcal{F}$ with the restriction from Lemma 4. If H becomes 4–safe, we proceed as described in Lemma 1. In the other case it remains to explore the backyard $\mathcal{B}$ from s. Observe, we are now in a situation similar to our lower bound construction. We know there are reflex vertices in $\mathcal{B}$

that hide polygon edges we have to learn. But it is not clear whether to approach the corresponding cuts on the left or right side of the hole.

We describe how to learn a group of left reflex vertices, compare [12]. The existence of these vertices has been "observed" along the way while learning R, respectively $\mathcal{F}$. But, of course, this has not influenced the tours generated in these subroutines. Basically, cuts of such vertices can lie completely in $\mathcal{B}$ or they can cross the fence line f. As soon as we know that there is a cut not crossing f, H becomes 4–safe, since $\mathcal{T}_{\text{opt}}$ intersects f (Lemma 3).

A target vertex l can be located in three different regions of $\mathcal{B}$ (Fig. 4). Cuts of vertices in $\mathcal{B}_3$ crossing f on the left of p_{i_0} have been explored along the way (as soon as they have been discovered), otherwise they cannot be visible yet. All other cuts of vertices in $\mathcal{B}_1$ and $\mathcal{B}_2$ crossing f on the left are also crossing R, for the same reason. Therefore following the angle hull [12] of R on the left side is a suitable way to explore these cuts. Due to space limitations we have to omit further details. These can be found in [7,8].

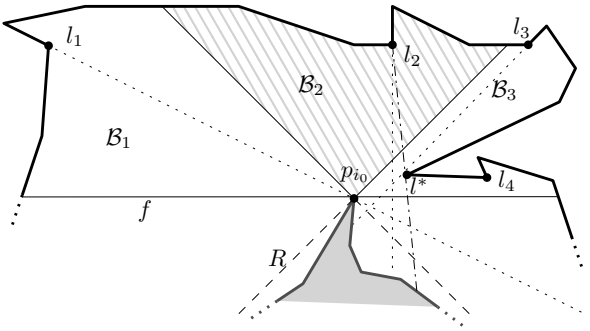


Fig. 4. Different types of left vertices in the backyard $\mathcal{B}$

A similar result can be shown for cuts crossing f on the right part. If we are sure that a vertex has to be explored from the right, we put it into a special list $\mathcal{V}$ which is learned afterwards (only from the right side, without doubling).

Vertices in $\mathcal{B}_1$ are hidden behind the top vertex p_{i_0} of H (e.g., l_1 in Fig. 4), otherwise they would have been already explored. Therefore, approaching p_{i_0} on a semicircle is competitive, wherever the target vertex is located. If it becomes visible, it is explored or it can be added to $\mathcal{V}$.

Vertices in $\mathcal{B}_2$ and $\mathcal{B}_3$ have either been seen from the right side and can also be added to $\mathcal{V}$ (e.g., l_4 in Fig. 4), or they have been discovered from the left and are hidden behind a vertex l^* from the right (e.g., l_2 and l_3 in Fig. 4). In the second case they have to be approached on the angle hull again.

Notice, that the possible paths (angle hull and semicircle) do not depend on special vertices. Therefore we can follow them via doubling until all left vertices become explored or we reach the fence. In the second case at least one cut is not crossing f and the hole is 4–safe. It can be shown that the length of the path

traveled in $\mathcal{F}$ is bounded by a constant times the shortest path length from s to f, see [7]. In summary, we get the following result.

Theorem 2. 1–*CPEX is a* $O(1)$–*competitive strategy for exploring polygons with at most one colored hole and given starting point on the outer boundary.*

A closer look at this case yields a competitive factor of ≈ 610, see [7].

4 General Case: Constant Number of Colored Holes

We extend the strategy to deal with more than one, say at most h, pairwise differently colored holes. The idea of c–safe holes can be adapted to reduce the exploration problem to a polygon with $(h-1)$ differently colored holes.

All visible holes are organized in a list $\mathcal{H}$. Each hole in $\mathcal{H}$ is marked with its current state: *discovered*, *critical*, or *safe*.

4.1 Shortest Tours around Holes

For any $H \in \mathcal{H}$ we denote by R_H a shortest tour around the hole H that starts and ends in s. If this tour is not unique, we choose the unique one encircling the largest area (Fig. 5(a)). We remark that R_H can encircle other holes, too. It is also possible that it does not touch H at all (Fig. 5(b)), and it can differ from the outer boundary of RCH $(H \cup \{s\})$ as well. In the situation that R_H encircles also another hole H', the shortest path properties imply that either $R_{H'} = R_H$ or the region encircled by $R_{H'}$ is properly contained in the region encircled by R_H.

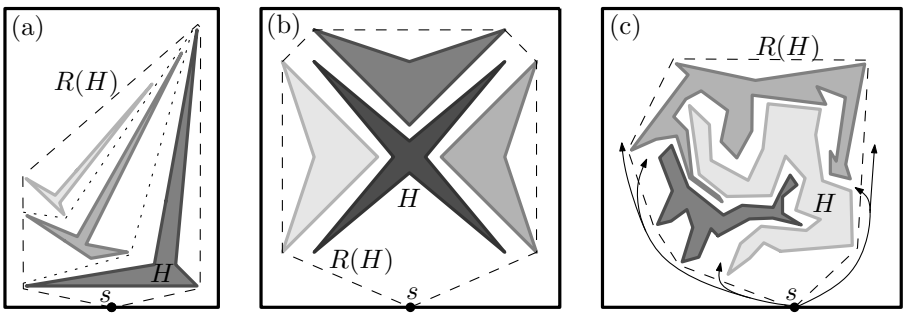


Fig. 5. (a) Three shortest paths encircling hole H, (b) R_H does not touch H, (c) the star search approach

Now, the exploration of R_H is more difficult because we can't predict whether it runs always through bicolored corridors with the color of H on one side or if it also uses corridors with two other colors. Again, we start the exploration of R_H cw. and ccw. around H, but whenever another hole H' occurs in the search

range, we have to check both possibilities: R_H could run cw. or ccw. around H' (Fig. 5(c)). Thus, we have to replace the doubling approach from 3.2 by a star search strategy [14]. The bicolored corridors form the edge set E of a planar graph with h faces and vertices of degree at least 3. From Euler's formula we can conclude $|E| \leq 3h/2$. Since each corridor will be used at most twice (from both sides) a $3h$-star search will suffice, which increases the competitive ratio for this phase by a factor of $2e \cdot 3h + 1$, [14].

Once knowing R_H, we derive the lower bound $\lambda_H \leq |\mathcal{T}_{\text{opt}}|$ in the same way as in 3.2, Fig. 2(b). Moreover, we can use all the conclusions for safe holes drawn in the 1–hole case. Lemma 1 for c–safe holes can be extended to h holes, too.

Lemma 5. *If a c–safe hole is found, the polygon can be explored with $(h-1)$–CPEX, guaranteeing a total path length $\leq (4c+2)\mathcal{C}_{h-1}\,|\mathcal{T}_{\text{opt}}|$.*

Proof. The hole H has been discovered and categorized c–safe. Therefore, we found a path b connecting it with the starting point, running completely in R_H. We have to ensure that

$$|b| \leq \frac{1}{2}\,|R_H| \quad . \tag{1}$$

As mentioned before, any obstacle interfering with R_H and b has to be a c–safe hole, too. That's why a c–safe hole H with path b satisfying (1) can be found and the construction of Lemma 1 can be used. □

Lemma 5 allows the recursive call of $(h-1)$–CPEX if a c–safe hole is found. In that case the status of all other holes in $\mathcal{H}$ is reset to *discovered* because in the newly derived polygon the shortest tour encircling a hole can have changed.

4.2 The Algorithm: h–CPEX

For our algorithm we initialize $\mathcal{H}$ as an empty list and set $\lambda = 0$. There are two basic rules h–CPEX will follow:

(R1) As soon as a hole is discovered, we will classify it. Only exception: We are currently classifying another hole.

(R2) As soon as a hole gets classified as 6–safe, we recurse and invoke $(h-1)$–CPEX.

Overall, the CPEX exploration is divided into three major steps:

1. Classifying Holes

If no hole has been discovered yet, we apply HIKK for simple polygons until the first hole becomes visible and add it to our list $\mathcal{H}$ marked as *discovered*. Now, R_H has to be learned for every discovered hole H with the help of the star search algorithm, visiting all possible corridors, until a point p on R_H is visible from both sides of the hole. If such a point is found, the strategy has to be applied another round: The shortest path could have been missed because of the malicious adversary. Next, we compute the lower bound λ_H and define $\lambda = \max(\lambda,\ \lambda_H)$. If new holes are found, they are added to $\mathcal{H}$,

too. If $|R_H| \leq 6 \cdot \lambda$, the hole is safe and we apply R2. Otherwise, we mark H as 6–critical. If λ has changed, we have to check all holes previously marked critical. They might be safe now and we can recurse as well.

2. Exploring Front Yards

At this stage, $\mathcal{H}$ only contains critical holes. For each group G_R, formed by holes that have the same shortest tour R surrounding them, we create $\mathcal{F}_R$ by inserting a fence f_R and intersecting the polygon with the corresponding half plane through the top of R (see strategy for one hole). Because the fence connects at least one hole with the outer boundary, the number of holes decreases and we have to update list $\mathcal{H}$ (Fig. 6(a) and (b)).

Now, $(h-1)$–CPEX for $\mathcal{F}_R$ can be used. If its path length exceeds $\mathcal{C}_{h-1} \cdot x$, all holes in G_R become c–safe and we recurse. Otherwise, $\mathcal{F}_R$ is explored completely. If a new hole is discovered we add it to $\mathcal{H}$ and apply R1.

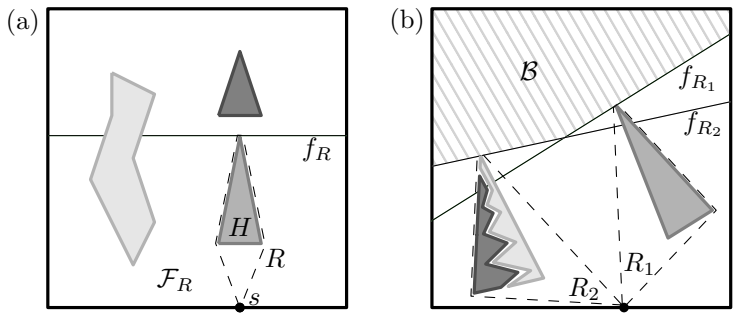


Fig. 6. The fence, front yards and backyard in a polygon with several holes

3. Exploring the Backyard

Finally, we explore the backyard $\mathcal{B} = \mathcal{P} \setminus \bigcup_R \mathcal{F}_R$ (Fig. 6(b)) as described before in section 3.3. The doubling approach has to be replaced by star search again. As in step 2, if a new hole is discovered, add it to $\mathcal{H}$ and apply R1.

4.3 The Competitive Factor

Theorem 3. *The strategy h–CPEX is $(h + c_0)!$-competitive.*

Proof. Recall that $\mathcal{C}_h$ denotes the competitive factor of h–CPEX. For 0–CPEX we use the HIKK–factor $\mathcal{C}_0 = 26.5$. Analyzing the different stages of h–CPEX, we obtain the following recursive estimation:

$$\mathcal{C}_h \leq c_1 h^2 + c_2 \mathcal{C}_{h-1} + h\mathcal{C}_{h-1} + c_3 h \ .$$

The first term comes from the classification of the h holes, each using a $3h$–star search with a $O(h)$ competitive factor. The second term comes into play whenever we have a recursive call of $(h-1)$–CPEX for a safe hole. The constant

$c_2 = 26$ stems from Lemma 5 dealing with 6-safe holes. In the case that all holes are 6-critical, we have to explore at most h front yards, each implying a recursive call of $(h-1)$–CPEX and, finally, the exploration of the backyard. The latter is basically an h–star search for groups of left and right vertices. This estimation is obviously dominated by the second and the third term. We get $\mathcal{C}_h \leq c_4 \cdot (h+c_2)! \cdot \mathcal{C}_0 \leq (h+c_0)!$ for sufficiently large constants c_4 and c_0. □

5 Conclusion and Future Work

We have addressed the problem of online exploring polygonal scenes cluttered with at most h polygonal obstacles (holes). In the standard model exploring the scene includes the subtask of recognizing which boundary parts belong to holes and which edges form the outer boundary. In this paper we proposed a modified model making this subtask trivial by giving each hole a special color.

Under this assumption we described a competitive exploration strategy for each $h > 0$. We consider this to be a major breakthrough towards settling the general conjecture from [4] that such competitive strategies exist in the uncolored case as well. The missing link could be a combination of star search with a HIKK–like strategy to learn which holes exist in an uncolored scene. Moreover, we are sure that the competitive factor can be considerably improved.

The main technical novelty of h–CPEX is the use of distance information for making decisions. Recall that the HIKK–strategy [12] is mainly based on topological information, like the order of vertices along the boundary.

Finally, we remark that for $h \geq 2$ holes the task of exploring the polygon is no longer equivalent to exploring all edges, compare Fig. 7. However, this problem is not an issue for h–CPEX because of its recursive structure.

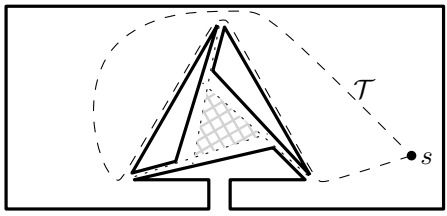


Fig. 7. Seeing all boundary edges does not guarantee full exploration

Acknowledgment. We would like to thank the anonymous referees for their valuable comments.

References

1. Albers, S., Kursawe, K., Schuierer, S.: Exploring Unknown Environments with Obstacles. In: Proc. of the 10th SODA, pp. 842–843. SIAM, Philadelphia (1999)
2. Berman, P.: On-line Searching and Navigation. In: Fiat, A., Woeginger, G. (eds.) Online Algorithms 1996. LNCS, vol. 1442, pp. 147–177. Springer, Heidelberg (1998)
3. Chrobak, M., Kenyon-Mathieu, C.: Online Algorithms Column 10: Competitiveness via doubling. SIGACT News 37(4), 115–126 (2006)
4. Deng, X., Kameda, T., Papadimitriou, C.: How to Learn an Unknown Environment. In: Proceedings 32nd FOCS, pp. 298–303. IEEE Computer Society (1991)
5. Deng, X., Kameda, T., Papadimitriou, C.: How to Learn an Unknown Environment I: The Rectilinear Case. JACM 45(2), 215–245 (1998)
6. Dumitrescu, A., Tóth, C.D.: Watchman tours for polygons with holes. CGTA 45(7), 326–333 (2012)
7. Georges, R.: Online-Erkundung von Polygonen. Diploma thesis, Freie Universität Berlin (2012), `http://www.georges-1-cpex.de.vu`
8. Georges, R., Hoffman, F., Kriegel, K.: Online Exploration of Polygons with Holes. In: arXiv.org, arXiv:1207.0240v1 (2012)
9. Georges, R., Hoffmann, F., Kriegel, K.: On the Exploration Problem for Polygons with One Hole. In: Abstracts 28th European Workshop Comput. Geom. University Perugia (2012)
10. Hagius, R., Icking, C., Langetepe, E.: Lower Bounds for the Polygon Exploration Problem. In: Abstracts 20th European Workshop Comput. Geom. Universidad de Sevilla (2004)
11. Hammar, M., Nilsson, B.J., Persson, M.: Competitive exploration of rectilinear polygons. Theor. Comput. Sci. 354(3), 367–378 (2006)
12. Hoffmann, F., Icking, C., Klein, R., Kriegel, K.: The Polygon Exploration Problem. SIAM J. Comput. 31, 577–600 (2002)
13. Klein, R.: Algorithmische Geometrie. Springer (2005)
14. Papadimitriou, C.H., Yannakakis, M.: Shortest Paths Without a Map. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 610–620. Springer, Heidelberg (1989)

Probabilistic k-Median Clustering in Data Streams

Christiane Lammersen[1,⋆], Melanie Schmidt[2,⋆⋆], and Christian Sohler[2]

[1] Simon Fraser University, Burnaby, B.C., Canada
clammers@cs.sfu.ca
[2] TU Dortmund, Germany
{melanie.schmidt,christian.sohler}@udo.edu

Abstract. We define the notion of coresets for probabilistic clustering problems and propose the first (k, ε)-coreset constructions for the probabilistic k-median problem in the metric and Euclidean case. The coresets are of size $\text{poly}(\varepsilon^{-1}, k, \log(W/(w_{\min} \cdot p_{\min} \cdot \delta)))$, where W is the expected total weight of the weighted probabilistic input points, $w_{\min}$ is the minimum weight of a probabilistic input point, $p_{\min}$ is the minimum realization probability, and δ is the error probability of the construction. We show how to maintain our coreset for Euclidean spaces in data streams.

1 Introduction

Many real world datasets contain uncertain, imprecise or incomplete data. Typical examples are measurements of sensor networks or datasets arising from record linkage across multiple data sources. Imagine, for example, that we maintain a dataset representing our knowledge on a certain set of entities. Now, we want to add information from an additional dataset containing data records of some, but not all of our entities. The new dataset might also contain data records of entities not present in our knowledge base. In our scenario, we might not be sure whether two data records from the two sets belong to the same entity or not, but we might have a good idea of how likely this is. In a Baysian way, this likeliness can be modeled as a probability. Thus, if we view our knowledge base as a set X of multi-dimensional points, each data record from our new set can be seen as a discrete probability distribution over X with a total probability between 0 and 1. If we are now interested in analyzing our new data, we have to deal with uncertainty, i. e., we must be able to cope with uncertain points.

One important tool for data analysis also needed in the case of uncertain data is *clustering*, i. e., the problem to partition a given set of objects into subsets called *clusters* such that objects in the same cluster are similar and objects in different clusters are dissimilar. Clustering is a useful tool for data compression,

⋆ Research partially supported by the EU within the 7th Framework Programme under contract No. 255827, an Ebco Eppich and a PIMS Fellowship and DFG grant SO 514/4-3.

⋆⋆ Corresponding author.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 70–81, 2013.

object classification or pattern recognition. Cormode and McGregor [12] introduced two variations of *probabilistic clustering* both assuming that the input is a set of probabilistic points, each formalized as a probability distribution function which describes the possible locations of the points. In *unassigned clustering*, each point is assigned to the closest cluster center. In the second variation, called *assigned clustering*, each point is assigned to a fixed cluster center, no matter where it is actually located. We focus on the assigned version of the *probabilistic k-median problem*. The k-median objective is one of the most frequently used clustering objectives. In our probabilistic version of the k-median problem, we allow that the total realization probability of a point can be smaller than 1, i. e., it is possible that a point is not realized at all. The goal is now to find k cluster centers, which are deterministic points in the considered metric space, and an assignment of the points to these cluster centers such that the sum of the expected distances of the points to the cluster centers is minimized. Note that we have to assign each point to a cluster center before we know where it is eventually realized.

In addition to containing uncertain data, real world data sets are often large and are given as a data stream or stored on hard disks. In these cases, random access to the data is too time consuming or not even possible, which then calls for the development of streaming algorithms. We study the development of streaming algorithms based on *coresets*. Intuitively, a coreset is a small set of weighted points that approximates the input points with respect to the studied clustering problem. Instead of clustering the original dataset, a clustering algorithm can then be run on the small coreset to obtain a $(1 + \varepsilon)$-approximation in shorter running time. We develop two different coreset constructions, one for the *metric* and one for the *Euclidean* uncertain k-median-clustering problem.

Related Work on Clustering Uncertain Data. In recent publications, some traditional clustering heuristics have been extended so that they can handle uncertain data. For instance, Chau et al. [10] and Ngai et al. [34] extended Lloyd's k-means algorithm [16,32], and Kriegel and Pfeifle [28,29] and Xu and Li [36] extended the density-based clustering algorithm DBSCAN [14] for handling uncertainty. Furthermore, Günnemann et al. [20] developed a subspace clustering heuristic for uncertain data. A survey of uncertain data mining and management applications can be found in [2].

Surprisingly, only a few theoretical results on clustering uncertain data have been obtained so far [12,19]. Cormode and McGregor [12] achieve a $\mathcal{O}(1)$-approximation for the assigned metric k-median problem by first computing the 1-median of each probabilistic input point and then clustering the 1-medians. They also consider other clustering problems. For the unassigned Euclidean k-median problem and both the unassigned and assigned Euclidean k-means problem, they obtain a $(1+\varepsilon)$-approximation by using a simple reduction to weighted deterministic clustering problems. For the unassigned metric k-center problem, they propose a bicriteria approximation algorithm which results in a constant factor approximation but uses $2k$ instead of k cluster centers. Guha and Munagala [19] improve the last-mentioned result by using a reduction to a 'truncated'

version of a deterministic metric k-median problem. They obtain a constant factor approximation for both the unassigned and assigned metric k-center problem that preserves the number of allowed cluster centers k.

Related Work on Clustering Certain Data. The deterministic k-median clustering problem is well-studied. We start with an overview for Euclidean k-median. Arora et al. [4] develop the first $(1+\varepsilon)$-approximation algorithm for the Euclidean k-median problem in the plane by extending the technique developed by Arora [3] for the Euclidean TSP. Kolliopoulos and Rao [27] improve their work by solving the d-dimensional case with $d \geq 2$ and reducing the running time to $\mathcal{O}(2^{\tilde{\mathcal{O}}(1/\varepsilon)^{d-1}} n \log^{d+6} n)$, where n is the number of input points. Further improvements were then based on *coresets*. Agarwal et al. [1] introduced the concept of coresets for extent measures of points. The first coreset construction for the clustering problems is due to Bădoiu et al. [6] leading to a $(1+\varepsilon)$-approximation with an expected running time of $\mathcal{O}(2^{\mathrm{poly}(k,1/\varepsilon)} d^{\mathcal{O}(1)} n \log^{\mathcal{O}(k)} n)$ for the k-median problem. Har-Peled and Mazumdar [22] introduce strong coresets for clustering problems. There are various coreset constructions and $(1+\varepsilon)$-approximations [11,15,17,21,22,30]. Most relevant to our work is the construction by Chen [11] which gives a coreset of size $\mathcal{O}(dk^2\varepsilon^{-2}\log(n)\log(k/\varepsilon))$ leading to a $(1+\varepsilon)$-approximation in time $\mathcal{O}(ndk + 2^{(k/\varepsilon)^{\mathcal{O}(1)} d^2 \log^{k+2} n})$.

For general metric spaces, a $(1+\varepsilon)$-approximation with running time polynomial in k is not possible for $\varepsilon < 0.73$ unless NP $\subseteq$ DTIME$(n^{\mathcal{O}(\log\log n)})$ [25]. Charikar et al. [9] give the first constant-factor approximation. Indyk [24] develops a randomized bicriteria approximation with constant approximation factor and $\mathcal{O}(k)$ centers. Based on that, Guha et al. [18] give a constant-factor approximation with running time $\tilde{\mathcal{O}}(nk)$ and show how to maintain it in insertion-only data streams. Mettu and Plaxton [33] prove an $\Omega(nk)$ lower bound for any constant-factor approximation, even for randomized algorithms, and also achieve a running time of $\tilde{\mathcal{O}}(nk)$. Chen [11] uses coresets to develop a $(10+\varepsilon)$-approximation algorithm with running time $\mathcal{O}(nk + k^7\varepsilon^{-5}\log^5 n)$. The coreset has size $\mathcal{O}(dk^2\varepsilon^{-2}\log n\log(k/\varepsilon))$. The approximation factor is improved to 6 by Jain and Vazirani [26], to 4 by Charikar and Guha [8], and finally to 3 by Arya et al. [5]. The latter algorithm uses local search and analyzes the locality gap.

To the best of our knowledge, there does not exist any coreset construction for probabilistic clustering problems.

Our Contribution. The focus of our work is introducing and constructing probabilistic coresets. What should such a coreset look like? It can certainly contain probabilistic points. However, in addition to the number of such points, the overall storage size is also influenced by the representation size of the probability distributions of the points. Thus, we define a coreset by restricting both the number and the representation size of the probabilistic points.

Our first construction is a reduction to the deterministic k-median problem. Note that the expected distances between points do not satisfy the properties of a metric space because two copies of the same probabilistic point do not have expected distance zero as it should be for identical elements of a metric space.

Thus, the intuitive reduction does not work. However, the *earth mover distance* can be used to define a metric space and hence makes the use of existing coreset constructions for the deterministic metric k-median problem possible.

In the Euclidean case, one can also transform a problem instance into an input for the deterministic metric k-median problem, because a Euclidean space is also a metric space. However, the resulting instance is indeed metric but not Euclidean anymore. Thus, a coreset construction based on the transformed input can only use metric coreset constructions and not constructions specifically for Euclidean inputs. Now, the problem is that while metric algorithms provide a coreset which grants an approximation for all centers picked from a finite subset of the metric space, coresets for the Euclidean clustering problem have to approximate center sets from the infinite Euclidean space. Due to this fact, one cannot use a metric coreset construction to compute a coreset for the Euclidean k-median problem.

We develop a coreset construction for the probabilistic Euclidean k-median problem by extending and further developing a technique of Chen [11] for the deterministic k-median problem. The main idea of Chen's construction is to use a bicriteria approximation to partition the Euclidean space into (ring-shaped) regions of points which are close to each other (in terms of their optimal clustering cost) and then to sample representatives from each such subset. When we want to apply Chen's construction to probabilistic data, we are faced with the problem that a probabilistic point can be realized in different regions. Furthermore, there exists no bi-criteria approximation. We overcome these difficulties by first applying Chen's construction to probabilistic points by using 1-medians of a probabilistic points as their location. The probem is now to deal with different possible realizations of probabilistic points. In order to do so, we further refine Chen's partition and replace uniform sampling by a weighted sampling procedure. In the analysis we show that after these modifications the resulting point set is a probabilistic coreset with sufficiently high probability.

Both our coreset constructions also work for the *weighted* probabilistic k-median problem. Furthermore, we show how to maintain the Euclidean coreset in data streams.

2 Preliminaries

Let $M = (X, \mathrm{D})$ be a metric space, where X is a set of points and D is a distance function defined on X. For a finite subset $Y \subseteq X$ and a point $x \in X$, let $\mathrm{D}(x, Y) := \min_{y \in Y}\{\mathrm{D}(x, y)\}$ denote the minimal distance between x and points in Y. For finite subsets $Y, Z \subseteq X$, let $\mathrm{D}(Y, Z) := \min_{y \in Y}\{\mathrm{D}(y, Z)\}$ denote the minimal distance between points in Y and Z. For a finite set $Y \subseteq X$, let $\mathrm{diam}(Y) := \max_{y,z \in Y} \mathrm{D}(y, z)$ denote the *diameter* of Y. For a point $x \in X$ and a non-negative value R, define $\mathcal{B}(x, R)$ as the ball with center x and radius R.

Next, we define the *probabilistic k-median-clustering problem*. Let $\mathcal{X} := \{x_1, \ldots, x_m\} \subseteq X$ be a finite set of m points from the metric space M, and let $V := \{v_1, \ldots, v_n\}$ be a set of n *nodes*, where each node v_i follows an

independent probability distribution $\mathcal{D}_i$ over $\mathcal{X}$. For any $i \in [n]$ and any $j \in [m]$, we denote the probability that the node v_i is realized at x_j by p_{ij}. We denote the total probability that v_i is realized by $p_i := \sum_{j=1}^{m} p_{ij}$. We assume that $p_i \leq 1$, i. e., with probability $1 - p_i$ node v_i is not realized. Let $p_{\min}$ be the smallest realization probability, i. e., $p_{\min} = \min_{i\in[n],j\in[m]}\{p_{ij}\} \leq p_{ij}$ for all $i \in [n], j \in [m]$.

Definition 1 (Probabilistic k-Median Problem). *Given $k \in \mathbb{N}$, a finite set of κ possible center locations $\mathcal{C} \subseteq X$ and a positive weight function $w : V \to \mathbb{R}_{\geq 0}$ on the set of nodes V, the* weighted probabilistic metric k-median-clustering problem *for V is to find a set $C := \{c_1, \ldots, c_k\} \subseteq \mathcal{C}$ of k cluster centers and an assignment $\rho : V \to C$ minimizing the expected k-median-clustering cost $\mathbf{E}_{\mathcal{D}}\left[\mathrm{cost}_w(V, C, \rho)\right] := \sum_{i=1}^{n} w(v_i) \sum_{j=1}^{m} p_{ij} \cdot \mathrm{D}(x_j, \rho(v_i))$. The* unweighted *problem results from setting $w(v_i) = 1$ for all v_i. We denote the cost of an optimal clustering by $\mathrm{cost}_k^*(V)$, the total weight by $W := \sum_{v_i \in V} w(v_i)p_i$ and the minimum weight by $w_{\min} := \min_{v_i \in V}\{w(v_i)\}$ (we assume that the problem is scaled such that $w_{\min} \geq 1$). The definition is verbatim for the Euclidean case, except that $\mathcal{C} = \mathbb{R}^d$ is not finite.*

The deterministic k-median clustering problem is a special case of the probabilistic problem, where $m = n$ and, for each node v_i, we have $p_{ii} = 1$ and $p_{ij} = 0$ for all $j \neq i$. Note that in the literature it is typically assumed that $\mathcal{C} = \mathcal{X}$, i. e., the set of possible center locations is equal to the set of clustered points. Thus, our definition is a generalization of the typical definition. Additionally note that if for each node v_i we have one $j \in [m]$ with $p_{ij} = p_i$ and hence $p_{ij'} = 0$ for all $j' \neq j$, then the probabilistic k-median clustering can be immediately reduced to a weighted deterministic k-median clustering. Note that we use the notation cost_k^* in this case as well, then referring to the optimal weighted deterministic k-median-clustering cost.

Finally, we introduce *coresets* for the probabilistic k-median-clustering problem. Intuitively, a coreset is a small representation of the input point set that has similar clustering cost as the original points for any set of centers. It is similar to the original input, but smaller and the points are weighted in compensation. As the storage size of a probabilistic point set is influenced by the number of probabilistic points and by the size of the individual probability distributions, we are interested in coresets where both parameters are small. Note that the following formal definition of coresets is relatively general and allows that $\mathcal{C} \subsetneq \mathcal{X}$. In this case, the following coreset is a *weak* coreset.

Definition 2 (Coreset for Probabilistic k-Median Problem, [12]). *Given a set of nodes V, let $U := \{u_1, \ldots, u_s\} \subseteq V$ be a subset annotated with a positive weight function $w' : U \to \mathbb{R}_{\geq 0}$ and probability distributions $\mathcal{D}' := \{\mathcal{D}'_1, \ldots, \mathcal{D}'_s\}$ over $\mathcal{X}$ for all nodes in U. Given $k \in \mathbb{N}$, a finite set of κ possible center locations $\mathcal{C} \subseteq X$ and a precision parameter ε, $0 < \varepsilon \leq 1$, the set U is called (k, ε)-coreset of V for the probabilistic metric k-median-clustering problem if, for each $C \subseteq \mathcal{C}$ of size $|C| = k$, we have*

$$\left| \min_{\rho:U\to C} \mathbf{E}_{\mathcal{D}'}\left[\mathrm{cost}_{w'}(U, C, \rho)\right] - \min_{\rho:V\to C} \mathbf{E}_{\mathcal{D}}\left[\mathrm{cost}(V, C, \rho)\right] \right| \leq \varepsilon \cdot \min_{\rho:V\to C} \mathbf{E}_{\mathcal{D}}\left[\mathrm{cost}(V, C, \rho)\right].$$

The definition is verbatim for the Euclidean case, except that $\mathcal{C} = \mathbb{R}^d$. *If the nodes in* V *are weighted by a positive weight function* $w : V \to \mathbb{R}_{\geq 0}$, *then* $\mathrm{cost}(V, C, \rho)$ *is replaced by* $\mathrm{cost}_w(V, C, \rho)$.

In the following, for all u_i belonging to the coreset that we compute, we we denote the probability that u_i is realized at x_j by p'_{ij} for all $j \in [m]$. Further, we denote the total probability that u_i is realized by $p'_i := \sum_{j=1}^{m} p'_{ij}$.

Omitted Proofs. Note that due to space limitations, some proofs in Section 3 and all proofs in Section 4 are omitted and can be found in a technical report [31].

3 Coreset for Metric k-Median

Morphing Probability Distributions. We can compute a coreset for the probabilistic metric k-median-clustering problem by reducing the problem to the deterministic metric k-median-clustering problem. Imagine for the moment that the total realization probabilities p_i are uniform for all input nodes $v_i \in V$, i. e., we have $p_i = p$ for all $i \in [n]$ and some fixed probability p, $0 < p \leq 1$. Then, two probability distributions can be 'morphed' into one another by using the *earth mover distance* (EMD) as the cost needed to morph one probability distribution into another. It is defined as follows:

Definition 3. *Let* $v_{i'}$ *and* $v_{i''}$ *be two nodes in* V *with* $p_{i'} = p_{i''}$. *We say a mapping* $\varrho : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ *morphs* $v_{i'}$ *into* $v_{i''}$ *if it satisfies* $\sum_{x_j \in \mathcal{X}} \varrho(x_{j'}, x_j) = p_{i'j'}$ *and* $\sum_{x_j \in \mathcal{X}} \varrho(x_j, x_{j''}) = p_{i''j''}$ *for all* $x_{j'}, x_{j''} \in \mathcal{X}$. *The cost of* ϱ *is defined as* $\mathrm{morph}(v_{i'}, v_{i''}, \varrho) := \sum_{x_{j'} \in \mathcal{X}} \sum_{x_{j''} \in \mathcal{X}} \varrho(x_{j'}, x_{j''}) \cdot \mathrm{D}(x_{j'}, x_{j''})$. *The expected* earth mover distance $\mathrm{EMD}(v_{i'}, v_{i''})$ *between* $v_{i'}$ *and* $v_{i''}$ *is the minimum cost of a mapping that morphs* $v_{i'}$ *into* $v_{i''}$.

Theorem 1. *Let* $M = (X, D)$, V, $\mathcal{D}_1, \ldots, \mathcal{D}_n$, $\mathcal{C}$ *with* $|\mathcal{C}| = \kappa$, k *and* w *form a weighted probabilistic metric* k*-median problem instance, let* $0 < \delta, \varepsilon < 1$ *be given. A* (k, ε)*-coreset given by* U *and* $\mathcal{D}'_1, \ldots, \mathcal{D}'_n$ *can be computed in time* $\mathcal{O}(\varepsilon^{-10} knm \log^7(\kappa W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta)))$ *with error probability* δ. *The size of* U *is* $\mathcal{O}\left(\varepsilon^{-3} k^2 \cdot \log^3(\kappa W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta))\right)$, *and each probability distribution in* $\mathcal{D}'$ *assigns a positive probability to* $\mathcal{O}(\varepsilon^{-3} \log^2(\kappa W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta)))$ *points. The coreset construction requires* $\mathcal{O}\left(\varepsilon^{-6} k^2 \cdot \log^6(\kappa W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta)\right)$ *bits of space.*

Proof. Note that the cost to assign a node to a center in a probabilistic k-median clustering coincides with the EMD between the node and a node which has all its realization probability concentrated in the center. Furthermore, the EMD defined on a set of nodes with uniform total realization probabilities is a metric [35]. Let us assume for the moment that the realization probability of each node is a fixed value p and that the input is unweighted. Now, for each possible center location $x \in \mathcal{C}$, we consider an artificial node denoted by $\mathrm{node}(x)$ that is located at x

and has total realization probability p. We define a new metric space that has a point $\sigma(v_i)$ for each node $v_i \in V$ and a point $\sigma(\text{node}(x))$ for each artificial node $\text{node}(x)$. The distance between two points in the new metric space is given by the EMD between the corresponding original nodes. Let $\sigma(V)$ be the points resulting from the nodes in V, and let $\sigma(\text{node}(\mathcal{C}))$ be the points resulting from the artificial nodes. The computation of a (k, ε)-coreset of V for the probabilistic metric k-median problem is equivalent to the computation of a (k, ε)-coreset of $\sigma(V)$ for the deterministic metric k-median problem in which we use the new metric defined above and in which the k centers are points from $\sigma(\text{node}(\mathcal{C}))$.

Note that, although it is not explicitly mentioned in [11], the algorithm by Chen works for our more general version of the deterministic metric k-median clustering (where the set of clustered points and the set of possible center locations can be arbitrary finite subsets of points from the underlying metric space). The number of possible center locations has only a logarithmic influence on the coreset size. Thus, the computation of the (k, ε)-coreset can be done with the algorithm by Chen.

For that, it is necessary to be able to compute the EMD. Using an algorithm of Edmonds and Karp [13], the EMD between any two nodes $v_{i'}$ and $v_{i''}$ with $p_{i'} = p_{i''}$ can be computed in $\mathcal{O}(m^3)$ time. In order to improve the running time needed to compute the EMD between nodes and to have a small representation of a coreset node, we approximate the probability distribution $\mathcal{D}_i$ of each node $v_i \in V$ by computing a $(1, \varepsilon)$-coreset of v_i for the probabilistic 1-median problem. This means, for any center $c \in \mathcal{C}$, the expected cost to assign the node v_i to the center c is $(1\pm\varepsilon)$-approximated by the coreset. We can use these coreset instead of the original nodes because they only induce a small error to the overall clustering cost. The $(1, \varepsilon)$-coreset construction can directly be transferred from weighted deterministic 1-median clustering.

Transfering the above to the weighted case is straightforward. This also solves the problem of different realization probabilities, because every input can be adjusted such that all p_i are 1 by multiplying each p_{ij} with $1/p_i$ and at the same time, multiplying w_i by p_i. This does not change the objective function. □

4 Coreset for Euclidean k-Median

Coreset Construction. The algorithm by Chen [11] computes a coreset for the deterministic problem in two steps: (*i*) partitioning the nodes into disjoint subsets, (*ii*) drawing random sample nodes from each such subset. We add a third step, (*iii*) approximating the probability distribution of each sample node. In the following, we describe our realization of these steps by comparing them with the steps in [11].

Step (*i*) partitions the nodes into groups that are similar in terms of their locations and their contributions to the total clustering cost. Chen starts by computing a bicriteria approximation for the deterministic input point set. In the probabilistic case no such approximation is known and we give the first such

approximation algorithm here. We start with computing point representatives for each node.

- For every node $v_i \in V$, compute a point y_i that satisfies $\sum_{j=1}^{m} p_{ij} \cdot \mathrm{D}(x_j, y_i) \leq 2 \cdot \min_{x_{j'} \in X} \sum_{j=1}^{m} p_{ij} \cdot \mathrm{D}(x_j, x_{j'})$, i. e., y_i is a 2-approximation of the probabilistic 1-median for v_i. Let $Y := \{y_1, y_2, \ldots, y_n\}$ be the weighted set of the resulting n points where the weight of y_i is set to $w(y_i) := w(v_i)$.

Now, we compute a deterministic bicriteria approximation for the set of 1-medians interpreted as a deterministic clustering problem.

- Compute a set $\mathcal{A} \subseteq Y$ which is a center set of an $[\alpha, \beta]$-bicriteria approximation to $\mathrm{cost}_k^*(Y)$. That is, $\mathcal{A} := \{a_1, \ldots, a_\tau\}$ satisfies

$$\min_{\rho: Y \to \mathcal{A}} \mathbf{E}_{\mathcal{D}_Y} \left[\mathrm{cost}_w(Y, \mathcal{A}, \rho)\right] \leq \alpha \, \mathrm{cost}_k^*(Y) \ ,$$

  where $\alpha \geq 1$ is some constant, $\tau \leq \beta k$, $\beta = \mathcal{O}(\log(W/(w_{\min} \cdot p_{\min} \cdot \varepsilon)))$, $W := \sum_{v_i \in V} w(v_i) p_i$ is the expected total weight of the nodes, $w_{\min} := \min\{\min_{v_i \in V}\{w(v_i)\}, 1\}$ is either the minimum weight of a node in V or 1, and $p_{\min}$ is the smallest realization probability, i. e., $p_{\min} \leq p_{ij}$ for each $i \in [n]$ and each $j \in [m]$. Let $\sigma_Y : Y \to \mathcal{A}$ be an assignment satisfying $\mathbf{E}_{\mathcal{D}_Y} \left[\mathrm{cost}_w(Y, \mathcal{A}, \sigma_Y)\right] = \min_{\rho: Y \to \mathcal{A}} \mathbf{E}_{\mathcal{D}_Y} \left[\mathrm{cost}_w(Y, \mathcal{A}, \rho)\right]$. Note that σ_Y assigns each point in Y to the closest center in $\mathcal{A}$. Let $\sigma_V : V \to \mathcal{A}$ be the corresponding assignment for V such that $\sigma_V(v_i) = \sigma_Y(y_i)$ for all $i \in [n]$.

Next, we show that the computed set $\mathcal{A}$ is actually a $[3\alpha + 2, \beta]$-bicriteria approximation to probabilistic optimum $\mathrm{cost}_k^*(V)$. This enables us to transfer the partitioning step by Chen: We find a bound R on the average radius of the optimal cost of a probabilistic k-median clustering for V and an upper bound $2^\nu R$ on the distance between an arbitrary point in Y and its closest center in $\mathcal{A}$. The value of ν is $\lceil \log((9\alpha + 6)W/(w_{\min} \cdot p_{\min})) \rceil$. This ensures that the following Chen-like partitioning is indeed a partitioning into disjoint subsets:

- For all $\ell \in [\tau]$, define $Y_\ell \subseteq Y$ as the subset of points in Y that σ_Y assigns to a_ℓ, i. e., Y_ℓ is the set of all points whose closest point in $\mathcal{A}$ is a_ℓ (ties broken arbitrarily). Set $V_\ell := \{v_i | y_i \in Y_\ell\}$. Furthermore, for each $\ell \in [\tau]$ and each $h \in \{0, 1, \ldots, \nu\}$, let

$$Y_{\ell,h} := \begin{cases} Y_\ell \cap \mathcal{B}(a_\ell, R) & h = 0 \\ Y_\ell \cap [\mathcal{B}(a_\ell, 2^h R) \backslash \mathcal{B}(a_\ell, 2^{h-1} R)] & h \geq 1 \end{cases}$$

  be the *h-th ring set* for the center a_ℓ and the corresponding set Y_ℓ. Set $V_{\ell,h} := \{v_i | y_i \in Y_{\ell,h}\}$.

This gives us subsets of nodes which have relatively close 1-medians. For our purposes this partitioning is not sufficient, as the probability distributions of the nodes can have different variances and may not behave similarly according to the cost function. Thus, we further subdivide the ring sets according to the width $\sum_{x_j \in \mathcal{X}} (p_{ij}/p_i) D(x_j, y_i)$ of the probability distribution of v_i. Let $2^\mu R$ be an upper bound on the width of the probability distributions. The value for μ

is $\lceil \log((6\alpha+4)W/(w_{\min}\cdot p_{\min}))\rceil$. For each $\ell \in [\tau]$, each $h \in \{0,1,\ldots,\nu\}$, and each $a \in \{0,1,\ldots,\mu\}$, let

$$Y_{\ell,h,a} := \begin{cases} \{y_i \in Y_{\ell,h} \mid \sum_{x_j\in\mathcal{X}} (p_{ij}/p_i)\cdot \mathrm{D}(x_j,y_i) \le R\} & a=0 \\ \{y_i \in Y_{\ell,h} \mid 2^{a-1}R < \sum_{x_j\in\mathcal{X}} (p_{ij}/p_i)\cdot \mathrm{D}(x_j,y_i) \le 2^a R\} & a\ge 1 \end{cases}.$$

$V_{\ell,h,a}$ is the corresponding node set and the desired partitioning.

Step (*ii*) Chen samples uniformly from each of his ring sets. Due to the different total realization probabilities of nodes, our sampling is weighted and results in a coreset of individually weighted coreset points. In more detail: From each $V_{\ell,h,a}$, we sample a multiset $U_{\ell,h,a}$ with $s''' := \lceil c\cdot\varepsilon^{-2}\cdot[\log(1/\delta)+k\,(\log(k)+d\log(1/\varepsilon)+\log(\log(W/(w_{\min}\cdot p_{\min}\cdot\varepsilon))))]\rceil$ nodes, where c is a sufficiently large constant. The nodes are sampled with replacement, and, in each sample step, a node $v_i \in V_{\ell,h,a}$ is picked with probability $w(v_i)p_i/\sum_{v_{i'}\in V_{\ell,h,a}} w(v_{i'})p_{i'}$. We set the weight of the sample node v_i to $w'(v_i) = \sum_{v_{i'}\in V_{\ell,h,a}} w(v_{i'})p_{i'}/(p_i s''')$.We store all sampled nodes in the multiset $U := \{u_1,\ldots,u_s\} := \bigcup_{\ell=1}^{\tau}\bigcup_{h=0}^{\nu}\bigcup_{a=0}^{\mu} U_{\ell,h,a}$.

Step (*iii*) computes an approximated probability distribution Z_i for each coreset node v_i that assigns a positive probability to at most $\mathcal{O}(\varepsilon^{-2}\gamma^2 d\log^2(nmW)\cdot(d\log(1/\varepsilon)+\log(\gamma)\log(nmW)+\log(n/\delta)))$ points from $\mathcal{X}$.

Correctness. Due to space limitations, we only give a very rough sketch of the proof that our coreset construction is correct. The details can be found the full version [31]. As a basic tool to bound the number of sample nodes needed to reduce the error in the clustering cost below a certain threshold, we use the following result due to Haussler [23] (also used by Chen). The first problem that we have to overcome is that the input nodes are weighted due to the probabilities and the weight function, but Haussler's Lemma samples uniformly at random. We solve this by defining an appropriate bipartite mapping between the input nodes and a set of suitable chosen unweighted nodes. In order to transfer the properties of the unweighted sampling to our weighted setting, we need to weight all sample nodes individually (instead of giving all samples from one $V_{\ell,h,a}$ the same weight correlating to the number of nodes in $V_{\ell,h,a}$). We show that when we sample sufficiently many nodes from each $V_{\ell,a,h}$, the total error induced is bounded by $\xi \sum_{\text{all } V_{\ell,a,h}} \sum_{v_{i'}\in V_{\ell,a,h}} w(v_{i'})p_{i'}\Big(\mathrm{D}(Y(V_{\ell,a,h}),C)+\mathrm{diam}(Y(V_{\ell,a,h}))+\max_{y_i\in Y(V_{\ell,a,h})}\sum_{x_j\in\mathcal{X}}\frac{p_{ij}}{p_i}\cdot\mathrm{D}(x_j,y_i)\Big)$ with high probability. We show that the first of the three summands is bounded because we use 1-medians to calculate $\mathcal{A}$. Then, we prove that the second term is bounded because we partitioned into groups of nodes with closely located approximated 1-medians. The third term is bounded because the nodes also have probability distribution of similar widths.

Similar to Chen, we discretize the input space by defining 'huge balls' restricting the position of possible centers and subdivide these by a suitable grid in order to cope with the infinitely many possible centers from $\mathbb{R}^d$. Finding suitable huge balls turns out to be challenging, as the probability distributions can be quite

wide spread. Finally, we use known results from [30,24,11] to construct Y, $\mathcal{A}$ and Z_i for $i \in [n]$ efficiently and obtain the following theorem.

Theorem 2. *Let $M = (X, D)$ with D being the Euclidean distance function, V, $\mathcal{D}_1, \ldots, \mathcal{D}_n$, $\mathcal{C}$, k and w form a weighted probabilistic metric k-median problem instance, let $0 < \delta, \varepsilon < 1$ be given. A (k, ε)-coreset given by U and $\mathcal{D}'_1, \ldots, \mathcal{D}'_n$ can be computed in time $\mathcal{O}\left(knm \log\left(\log\left(W/(w_{\min} \cdot p_{\min} \cdot \varepsilon)\right)\right) \cdot \log\left(n/\delta\right)\right)$ with error probability δ. The size of U is $\mathcal{O}\left(\varepsilon^{-2}k^2 d \cdot \log^4\left(W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta)\right)\right)$, and each probability distribution in $\mathcal{D}'$ assigns $\mathcal{O}\left(\varepsilon^{-2}d \cdot \log^3\left(W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta)\right)\right)$ points from $\mathcal{X}$ a positive probability. The coreset construction has a space requirement of $\mathcal{O}\left(\varepsilon^{-4}k^2d^2 \cdot \log^8\left(W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta)\right)\right)$.*

5 Streaming Algorithm

Our coreset for the Euclidean k-median problem can be maintained in data streams by using the merge-and-reduce technique [7,22]. We assume that the set V is given as a stream of n weighted nodes in worst-case order. Each node v_i is given as a consecutive chunk that is a sequence of up to m point-probability pairs in worst case order representing the probability distribution $\mathcal{D}_i$ of the node v_i. Streaming algorithms are only allowed one sequential scan over the data and can only use an update time and local memory that is polylogarithmic in the size of the input stream. Our algorithm needs space that is polylogarithmic in n, m and $W/(w_{\min} \cdot p_{\min})$, and the update time per node is polylogarithmic in n and $W/(w_{\min} \cdot p_{\min})$, but it might be linear in m since the probability distribution of a node can be represented by m point-probability pairs.

Theorem 3. *Let $k \in \mathbb{N}$, $0 < \delta, \varepsilon \leq 1$, V, $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ over $\mathcal{X}$ with $|\mathcal{X}| = m$ be given in form of a data stream. One can compute a weighted subset $U := \{u_1, \ldots, u_s\} \subseteq V$ and a set of probability distributions $\mathcal{D}' := \{\mathcal{D}'_1, \ldots, \mathcal{D}'_s\}$ such that the nodes in U together with the probability distributions $\mathcal{D}'$ build a (k, ε)-coreset of V for the probabilistic k-median-clustering problem with probability $1 - \delta$. The size of U is $\mathcal{O}\left(\varepsilon^{-2}k^2 d \cdot \log^7\left(W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta)\right)\right)$, where each probability distribution in $\mathcal{D}'$ assigns $\mathcal{O}\left(\varepsilon^{-2}d \cdot \log^6\left(W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta)\right)\right)$ points from $\mathcal{X}$ a positive probability.*
The streaming algorithm requires $\mathcal{O}\left(\varepsilon^{-4}k^2d^2 \cdot \log^{14}\left(W/(w_{\min} \cdot p_{\min} \cdot \varepsilon\delta)\right)\right)$ space and has an update time per node of $\mathcal{O}\left(\varepsilon^{-4}k^3d^2m \cdot \log^{14}\left(W/w_{\min} \cdot p_{\min} \cdot \varepsilon\delta\right)\right)$.

6 Applications

Coresets can be used to speed up approximation algorithms by computing a solution on the small coreset instead of the possibly huge original input set. For an α-approximation algorithm, this leads to an $\alpha(1 + \varepsilon)$-approximation. In the metric case, the only approximation algorithm known so far is the reduction to the deterministic k-median problem by Cormode and McGregor [19]. Using their algorithm, we get a constant-factor approximation. However, the computation

of our coreset has roughly the same running time as the algorithm in [19]. In the Euclidean case, no approximation algorithm is known so far. By using our coreset, we get a randomized $(1+\varepsilon)$-approximation in time roughly $(W/(w_{\min} \cdot p_{\min}))^{\log^5(W/(w_{\min} \cdot p_{\min}))}$ for constant ε, k and d.

As soon as algorithms with better approximation guarantee (in the metric case) or lesser dependence on ε, k or d (in the Euclidean case) are known, even with running time exponential in the expected total weight W or in the term $W/(w_{\min} p_{\min})$, the speed-up directly leads to polynomial-time versions.

Finally, the coreset construction can be beneficial in itself as it enables queries to the clustering cost in a setting where data comes in a stream and can only be stored in sketches.

Acknowledgments. The authors thank the anonymous referees for their detailed and useful comments, especially for suggesting to try to extend Theorem 1 to the non-uniform case and for pointing out that the proof can be shortened.

References

1. Agarwal, P.K., Har-Peled, S., Varadarajan, K.R.: Approximating extent measures of points. Journal of the ACM 51(4), 606–635 (2004)
2. Aggarwal, C.C., Yu, P.S.: A survey of uncertain data algorithms and applications. IEEE Transactions on Knowledge and Data Engineering 21(5), 609–623 (2009)
3. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. Journal of the ACM 45(5), 753–782 (1998)
4. Arora, S., Raghavan, P., Rao, S.: Approximation schemes for euclidean k-medians and related problems. In: Proc. of the 30th STOC, pp. 106–113 (1998)
5. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM Journal on Computing 33(3), 544–562 (2004)
6. Bădoiu, M., Har-Peled, S., Indyk, P.: Approximate clustering via core-sets. In: Proc. of the 34th STOC, pp. 250–257 (2002)
7. Bentley, J.L., Saxe, J.B.: Decomposable searching problems I: Static-to-dynamic transformation. Journal of Algorithms 1(4), 301–358 (1980)
8. Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location problems. SIAM Journal on Computing 34(4), 803–824 (2005)
9. Charikar, M., Guha, S., Tardos, É., Shmoys, D.B.: A constant-factor approximation algorithm for the k-median problem. Journal of Computer and System Sciences 65(1), 129–149 (2002)
10. Chau, M., Cheng, R., Kao, B., Ng, J.: Uncertain data mining: An example in clustering location data. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) PAKDD 2006. LNCS (LNAI), vol. 3918, pp. 199–204. Springer, Heidelberg (2006)
11. Chen, K.: On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. SIAM Journal on Computing 39(3), 923–947 (2009)
12. Cormode, G., McGregor, A.: Approximation algorithms for clustering uncertain data. In: Proc. of the 27th PODS, pp. 191–200 (2008)
13. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM 19(2), 248–264 (1972)

14. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. 2nd ACM SIGKDD, pp. 226–231 (1996)
15. Feldman, D., Monemizadeh, M., Sohler, C.: A PTAS for k-means clustering based on weak coresets. In: Proc. of the 23rd SoCG, pp. 11–18 (2007)
16. Forgey, E.: Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. Biometrics 768(21) (1965)
17. Frahling, G., Sohler, C.: Coresets in dynamic geometric data streams. In: Proc. of the 37th STOC, pp. 209–217 (2005)
18. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: Theory and practice. IEEE Transactions on Knowledge and Data Engineering 15(3), 515–528 (2003)
19. Guha, S., Munagala, K.: Exceeding expectations and clustering uncertain data. In: Proc. of the 28th PODS, pp. 269–278 (2009)
20. Günnemann, S., Kremer, H., Seidl, T.: Subspace clustering for uncertain data. In: Proc. of the SIAM International Conference on Data Mining, pp. 385–396 (2010)
21. Har-Peled, S., Kushal, A.: Smaller coresets for k-median and k-means clustering. Discrete & Computational Geometry 37(1), 3–19 (2007)
22. Har-Peled, S., Mazumdar, S.: On coresets for k-means and k-median clustering. In: Proc. of the 36th STOC, pp. 291–300 (2004)
23. Haussler, D.: Decision theoretic generalizations of the pac model for neural net and other learning applications. Information & Computation 100(1), 78–150 (1992)
24. Indyk, P.: Sublinear time algorithms for metric space problems. In: Proc. of the 31st STOC, pp. 428–434 (1999)
25. Jain, K., Mahdian, M., Saberi, A.: A new greedy approach for facility location problems. In: Proc. of the 34th STOC, pp. 731–740 (2002)
26. Jain, K., Vazirani, V.V.: Primal-dual approximation algorithms for metric facility location and k-median problems. In: Proc. of the 40th FOCS, pp. 2–13 (1999)
27. Kolliopoulos, S.G., Rao, S.: A nearly linear-time approximation scheme for the euclidean k-median problem. SIAM Journal on Computing 37(3), 757–782 (2007)
28. Kriegel, H.P., Pfeifle, M.: Density-based clustering of uncertain data. In: Proc. of the 11th ACM SIGKDD, pp. 672–677 (2005)
29. Kriegel, H.P., Pfeifle, M.: Hierarchical density-based clustering of uncertain data. In: IEEE International Conference on Data Mining (ICDM), pp. 689–692 (2005)
30. Kumar, A., Sabharwal, Y., Sen, S.: Linear-time approximation schemes for clustering problems in any dimensions. Journal of the ACM 57(2) (2010)
31. Lammersen, C., Schmidt, M., Sohler, C.: Probabilistic k-median. Tech. rep., Report CGL-TR-02, STREP project Computational Geometric Learning (2011), `http://cgl.uni-jena.de/pub/Publications/WebHome/CGL-TR-02.pdf`
32. Lloyd, S.P.: Least squares quantization in pcm. IEEE Transactions on Information Theory 28(2), 129–136 (1982)
33. Mettu, R.R., Plaxton, C.G.: Optimal time bounds for approximate clustering. Machine Learning 56(1-3), 35–60 (2004)
34. Ngai, W.K., Kao, B., Chui, C.K., Cheng, R., Chau, M., Yip, K.Y.: Efficient clustering of uncertain data. In: Proc. of the 6th IEEE ICDM, pp. 436–445 (2006)
35. Rubner, Y., Tomasi, C., Guibas, L.J.: A metric for distributions with applications to image databases. In: Proc. of the 6th ICCV, pp. 59–66 (1998)
36. Xu, H., Li, G.: Density-based probabilistic clustering of uncertain data. In: Proc. of the 1st CSSE, vol. 4, pp. 474–477 (2008)

Linear Time Approximation for Dominating Sets and Independent Dominating Sets in Unit Disk Graphs

Guilherme D. da Fonseca[1], Celina M.H. de Figueiredo[2], Vinícius G.P. de Sá[3], and Raphael Machado[4]

[1] Dept. de Informática Aplicada, UniRio, Brazil
[2] PESC/COPPE, UFRJ, Brazil
[3] Dept. de Ciência da Computação, UFRJ, Brazil
[4] Instituto Nacional de Metrologia, Qualidade e Tecnologia, Brazil

Abstract. A unit disk graph is the intersection graph of n congruent disks in the plane. Dominating sets in unit disk graphs are widely studied due to their application in wireless ad-hoc networks. Since the minimum dominating set problem for unit disk graphs is NP-hard, several approximation algorithms with different merits have been proposed in the literature. On one extreme, there is a linear time 5-approximation algorithm. On another extreme, there are two PTAS whose running times are polynomials of very high degree. We introduce a linear time approximation algorithm that takes the usual adjacency representation of the graph as input and attains a 44/9 approximation factor. This approximation factor is also attained by a second algorithm we present, which takes the geometric representation of the graph as input and runs in $O(n \log n)$ time, regardless of the number of edges. The analysis of the approximation factor of the algorithms, both of which are based on local improvements, exploits an assortment of results from discrete geometry to prove that certain graphs cannot be unit disk graphs. It is noteworthy that the dominating sets obtained by our algorithms are also independent sets.

1 Introduction

A *unit disk graph* G is a graph whose n vertices can be mapped to points in the plane and whose m edges are defined by pairs of points within Euclidean distance at most 1 from one another. Alternatively, one can regard the vertices of G as mapped to coplanar disks of unit diameter, so that two vertices are adjacent whenever the corresponding disks intersect.

A *dominating set* D is a subset of the vertices of a graph such that every vertex not in D is adjacent to some vertex in D. An *independent dominating set* is a dominating set which is also an independent set. Note that any maximal independent set is an independent dominating set.

Dominating sets in unit disk graphs are widely studied due to their application in wireless ad-hoc networks [12]. Since it is NP-hard to compute the

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 82–92, 2013.

minimum dominating set of a unit disk graph [2], a number of approximation algorithms have been proposed [3,5,9,10,12,15,19]. Such algorithms are of two main types. *Graph-based algorithms* receive as input the adjacency representation of the graph and assume no knowledge of the point coordinates, whereas *geometric algorithms* work in the Real RAM model of computation and receive solely the vertex coordinates as input[1]. Thus far these two types of algorithms have been tackled separately in the literature for the dominating set problem in unit disk graphs. In this paper, we introduce approximation algorithms of both types, benefiting from the same approximation factor analysis.

Previous Algorithms. A graph-based 5-approximation algorithm that runs in $O(n + m)$ time was presented in [12]. The algorithm computes a maximal independent set, which turns out to be a 5-approximation because unit disk graphs contain no $K_{1,6}$ as induced subgraphs.

Polynomial-time approximation schemes (PTAS) were first presented as geometric algorithms [10] and later as graph-based algorithms [15]. Also, a graph-based PTAS for the more general disk graphs is proposed in [9]. Unfortunately, the complexities of the existing PTAS are high degree polynomials. For example, the PTAS presented in [15] takes $O(n^{225})$ time to obtain a 5-approximation (using the analysis from [3]). Although its analysis is not tight, the running time is too high even for moderately large graphs. The reason is that these PTAS invoke a subroutine that verifies (by brute force) whether a graph admits a dominating set with k vertices. Such subroutine is applied to several subgraphs, and the value of k grows as the approximation error decreases. A similar strategy is used to obtain a PTAS for the minimum independent dominating set [11].

The lack of fast algorithms with approximation factors less than 5 was noticed in [3], where geometric algorithms with approximation factors of 3 and 4 and running times respectively $O(n^{18})$ and $O(n^9)$ were presented. While a significant step towards approximating large instances, those algorithms require the geometric representation of the graph, and the running times are polynomials of rather high degrees. Linear and near-linear time approximation algorithms constitute an active topic of research, even for problems that can be solved exactly in polynomial time, such as maximum flow and maximum matching [1,18].

It is useful to contrast the minimum dominating set problem with the maximum independent set problem. While a maximal independent set is a 5-approximation to both problems, it is easy to obtain a geometric 3-approximation to the maximum independent set problem in $O(n \log n)$ time [14]. In the graph-based version, a related strategy takes roughly $O(n^5)$ time, though. No similar results are known for the minimum dominating set problem.

Packing problems usually arise in situations where one wants to enclose non-overlapping objects as densely as possible into recipients of given shape. Unit disk graphs are subject to a number of packing constraints that limit the size of independent sets (which correspond to disjoint disks) as a function of the distance

[1] The Real RAM model is a technical necessity, otherwise storing the coordinates of the vertices would require an exponential number of bits [13].

between vertices. The existing PTAS for dominating sets in unit disk graphs are based on some of these packing constraints, such as the *bounded growth property*: the size of an independent set formed by vertices within distance r of a given vertex is at most $(1 + 2r)^2$. Note, however, that the bounded growth property is not tight. For example, for $r = 1$, it gives an upper bound of 9 vertices where the actual maximum size is 5. Since the bounded growth property is strongly connected to the problem of packing circles in a circle [7], obtaining exact values for all r seems unlikely.

Our Contribution. Our main result consists of two approximation algorithms: a graph-based algorithm, which runs in linear $O(n + m)$ time, and its geometric counterpart, which runs in $O(n \log n)$ time in the Real RAM model, regardless of the number of edges.

The approximation factor of our algorithms is 44/9. The strategy for both algorithms is to construct a 5-approximate solution using the algorithm from [12] and to perform subsequent local improvements to that initial dominating set. Our main lemma (Lemma 7) uses forbidden subgraphs to show that a solution that admits no local improvement is a 44/9-approximation. Since the dominating sets produced by our algorithms are independent sets, the same approximation factor holds for the independent dominating set problem.

Proving that a certain graph is *not* a unit disk graph (and is therefore a forbidden induced subgraph) is no easy feat[2]. We make use of an assortment of results from discrete geometry in order to prove properties of unit disk graphs that are interesting *per se*. For example, we use universal covers and disk packings to show that the neighborhood of a clique in a unit disk graph contains at most 12 independent vertices. These properties, along with a tighter version of the bounded growth property, allow us to show that certain graphs are not unit disk graphs. Consequently, our algorithms employ a broader set of forbidden subgraphs, including, but not being limited to, the $K_{1,6}$.

2 Forbidden Subgraphs

In this section, we introduce some lemmas about the structure of unit disk graphs. These lemmas will be applied to prove our approximation factor in Section 3. We start by stating three previous results from the area of discrete geometry. The first lemma comes from the study of universal covers (for a recent survey see [8]).

Lemma 1 (Pál [16]). *If a set of points P has diameter 1, then P can be enclosed by a circle of radius $1/\sqrt{3}$.*

Packing congruent disks in a circle is a well studied problem. Exact bounds on the radius of the smallest circle packing k congruent disks are known for some small values of k, namely $k \leq 13$ and $k = 19$ [7]. The bound for $k = 13$ will be useful to us.

[2] The fastest known algorithm to decide whether a given graph is a unit disk graph is doubly exponential [17].

Lemma 2 (Fodor [7]). *The radius of the smallest circle enclosing* 13 *points with mutual distances at least* 1 *is* $(1+\sqrt{5})/2$.

The *density* of a packing is the ratio between the covered area and the total area. The following upper bound is useful when no exact bound is known.

Lemma 3 (Fejes Tóth [6]). *Every packing of two or more congruent disks in a convex region has density at most* $\pi/\sqrt{12}$.

Given a graph $G = (V, E)$ and a vertex $v \in V$, let $N(v)$ denote the *open neighborhood* of v and let $N[v] = N(v) \cup \{v\}$ denote the *closed neighborhood* of v. More generally, the *open r-neighborhood* of a vertex v is the set of vertices w such that the distance between v and w in G is exactly r, while the *closed r-neighborhood* of a vertex v is the set of vertices w such that the distance between v and w in G is at most r. For a set $S \subseteq V$, we let $N_S(v) = N(v) \cap S$ and $N_S[v] = N[v] \cap S$. Finally, given a subgraph G' of G, the *closed neighborhood* of G' is the set of vertices that belong to the closed neighborhood of some vertex of G'. The following two lemmas concern neighborhoods in unit disk graphs.

Lemma 4. *The closed neighborhood of a clique in a unit disk graph contains at most* 12 *independent vertices.*

Proof. By Lemma 1, the points which define a clique in a unit disk graph are contained inside a circle of radius $1/\sqrt{3}$. Therefore, the points corresponding to the closed neighborhood of such clique are contained inside a circle of radius $1 + (1/\sqrt{3})$. By Lemma 2, we have that a circle enclosing 13 points with mutual distances at least 1 has radius at least $(1+\sqrt{5})/2$. Since $(1+\sqrt{5})/2 > 1+(1/\sqrt{3})$, the lemma follows. □

Lemma 5. *Given an integer $r \geq 1$, the closed r-neighborhood of a vertex in a unit disk graph contains at most $\lfloor \pi(2r+1)^2/\sqrt{12} \rfloor$ independent vertices.*

Proof. All n disks of diameter 1 corresponding to the closed r-neighborhood of a vertex v must be enclosed by a circle W of radius $(2r+1)/2$ centered on v. Each disk of diameter 1 has area $\pi/4$ and W has area $(2r+1)^2\pi/4$. Using Lemma 3, we have $(n\ \pi/4)/((2r+1)^2\pi/4) \leq \pi/\sqrt{12}$, and the lemma follows. □

We say that a graph G is (k, ℓ)*-pendant* if there is a vertex v in G with k vertices of degree 1 in the open neighborhood of v and ℓ vertices of degree 1 in the open 2-neighborhood of v. We refer to v as a *generator* of the (k, ℓ)-pendant graph. The following lemma bounds the value of the parameter ℓ for a $(4, \ell)$-pendant unit disk graphs.

Lemma 6. *If G is a $(4, \ell)$-pendant unit disk graph, then $\ell \leq 8$.*

Proof. Let v be a generator of G. Since $K_{1,6}$ is a forbidden induced subgraph [12] and v has 4 neighbors of degree 1, we have that the remaining neighbors of v together with v itself form a clique. By Lemma 4, we have that $4 + \ell \leq 12$. □

3 Approximation Algorithms

In this section, we present our approximation algorithms. The key property to analyze the approximation factor is presented in Lemma 7, while the running time analyses are presented in Sections 3.1 and 3.2.

Hereafter, let $G = (V, E)$ be a unit disk graph, and let $D \subseteq V$ be an independent dominating set of G. If $v \in D$ and $uv \in E$, we say that v *dominates* u and, conversely, that u *is dominated* by v.

As already mentioned, unit disk graphs are free of induced $K_{1,6}$. Therefore, at most 5 vertices of D may belong to the closed neighborhood of any given vertex $v \in V$. A *corona* is a set $C \subseteq D$ consisting of exactly 5 neighbors of some vertex $c \in V \setminus D$. Such a vertex c is called a *core* of the corona C, and it is not necessarily unique. Notice that the subgraph induced by a corona C and a corresponding core c is a star, i.e. a graph formed by an independent set and a universal vertex.

A corona C is said to be *reducible* if there is a core c of C such that $D \cup \{c\} \setminus C$ is a dominating set. If no such core exists, C is dubbed *irreducible*. Given a reducible corona C and a corresponding core c, we refer to the operation that converts D into the smaller dominating set $D \cup \{c\} \setminus C$ as a *reduction*.

Lemma 7. *Let $G = (V, E)$ be a unit disk graph, D an independent dominating set in G, and D^* a minimum dominating set of G. If D contains no reducible coronas, then $\rho = |D|/|D^*| \leq 44/9$.*

Proof. We use a charging argument to bound the ratio between the cardinalities of D and D^*. Consider that each vertex $u \in D$ splits a *unit charge* evenly among the vertices in the closed neighborhood $N_{D^*}[u]$. The function $f : D^* \to (0, 5]$ below corresponds to the total charges assigned to each vertex $v^* \in D^*$, accumulating the (fractional) charges that v^* received from the vertices in $N_D[v^*]$:

$$f(v^*) = \sum_{u \in N_D[v^*]} \frac{1}{|N_{D^*}[u]|}. \tag{1}$$

Note that, since D and D^* are dominating sets, neither $N_{D^*}[u]$ nor $N_D[v^*]$ are ever empty, and $f(v^*) \leq N_D[v^*]$. Such function f allows us to write the cardinality of D as

$$|D| = \sum_{v^* \in D^*} f(v^*).$$

Since

$$\rho = \frac{|D|}{|D^*|} = \frac{\sum_{v^* \in D^*} f(v^*)}{|D^*|}$$

is precisely the average value of $f(\cdot)$ over the elements of D^*, we obtain the desired bound $\rho \leq 44/9$ by showing that the existence of vertices c^* in D^* with $f(c^*) > 44/9$ is counterbalanced by a sufficiently large number of vertices x^* in D^* with $f(x^*) \leq 4$.

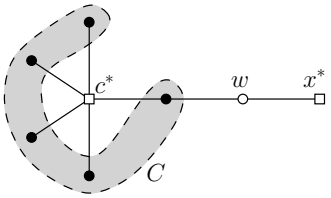


Fig. 1. Figure for the proof of Lemma 7. A proper, induced subgraph, where squares were used for a subset of D^*, solid circles for a subset of D (the corona C) and hollow circles for vertices not in $D \cup D^*$. Vertices w and x^* are respectively witness and reliever of core c^*.

Before we continue, we observe that $f(c^*) > 44/9$ means exactly $f(c^*) = 5$, because the sum in (1) has at most 5 terms, all of which are of the form $1/i$ for integer $i \geq 1$.

Thus, let c^* be a vertex in D^* with $f(c^*) = 5$. Clearly, $c^* \notin D$, otherwise $f(c^*) \leq |N_D[c^*]| = 1$, because D is an independent set. Moreover, c^* must have exactly 5 neighbors in D, since a greater number of neighbors in D would imply the existence of an induced $K_{1,6}$ in G, which is not possible, and a lesser number would imply $f(c^*) \leq |N_D[c^*]| \leq 4$, a contradiction. Vertex c^* is therefore a core.

Now let $C \subset D$ be the corona of which c^* is a core. Since C is irreducible (by the hypothesis of the lemma), there must be a vertex $w \in V \setminus (C \cup \{c^*\})$, such that:

(i) w is only dominated, in D, by vertices that belong to C; and
(ii) w is not adjacent to c^*.

We call w a *witness* of c^* (meaning the corona having c^* as a core fails to be reducible due to w). Now, for all $u \in C$, it holds that the only vertex in $N_{D^*}[u]$ must be the very core c^*, otherwise the contribution of u in (1) would be at most $1/2$, and $f(c^*)$ would be at most $9/2 < 5$, a contradiction. In particular, the witness w cannot belong to D^*. But D^* is a dominating set, so there must exist a vertex $x^* \in D^*$ that is adjacent to w. We call x^* a *reliever* of c^*. Figure 1 illustrates this situation.

We now show that $|N_D[x^*]| \leq 4$. For the sake of contradiction, assume $|N_D[x^*]| > 4$. Because G is free of induced $K_{1,6}$, such number must be exactly 5, so that x^* is the core of a corona $C' \subset D$. However, due to (i) above, $N_{C'}(w) = \emptyset$, hence $C' \cup \{w\}$ is an independent set of G, constituting, along with the core x^*, an induced $K_{1,6}$ in G, a contradiction. Since $f(x^*) \leq |N_D[x^*]|$, we have $f(x^*) \leq 4$.

We have just shown that the existence of a vertex c^* in D^* with $f(c^*) = 5$ implies the existence of a vertex $x^* \in D^*$ such that $f(x^*) \leq 4$. Were this correspondence one-to-one, we would be able to state that the average of $f(\cdot)$ over the elements of D^* was no greater than 4.5. Unfortunately, this correspondence is not necessarily one-to-one, as illustrated in Figure 2.

Still, the lemmas in Section 2 allow us to bound the ratio between the number of vertices c^* with $f(c^*) = 5$ and the number of vertices x^* for which the values

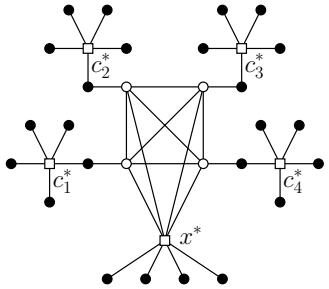


Fig. 2. A unit disk graph where 4 distinct cores $c_1^*, \ldots, c_4^*$ share the same reliever x^*

of f are significantly lower. Let $x^* \in D^* \setminus D$ be a reliever. In order to obtain the claimed bound, we consider two cases according to the size of $N_D[x^*]$:

(i) $|\boldsymbol{N_D[x^*]}| \leq \mathbf{3}$. By Lemma 5, the closed 4-neighborhood of x^* contains at most 73 independent vertices. Since each corona contains 5 independent vertices (only adjacent to their cores), at most $\lfloor 73/5 \rfloor = 14$ coronas may share a common reliever[3]. Let $c_1^*, \ldots, c_{14}^*$ denote the cores of such coronas. If $|N_D[x^*]| \leq 3$, then the average value of $f(\cdot)$ among $x^*, c_1^*, \ldots, c_{14}^*$ is at most

$$\frac{3 + 14 \cdot 5}{15} < 4.867.$$

(ii) $|\boldsymbol{N_D[x^*]}| = \mathbf{4}$. By Lemma 6, if $|N_D[x^*]| = 4$, then at most 8 cores $c_1^*, \ldots, c_8^*$ may have x^* as their common reliever, for otherwise we obtain a $(4, 9)$-pendant graph, which cannot be a unit disk graph, . Thus, the average value of $f(\cdot)$ among $x^*, c_1^*, \ldots, c_8^*$ is at most

$$\frac{4 + 8 \cdot 5}{9} = 44/9 = 4.888\ldots.$$

The worst case is therefore the one in which $|N_D[x^*]| = 4$, for an average $\rho = 44/9$, and the lemma follows. □

3.1 Graph-Based Algorithm

By Lemma 7, an independent dominating set with no reducible coronas is a 44/9-approximation to the minimum dominating set. In this section, we describe how to obtain such set in linear time given the adjacency list representation of the graph.

We can easily compute a maximal independent set D, which is a 5-approximation to the minimum dominating set [12], in $O(n+m)$ time. An independent dominating set with no reducible coronas can then be obtained by iteratively performing reductions. However, naively performing such reductions leads to a

[3] We would like to thank an anonymous referee for this simplified argument.

running time of $O(n^2m)$, since (i) there are $O(n)$ candidates to being the core of a reducible corona, (ii) detecting whether a vertex v is in fact the core of a reducible corona by inspecting the 3-neighborhood of v takes $O(m)$ time, and (iii) we may need to reduce a total of $O(n)$ coronas. Fortunately, the following algorithm modifies the set D and returns an independent dominating set with no reducible coronas in $O(n + m)$ time.

(1) For each vertex $v \in V \setminus D$, compute $N_D(v)$.
(2) For each vertex $v \in V \setminus D$, if $|N_D(v)| = 5$, add $N_D(v)$ to the list of coronas $\mathcal{C}$ (unless it is already there).
(3) Let $B \leftarrow \emptyset$. For each corona $C \in \mathcal{C}$, if there is a vertex c such that $D \cup \{c\} \setminus C$ is a dominating set, then add c to the set B.
(4) Choose a maximal subset B' of B such that the pairwise distance of the vertices in B' is at least 5.
(5) For each vertex $c \in B'$, perform a reduction $D \leftarrow D \cup \{c\} \setminus N_D(c)$.
(6) Repeat all the steps above until $B' = \emptyset$.

The algorithm is correct since all changes made to D along its execution preserve the property that D is an independent dominating set. Notice that, in step (4), we only reduce coronas that are sufficiently far from each other, in order to guarantee that we do not reduce a corona that may have ceased to be reducible due to a previous reduction. Moreover, the algorithm always terminates because the size of D decreases at every iteration, except for the last one. Next, we show that the running time is $O(n + m)$.

Step (1) can be easily implemented to run in $O(n + m)$ time. To execute step (2) in $O(n + m)$ time, we must determine in constant time whether a corona is already in the list $\mathcal{C}$. This can be achieved by indexing each corona C by an arbitrary vertex $v \in C$ (say, the one with the lowest index), and by storing with v a list of coronas that are in $\mathcal{C}$ and whose index is v. Note that, because of the packing constraints inherent to unit disk graphs, the number of coronas that contain a given vertex is $O(1)$.

Step (3) can be implemented as follows (for each corona $C \in \mathcal{C}$):

(3a) Let S_1 be the union of the open neighborhoods of the 5 vertices in C.
(3b) Let S_2 be the subset of S_1 containing only the vertices v with $N_D(v) \subseteq C$.
(3c) Let S_3 be the intersection of the closed neighborhoods $N[v]$ of all $v \in S_2 \cup C$.
(3d) If $S_3 \neq \emptyset$, then add an arbitrary vertex of S_3 to the set B.

The steps above take $O(n + m)$ total time when executed for all coronas $C \in \mathcal{C}$, because the number of coronas that contain or are adjacent to a given vertex is also $O(1)$ by packing constraints.

It is easy to perform steps (4) and (5) in linear time. It remains to show that the whole process is only repeated for a constant number of iterations. Let $\mathcal{C}_1, \ldots, \mathcal{C}_k$ denote the set of reducible coronas at each iteration of the algorithm with $\mathcal{C}_k = \emptyset$. Note that the reductions performed in step (5) never create a new reducible corona. Therefore $\mathcal{C}_1 \supset \cdots \supset \mathcal{C}_k$. Let C denote a corona that was reduced in the last iteration k. If C was not reduced during a previous iteration

$i < k$, then another corona within constant distance from C was reduced at that very iteration i. Since, again by packing constraints, the maximum number of coronas within constant distance from C is itself a constant, we have $k = O(1)$.

The following theorem summarizes the result from this section.

Theorem 1. *Given the adjacency list representation of a unit disk graph with n vertices and m edges, we can find a 44/9-approximation to the minimum dominating set in $O(n + m)$ time.*

3.2 Geometric Algorithm

In this section, we describe how to obtain an independent dominating set with no reducible corona in $O(n \log n)$ time given the geometric representation of the graph. The input for our algorithm is a set P of n points. Without loss of generality, we assume that the corresponding unit disk graph is connected (otherwise, we can compute the connected components in $O(n \log n)$ time using a Delaunay triangulation [4]). We use terms related to vertices of the graph and to the corresponding points interchangeably. For example, we say a set of points is independent if all pairwise distances are greater than 1.

We want the points of P to be structured in suitable fashion. Thus, as a preliminary step, we sort the points by x-coordinates and by y-coordinates separately (such orderings will also be useful later on), and we partition the points of P according to an infinite grid with unitary square cells by performing two sweeps on the sorted points. Without loss of generality, we assume that no point lies on the boundary of a grid cell. Given $p \in P$, let $\sigma(p)$ denote the grid cell that contains p. We refer to the set of at most 8 non-empty grid cells surrounding a cell Q as the *open vicinity* of Q, denoted $N(Q)$, and to the union of Q and its open vicinity as the *closed vicinity* of Q, denoted $N[Q]$. Note that a point p can only be adjacent to points in the closed vicinity of $\sigma(p)$, that is, $N[p] \subset N[\sigma(p)]$. Each point $p \in P$ stores a pointer to its containing cell $\sigma(p)$. Also, each cell stores the list of points it contains and pointers to the cells in its open vicinity. Since the diameter of the point set is at most n due to the graph connectivity, this whole step can be done in $O(n \log n)$ time.

We are now able to show how to compute a maximal independent set D efficiently. We begin by making a copy P' of P, and by letting $D \leftarrow \emptyset$. Then we repeat the two following steps while set P' is non-empty. (i) Choose an arbitrary point $p \in P'$ and add it to set D. (ii) For each point p' in the closed vicinity of $\sigma(p)$, remove p' from P' if $\|pp'\| \leq 1$. When P' becomes empty, D is an independent dominating set. This process takes $O(n)$ time due to the two following facts. First, a cell belongs to the closed vicinity of a constant number of cells. Second, the number of points inside a cell with pairwise distances greater than 1 is at most a constant.

We now have that D is a maximal independent set, and therefore a 5-approximation to the minimum dominating set. Next, we show how to modify D in order to produce an independent dominating set with no reducible corona, therefore a 44/9-approximation to the minimum dominating set. The algorithm

follows a close parallel to the one in Section 3.1, but each step takes no more than $O(n \log n)$ time using the geometric representation of the graph.

Since D is an independent set and a grid cell Q has side 1, a simple packing argument shows that $|D \cap Q| \leq 4$. We store the set $D \cap Q$ in the corresponding cell Q. In order to compute $N_D(p)$, it suffices to inspect at most the 36 points in $D \cap Q$ for $Q \in N[\sigma(p)]$. We can then build a list of coronas in $O(n)$ time (steps (1) and (2) of Section 3.1).

To perform step (3), we need to find out whether there is a vertex c such that $D \cup \{c\} \setminus C$ is a dominating set, for each corona $C = \{p_1, \ldots, p_5\}$. First, we make S_1 the union of $N_D(p_i)$ for $1 \leq i \leq 5$. Then, we make S_2 the subset of S_1 containing only the points p with $N_D(p) \subseteq C$. These first two steps are similar to steps (3a) and (3b) in Section 3.1. The remaining sub-steps of step (3) are significantly different, though.

We proceed by making $S_3 = S_2 \cup C$. We need to determine whether there is a point $p \in S_3$ that is adjacent to all points in S_3. For each $p \in S_3$, let $\beta(p)$ denote the disk of radius 1 centered at p. Let R denote the convex region defined by the intersection of $\beta(p)$ for all $p \in S_3$. A point p is adjacent to all points in S_3 if and only if $p \in R$. We can compute the region R in $O(|S_3| \log |S_3|)$ time using divide-and-conquer in a manner analogous to half-plane intersection [4]. We can then test whether each point $p \in S_3$ belongs to the region R in logarithmic time using binary search (remember the points were previously sorted). If there is at least one point $p \in S_3 \cap R$, then we add p to the set B. Therefore, the whole step (3) takes $O(n \log n)$ time.

In step (4) of the geometric algorithm, we choose an alternative set $B' \subset B$ which can be computed in $O(n)$ time as follows. For each $p \in B$, we add p to B' and then remove from B all points that are contained in the cells within Euclidean distance at most 4 of $\sigma(p)$. Since by packing constraints there are $O(1)$ points in the intersection of D and the closed vicinity of a cell, we can easily perform step (5) in $O(n)$ time.

We summarize the result from this section in the following theorem.

Theorem 2. *Given a set of n points representing a unit disk graph, we can find a 44/9-approximation to the minimum dominating set in $O(n \log n)$ time in the Real RAM model of computation.*

4 Conclusion and Open Problems

We introduced novel linear and near-linear time algorithms for approximating the minimum dominating set and minimum independent dominating set in a unit disk graph, proving an upper bound of 44/9 to the approximation factor of our algorithms. Nevertheless, the best lower bound we are aware of is 4.8, which is attained by the unit disk graph in Figure 2. Closing this gap would likely require the development of new tools to prove that certain graphs are not unit disk graphs. Computer generated proofs may be useful towards this goal.

References

1. Christiano, P., Kelner, J.A., Madry, A., Spielman, D.A., Teng, S.-H.: Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In: Proc. 43rd Annual ACM Symp. on Theory of Computing (STOC), pp. 273–282 (2011)
2. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discrete Mathematics 86(1-3), 165–177 (1990)
3. De, M., Das, G., Nandy, S.: Approximation algorithms for the discrete piercing set problem for unit disk. In: Proc. 23rd Canadian Conference on Computational Geometry (CCCG), pp. 375–380 (2011)
4. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer (2010)
5. Erlebach, T., Mihalák, M.: A $(4 + \epsilon)$-approximation for the minimum-weight dominating set problem in unit disk graphs. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 135–146. Springer, Heidelberg (2010)
6. Fejes Tóth, L.: Lagerungen in der Ebene, auf der Kugel und im Raum. Springer (1953)
7. Fodor, F.: The densest packing of 13 congruent circles in a circle. Contributions to Algebra and Geometry 44(2), 431–440 (2003)
8. Frinch, S.R.: Mathematical Constants. Encyclopedia of Mathematics and its Applications, vol. 94. Cambridge (2003)
9. Gibson, M., Pirwani, I.A.: Algorithms for dominating set in disk graphs: Breaking the log$n$ barrier. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 243–254. Springer, Heidelberg (2010)
10. Hunt III, H.B., Marathe, M.V., Radhakrishnan, V., Ravi, S., Rosenkrantz, D.J., Stearns, R.E.: NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. Journal of Algorithms 26, 238–274 (1998)
11. Hurink, J.L., Nieberg, T.: Approximating minimum independent dominating sets in wireless networks. Information Processing Letters 109(2), 155–160 (2008)
12. Marathe, M.V., Breu, H., Hunt III, H.B., Ravi, S.S., Rosenkrantz, D.J.: Simple heuristics for unit disk graphs. Networks 25(2), 59–68 (1995)
13. McDiarmid, C., Muller, T.: Integer realizations of disk and segment graphs. preprint, arXiv:1111.2931 (2011)
14. Nieberg, T.: Independent and Dominating Sets in Wireless Communication Graphs. PhD thesis, University of Twente (2006)
15. Nieberg, T., Hurink, J., Kern, W.: Approximation schemes for wireless networks. ACM Transactions on Algorithms 4(4), 49:1–49:17 (2008)
16. Pál, J.: Ein minimumprobleme für ovale. Math. Annalen 83, 311–319 (1921)
17. Spinrad, J.: Efficient Graph Representations. Fields Inst. monographs. AMS (2003)
18. Vinkemeier, D.E.D., Hougardy, S.: A linear-time approximation algorithm for weighted matchings in graphs. ACM Transactions on Algorithms 1, 107–122 (2005)
19. Zou, F., Wang, Y., Xu, X.-H., Li, X., Du, H., Wan, P., Wu, W.: New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs. Theoretical Computer Science 412(3), 198–208 (2011)

On Minimum-and Maximum-Weight Minimum Spanning Trees with Neighborhoods

Reza Dorrigiv[1], Robert Fraser[2], Meng He[1], Shahin Kamali[2], Akitoshi Kawamura[3], Alejandro López-Ortiz[2], and Diego Seco[4]

[1] Dalhousie University, Halifax, Canada
[2] University of Waterloo, Waterloo, Canada
[3] University of Tokyo, Tokyo, Japan
[4] University of A Coruña, A Coruña, Spain

Abstract. We study optimization problems for the Euclidean minimum spanning tree (MST) on imprecise data. To model imprecision, we accept a set of disjoint disks in the plane as input. From each member of the set, one point must be selected, and the MST is computed over the set of selected points. We consider both minimizing and maximizing the weight of the MST over the input. The minimum weight version of the problem is known as the minimum spanning tree with neighborhoods (MSTN) problem, and the maximum weight version (MAX-MSTN) has not been studied previously to our knowledge. We provide deterministic and parameterized approximation algorithms for the MAX-MSTN problem, and a parameterized algorithm for the MSTN problem. Additionally, we present hardness of approximation proofs for both settings.

1 Introduction

We consider geometric problems dealing with imprecise data. In this setting, each point of the input is provided as a *region of uncertainty*, i.e., a geometric object such as a line, disk, set of points, etc., and the exact position of the point may be anywhere in the object. Each object is understood to represent the set of possible positions for the corresponding point. In our work, we study the Euclidean minimum spanning tree (MST) problem. Given a tree T, we define its weight $w(T)$ to be the sum of the weights of the edges in T. For a set of fixed points P in Euclidean space, the weight of an edge is the distance between the endpoints, and we write $mst(P)$ for the weight of the MST on P. Thus, $mst(P) = \min w(T)$, where the minimum is taken over all spanning trees T on P.

Given a set of disjoint disks as input, we wish to determine the minimum and maximum weight MSTs possible when a point is fixed in each disk. The minimum weight MST version of the problem has been studied previously, and is known as the minimum spanning tree with neighborhoods problem (MSTN). This paper introduces the maximum weight MST version of the problem, which we call the MAX-MSTN problem. Assume we are given a set $D = \{D_1, \ldots, D_n\}$ of disjoint disks in the plane, i.e., $D_i \cap D_j = \emptyset$ if $i \neq j$. The MSTN problem on D asks for the selection of a point $p_i \in D_i$ for each $D_i \in D$ such that the

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 93–106, 2013.

weight of the MST of the selected points is minimized. Similarly, MAX-MSTN asks for a selection of p_i such that the weight of the MST of the selected points is maximized.

1.1 Related Work

The first known MST algorithm was published over 80 years ago [11], and a number of successful variants have followed (see [8] for the history of the problem). A review of models of uncertainty and data imprecision for computational geometry problems is provided in [10]. Here, we discuss a few results that are directly related to the MST problem and our model of imprecision.

The MSTN problem on unit disks has been shown to admit a PTAS [13]. A hardness proof for a generalization of MSTN where the neighborhoods are either disks or rectangles appeared in [13], but the proof was faulty. One of the authors later conjectured that a reduction from planar 3-SAT might be used to show the hardness of the MSTN problem [12, p.106]. In Section 3.2, we prove this conjecture.

Löffler and van Kreveld [10] demonstrated that it is algebraically difficult to compute the MST when the regions of uncertainty are continuous regions of the plane, even for very simple inputs such as disks or squares, as the solution may involve the roots of high degree polynomials. It is of independent interest to see if the problem is combinatorially difficult. In the same paper, authors proved that the MSTN problem is (combinatorially) NP-hard if the regions of uncertainty are not pairwise disjoint, through a reduction from the minimum Steiner tree problem. In this paper we prove the hardness of the special case in which the regions are pairwise disjoint.

Erlebach et al. [6] used a model of uncertainty where information regarding the weight of an edge between a pair of points or the position of a point may be obtained by pinging the edge or vertex, and they sought to minimize the number of pings required while obtaining the optimal solution. The distinction is that in their work, they were interested in reducing the amount of communication that is required to locate points within a region of uncertainty, while in our model, the objective is to optimize the MST given regions of uncertainty.

Researchers have considered other related problems that deal with imprecise data. The travelling salesman with neighborhoods (TSPN) problem has been studied extensively. The problem was introduced by Arkin and Hassin [1], in a paper that has been applied, improved, built-upon or otherwise referenced over 150 times. There exists a PTAS for TSPN when the neighborhoods are disjoint unit disks [5]. The most general version of the problem, where regions may overlap and may have varying sizes, is known to be APX-hard [2]. The problem of maximizing the smallest pairwise distance in a set of n points with neighborhoods has also been studied and proved to be NP-hard [7].

1.2 Our Results

We present a variety of results related to the MSTN and MAX-MSTN problems. For both problems we assume the regions of uncertainty (disks) are disjoint.

- MAX-MSTN: deterministic 1/2-approximation;
- MAX-MSTN: parameterized $1 - \frac{2}{k+4}$-approximation (where k represents the separability of the instance, which is to be defined later);
- MAX-MSTN: proof of hardness of approximation;
- MSTN: parameterized $1 + 2/k$-approximation (k is the separability of the instance);
- MSTN: proof of hardness of approximation.

The deterministic approximation algorithm for MAX-MSTN (Section 2.1) is based on choosing the center points of the disks; the interesting aspect in this section lies in the analysis. The parameterized algorithms (Sections 2.2 and 3.1) for both settings were inspired by the observation that the approximation factor improves rapidly as the distance between disks increases. To address this, we introduce a measure of how much separation exists between the disks, which we call *separability*, and we analyze the approximation factor of the MST on disk centers with respect to separability.

For both hardness of approximation results, we establish that there is no FPTAS for the problems unless P=NP. Although the hardness proofs both consist of reductions from planar 3-SAT, the gadgets used are quite distinct and either reduction is interesting even given the existence of the other. In both cases, we construct an instance of our problem from the planar 3-SAT instance, and show that it is possible to compute the weight of the optimal solution to our problem assuming that the 3-SAT instance is satisfiable. If the instance is not satisfiable, we prove that the weight is changed by at least a constant amount (reduced by at least 0.33 units for MAX-MSTN, and increased by at least 0.84 units for MSTN).

2 MAX-MSTN

In this section we study a couple of approximation algorithms for the MAX-MSTN problem, and then we present the proof of hardness of approximation. We begin with a deterministic algorithm below, followed by a parameterized algorithm in Section 2.2.

2.1 Deterministic 1/2-Approximation Algorithm

To approximate the solution to MAX-MSTN, we first consider the algorithm that builds an MST on the centers of the disks. We show this algorithm approximates the optimal solution within a factor of 1/2, i.e., the weight of the MST built on the centers is not smaller than half of that of the optimal tree.

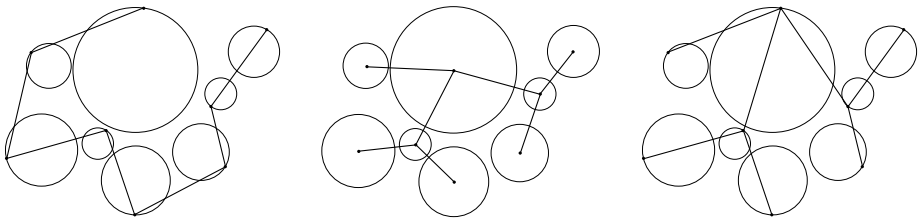


Fig. 1. To compare $w(T_c)$ with $w(T_{\text{opt}})$, we use an intermediate tree T'_c. (a) The optimal result for MAX-MSTN (T_{opt}). (b) The MST T_c on centers. (c) The spanning tree T'_c with the same topology as T_c, using the points of T_{opt}.

Theorem 1. *Consider the* MAX-MSTN *problem for a set D of disjoint disks. Let T_c denote the MST on the centers of the disks, and let T_{opt} be the maximum MST (i.e., the optimal solution to the problem). Then $w(T_c) \geq 1/2 \cdot w(T_{opt})$.*

Proof. Let T'_c be the spanning tree (not necessarily an MST) with the same topology (i.e., combinatorial structure of the tree) as T_c but on the points of T_{opt} (see Figure 1). Since T'_c and T_{opt} span the same set of points, and T_{opt} is an MST, we have $w(T_{\text{opt}}) \leq w(T'_c)$. On the other hand, since T'_c and T_c have the same topology, we have $w(T'_c) \leq 2w(T_c)$; this is because when we move the points from the center to somewhere else in the disks, the weight of each edge increases by at most the sum of the radii of the two involved disks and, since the disks are disjoint, the increase is at most equal to the original weight. To summarize, we have $w(T_{\text{opt}}) \leq w(T'_c)$ and $w(T'_c) \leq 2w(T_c)$, which completes the proof. □

2.2 Parameterized $1 - \frac{2}{k+4}$-Approximation Algorithm

Observe that in order to get the approximation algorithm for MAX-MSTN in Section 2.1, we require disks to be disjoint. Intuitively, if we know that disks are further apart, we can get better approximation ratios. We formalize this intuition by providing a parameterized analysis, i.e., we express the performance of the algorithm in terms of a *separability parameter*[1]. Let $r_{\max}$ be the maximum radius of our disks. We say that a given input for our problem satisfies *k-separability* if the minimum distance between any two disks is at least $k \cdot r_{\max}$. The separability of an input instance I is defined as the maximum k such that I satisfies k-separability. With this definition, we have the following result:

Theorem 2. *For* MAX-MSTN *when the regions of uncertainty are disjoint disks with separability parameter $k > 0$, the algorithm that builds an MST on the centers of the disks achieves a constant approximation ratio of $\frac{k+2}{k+4} = 1 - \frac{2}{k+4}$.*

[1] Separability is similar in spirit to the notion of a well-separated pair; see [3].

Proof. Let T_c be the MST on the centers of the disks. We can extend the analysis in the proof of Theorem 1 to show that the approximation factor is $\frac{k+2}{k+4} = 1 - \frac{2}{k+4}$ for any input that satisfies k-separability. Define T_{opt} and T'_c as before. Consider an arbitrary edge e in T'_c and let D_i and D_j be the two disks connected by e. Let r_i and r_j be the radii of D_i and D_j, respectively, and let d be the distance between D_i and D_j. In T_c the disks D_i and D_j are connected by an edge e' whose weight is $d + r_i + r_j$. The weight of e, on the other hand, can be at most $d + 2r_i + 2r_j$. Therefore, the ratio between the weight of an edge in T_c and its corresponding edge in T'_c is at least

$$\frac{d + r_i + r_j}{d + 2r_i + 2r_j} \geq \frac{kr_{\max} + r_i + r_j}{kr_{\max} + 2r_i + 2r_j} \geq \frac{kr_{\max} + r_{\max} + r_{\max}}{kr_{\max} + 2r_{\max} + 2r_{\max}} = \frac{k+2}{k+4}.$$

Since this holds for any edge of T'_c, we get $w(T_c) \geq \frac{k+2}{k+4} w(T'_c) \geq \frac{k+2}{k+4} w(T_{\text{opt}})$, and we get an approximation factor of $\frac{k+2}{k+4}$. □

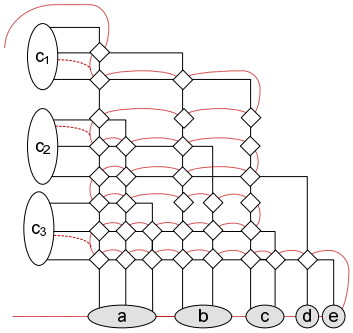
The approximation ratio gets arbitrarily close to 1 as k increases. This confirms our intuition that if the disks are further apart (more separate), we get a better approximation factor.

2.3 Hardness of Approximation

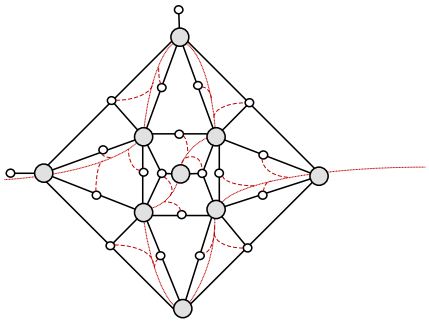
We present a hardness proof for the MAX-MSTN problem by a reduction from the planar 3-SAT problem [9]. Planar 3-SAT is a variant of 3-SAT in which the graph $G = (V, E)$ associated with the formula is planar.

Theorem 3. MAX-MSTN *does not admit an FPTAS unless* P=NP.

We show a reduction from any instance of the planar 3-SAT problem to the MAX-MSTN problem. In planar 3-SAT, we have a planar bipartite graph $G = (V, E)$, where $V = V_v \cup V_c$, so that there is a vertex in V_v for each variable and a vertex in V_c for each clause; there is an edge (v_i, v_j) in E between a variable vertex $v_i \in V_v$ and a clause vertex $v_j \in V_c$ if and only if the clause contains a literal of that variable in the 3-SAT instance. In [9] it was shown that the planar 3-SAT problem is NP-hard via a reduction from the standard 3-SAT problem. Further, it was observed that the resulting instance of planar 3-SAT permits the construction of a path $P = (V_v, E_P)$ using a set of edges E_P such that $E \cap E_P = \emptyset$, where P is connected and passes through all vertices in V_v without crossing any edge in E. We call this path P the *spinal path.* We further observe that additional edges can be added to P to get a *spinal tree* T which also covers clause vertices V_c. In this sense T will be a tree that covers all vertices without crossing an edge of G such that all vertices corresponding to clauses are leaves. These observations are illustrated in Figure 2. To prove the hardness of MAX-MSTN, we make use of the spinal tree. Due to lack of space, the complete reduction is omitted from this presentation (the details may be found in the complementary technical report [4]).



(a) The reduction from 3-SAT to planar 3-SAT.



(b) The planar gadget located in the intersection points. Variable and clause vertices are represented by large and small circles, respectively.

Fig. 2. The reduction from 3-SAT to planar 3-SAT as presented in [9]. The variable and clause vertices of 3-SAT are located respectively in x and y axis, and the edges are drawn as orthogonal paths (a). A planar gadget is placed on each intersection point. Each gadget includes some new variable and clause vertices (b). In [9], it is observed that there is a path (we call it the spinal path) that covers all variable vertices of planar instance without crossing any edge (solid lines). We observe that additional edges can be added to the spinal path to obtain a tree (spinal tree) which spans clause variables as leaves (dashed lines).

3 MSTN

In this section we present a parameterized algorithm for the MSTN problem, followed by the proof of hardness of approximation.

3.1 Parameterized $1 + 2/k$-Approximation Algorithm

Recall that to have *k-separability* means that the minimum distance between any two disks is at least $kr_{\max}$, and the separability of an input instance I is defined as the maximum k such that I satisfies k-separability.

Theorem 4. *For* MSTN *when the regions of uncertainty are disjoint disks with separability parameter $k > 0$, the algorithm that builds an MST on the centers of the disks achieves a constant approximation ratio of $\frac{k+2}{k} = 1 + 2/k$.*

Proof. Assume that we have a set D of n disks that satisfies k-separability. Let T_c be the MST on the centers and T_{opt} be an optimal MST, i.e., an MST that contains one point from each disk and its weight is the minimum possible. Define *Temp* as the spanning tree (not necessarily an MST) with the same topology as

T_{opt} but on the points of T_c, i.e., on the centers. Since T_c is an MST on centers, we have $w(T_c) \leq w(\mathit{Temp})$. Consider an arbitrary edge e in *Temp* and let D_i and D_j be the two disks that are connected by e. Let r_i and r_j be the radii of D_i and D_j, respectively, and let d be the distance between D_i and D_j. In T_{opt} the disks D_i and D_j are connected by an edge e' whose weight is at least d. The weight of e on the other hand is $d + r_i + r_j$. Therefore the ratio between the weight of an edge in T_{opt} and its corresponding edge in *Temp* is at least

$$\frac{d}{d + r_i + r_j} \geq \frac{kr_{\max}}{kr_{\max} + r_i + r_j} \geq \frac{kr_{\max}}{kr_{\max} + r_{\max} + r_{\max}} = \frac{k}{k+2}.$$

Since this holds for any edge of *Temp*, we get $w(T_c) \leq w(\mathit{Temp}) \leq \frac{k+2}{k} w(T_{\text{opt}})$. Therefore we get an approximation factor of $\frac{k+2}{k} = 1 + 2/k$ for the algorithm. □

As with the parameterized algorithm for MAX-MSTN, as the disks become further apart (as k grows), the approximation factor approaches 1.

3.2 Hardness of Approximation

To prove the hardness of the MSTN problem, we present a reduction from the planar 3-SAT problem. Recall that planar 3-SAT is a variant of 3-SAT in which the graph $G = (V, E)$ associated with the formula is planar.

Theorem 5. MSTN *does not admit an FPTAS unless* P=NP.

In the hardness proof of MAX-MSTN, we used a spinal tree in the reduction. In this section, we use the *spinal path* as a path $P = (V_v, E_P)$ with a set of edges E_P such that $E \cap E_P = \emptyset$, where P passes through all variable vertices in G without crossing any edge in E. As mentioned earlier, the restricted version of planar 3-SAT remains NP-hard [9]. To reduce planar 3-SAT to MSTN, we begin by finding a planar embedding of the graph associated with the SAT formula. We force the inclusion of the spinal path as a part of the MST using wires. We define a wire as a set of disks of radius 0 placed in close succession, so that we may interpret a wire as a fixed line in the MSTN solution. We replace each variable vertex of V by a *variable gadget* in our construction. These gadgets are composed of a set of disks and some wires, and are defined in such a way that we may choose the points so that the size of the MST is equal to a certain value, if and only if the SAT formula is satisfiable.

Variable Gadgets. A variable gadget is formed by a *k-flower*, where $k = 4c+6$ and c is the number of clauses in the planar 3-SAT instance that include the variable (each clause requires 4 disks, and each of the edges of the spinal path requires 3 disks). As illustrated in Figure 3, a k-flower is composed of k disks of unit radius, centered on the vertices of a regular k-gon. Also, each disk is tangent to its two neighboring disks, and each pair of consecutive disks D_i, D_{i+1}

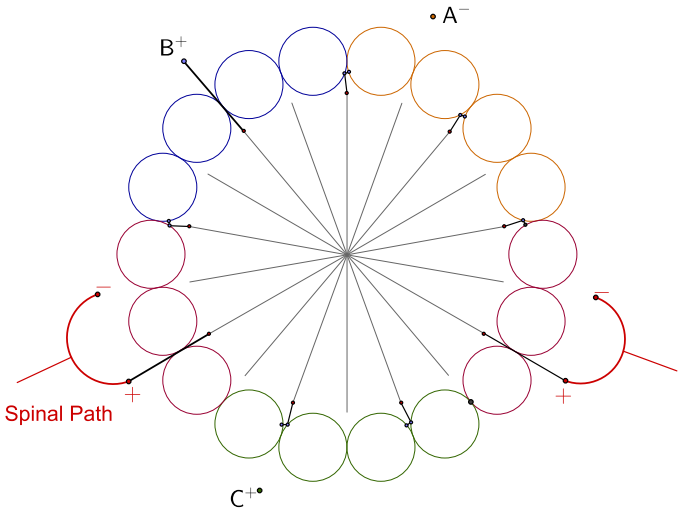


Fig. 3. A variable gadget with eighteen disks (containing an 18-flower and an 18-star) for a variable x. Here B^+ and C^+ are the endpoints of the wires that connect to clauses that include x in the positive form, while A^- represents a clause that includes x in negative form. The picture illustrates the case in which the algorithm takes the positive choice for x, and clause B is satisfied with x. Clause C is satisfied via some other variable, as is clause A, assuming that it is satisfied. Note that every other path on the k-star connects to a pair of disks on the k-flower.

intersects at a single point $q_{i,i+1} = D_i \cap D_{i+1}$, which we call a *tangent point*[2]. Moreover, there is a *k-star* in the middle of the gadget composed of k fixed wires, where the i^{th} wire connects a point unit distance from the tangent point $q_{i,i+1}$ to the center point of the k-star. The spinal path is placed so that it approaches the variable gadget twice, and each of these approaches requires three disks. We split the wires of the spinal path once near the variable gadget as shown in Figure 3, and wires terminate at a distance ≈ 1.755 from the nearest tangent point, for reasons discussed in the Clause Gadgets section.

Lemma 1. *Suppose we are given two unit disks D_1 and D_2 that intersect exclusively at a single point $q = D_1 \cap D_2$, and a line ℓ such that $q \in \ell$ and ℓ is tangent to both D_1 and D_2 (i.e., ℓ is the perpendicular bisector of the center points of D_1*

[2] Using this construction, pairs of disks of the k-flower trivially intersect at a single point, which simplifies our analysis. To achieve strict disjointedness, the disks of the k-flower may be contracted to have radius $1 - \gamma$ so that the tangent point is now distance γ from the nearest point in the adjacent disks. Any path which uses the tangent point in our analysis will have less than 2γ units of additional weight on these shrunken disks, and there are fewer than $n(4m + 6)$ disks, where n and m are the number of variables and clauses respectively. Choosing an appropriate value of γ so that $2\gamma n(4m + 6) \ll 0.845$ achieves the same result as our simplified analysis.

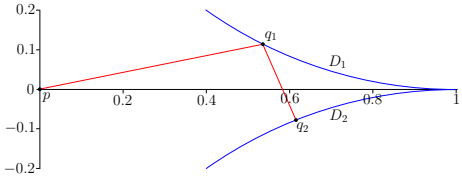


Fig. 4. The shortest possible path is shown from a point at the origin to some point in each of two unit disks; one of the disks is centered at (1,1), the other is at (1,-1).

and D_2). Now, given a point $p \in \ell$ where p is unit distance from q, the shortest path consisting of points $p, q_1 \in D_1$, and $q_2 \in D_2$ has weight $d \approx 0.755$.

Proof. If $q_1 = q_2 = q$, then the path has unit length, so a path of length d is shorter. A path with edges $e_1 = (q_1, p)$ and $e_2 = (p, q_2)$ has length at least 0.828, since the nearest point on D_1 or D_2 to p is $\sqrt{2} - 1 > 0.414$ units distant.

Therefore, we may assume without loss of generality that the path consists of the edges $e_1 = (p, q_1)$ and $e_2 = (q_1, q_2)$ and the path has length $d = w(e_1)+w(e_2)$, where $w(e)$ is the length of the edge e. Therefore, we must choose q_1 and q_2 so that d is minimized. Note that candidate positions for each of q_1 and q_2 may be restricted to the boundaries of their respective disks.

For the purposes of simplifying the proof, assume that p is at the origin of the Cartesian plane, and D_1 and D_2 are centered at $(1, 1)$ and $(1, -1)$, respectively. Then a point q_1 on the boundary of D_1 may be expressed as $(sin(\alpha)+1, cos(\alpha)+1)$, for some $\alpha \in [0 \ldots 2\pi]$, and analogously $q_2 = (sin(\beta)+1, cos(\beta)-1)$, for some $\beta \in [0 \ldots 2\pi]$. Therefore, we simply have to find the minimum of the function

$$f(\alpha, \beta) = \sqrt{(\sin\alpha + 1)^2 + (\cos\alpha + 1)^2} + \sqrt{(\sin\beta - \sin\alpha)^2 + (\cos\beta - \cos\alpha - 2)^2},$$

over the variables $\alpha \in [0 \ldots 2\pi], \beta \in [0 \ldots 2\pi]$. Using Maple, we see that this minimum has value $d \approx 0.755$, at $\alpha \approx 3.62, \beta \approx 5.89$. The optimal path in this setting is shown in Figure 4. Since this path is shorter than all other possible path configurations, we conclude that this is the shortest possible path including p and points $q_1 \in D_1$ and $q_2 \in D_2$. □

For the remainder of the discussion, we refer to the weight of this shortest path as the constant d. Before going to the details of the reduction, we consider optimal MSTN solutions when the problem instance is a variable gadget, as described above (without the wires approaching from clauses). We claim that such an instance has two possible MSTN solutions, and in each of these solutions consecutive pairs of disks are connected to a single wire of the k-star with a path of length d described in Lemma 1. We associate these two possible MSTN solutions with the two assignments for the variable. To prove the claim, we show that in an optimal MSTN solution for a k-star, there is no path containing points from more than two disks.

Lemma 2. *In an optimal MSTN solution for a k-star, a path containing a single wire of the k-star includes at most two disks from the k-flower, when $k \geq 8$.*

Proof. Recall that by Lemma 1, connecting a pair of disks to a k-wire may be done with weight d, while a wire may be connected to a single disk with weight $\sqrt{2}-1$. Therefore, three consecutive disks in a k-flower may be connected to two wires of the k-star using edges with weight $d+\sqrt{2}-1 \approx 1.169$, while four such disks may be connected with weight $2d \approx 1.51$.

Now consider three consecutive disks that we wish to connect to a single wire of the k-star. Given that $k \geq 8$, the minimum distance between the two non-adjacent disks is $d_{\min} \geq 2\sqrt{2+\sqrt{2}}-2 \approx 1.696$. Therefore, a path simply connecting three disks (and yet still disjoint from the k-star) has greater weight than even the path joining four disks using two wires of the k-star, and thus an optimal path containing one wire of a k-star in the MST contains points from at most two disks of the k-flower. □

Corollary 1. *In the optimal MSTN solution for a k-flower (when k is even), each consecutive pair of disks is connected to a single wire of the k-star via a path of length d.*

This follows immediately from Lemmas 1 and 2. Hence, there are two possible solutions for MSTN on a k-flower where k is an even number (this is the case in our construction). We use this fact to assign a truth value for the variable gadget: one configuration is arbitrarily considered to be `true`, the other `false`. In Figure 3, we show an example where the `true` configuration is used, and every other wire of the k-star has an edge to some point in the k-flower. The `false` configuration would contain edges between the complementary set of wires of the k-star and the disks of the k-flower.

Clause Gadgets. The clause gadgets are composed of three wires that meet at a single point. Each wire of the clause gadget is placed so that it terminates at a distance $1+d$ from a tangent point, where the terminal point is collinear with a line of the k-star on the relevant variable gadget. As a result, a line segment of length $2+d$ units can connect the clause gadget to the k-star of a variable gadget, while also intersecting the shared point between two disks of the k-flower. If the truth value of the k-flower gadget matches that of the clause, this means that connecting the clause to the flower requires two units of extra weight, since otherwise the two disks are connected to the k-star with d weight, as outlined in Lemma 1. Therefore, given a clause gadget where at least one literal matches the truth value of the corresponding variable gadget, the clause gadget is connected to the MST with two units of additional weight.

The spinal path wires terminate in positions exactly analogous to those of the clause gadgets so that the analysis is the same. This raises the possibility that the wires of a clause gadget may be connected to two variable gadgets, leaving a gap in the spinal path, but note that such a configuration does not affect the weight of the optimal tree. The spinal path is necessary however, since some variables may not be used by any clauses in an optimal solution.

Lemma 3. *Joining a clause wire to a k-flower that has a truth value differing from that of the clause requires at least ≈ 0.845 units of additional edge weight relative to a configuration with matching truth values.*

Proof. In an optimal MSTN solution on a construction corresponding to a satisfiable 3SAT instance, a pair of disks and a clause wire may be joined to the k-star with weight $2 + d$ units, and an additional adjacent pair of disks may be joined to the k-star with a path of weight d. Therefore, the total weight of the edges incident upon points in four such disks is $2 + 2d$.

Now consider a configuration where the truth value of the literal for each variable in a clause does not match the truth value of the corresponding variable gadgets. Connecting one of the clause gadget wires to the k-star requires an additional weight of $2 + d$, as discussed previously, which intersects points from two disks; call them D_i and D_{i+1}. The neighboring two disks in the k-flower, D_{i-1} and D_{i+2}, are not attached to the k-star by paths like those found in Lemma 1. Rather, each of these adjacent paths may be shortened to $\sqrt{2}$ to cover the two singleton disks. Note that there may be a non-empty sequence of pairs of disks connected as in Lemma 1 before the singleton is reached, creating a section of the flower with an inverted truth value for the variable[3]. Therefore, the net extra weight of such a transition is $2 + d + 2\sqrt{2} - (2 + 2d) = 2\sqrt{2} - d \approx 2.0735$.

A configuration that may require less additional weight is to connect the clause wire to the k-star using a path with points in disks D_{i-1} and D_i (it is a slightly modified configuration from that of Lemma 1). As k increases, the weight of such a path decreases. To minimize the length of the path, suppose that the centers of D_{i-1}, D_i and D_{i+1} are collinear (which occurs when $k = \infty$). Therefore, we can place the center of D_{i-1} at $(1, 1)$, the end of the k-star wire between D_{i-1} and D_i at $(0, 0)$, and the end of the clause wire at $(2 + d, -2)$ (Figure 5). The weight of the path from the k-star to D_{i-1} to the clause wire may be expressed by the function

$$f(\theta) = \sqrt{(1 + \sin\theta)^2 + (1 + \cos\theta)^2} + \sqrt{(1 + d - \sin\theta)^2 + (-3 - \cos\theta)^2},$$

which has a minimum length slightly greater than 3.60 units at $\theta \approx 3.49$ radians. Since this path intersects D_i, it is also the shortest path that includes a point $p_i \in D_i$. Therefore, w.l.o.g. a path connecting a clause wire to a wire in a variable gadget with a mismatched truth value has weight greater than 3.6. Note that such a path does not affect the truth value of the variable gadget, and so D_{i+1} and D_{i+2} may be joined to the k-star with a path of weight d. Therefore, the extra weight incurred for such a configuration is $> 3.6 + d - (2d + 2) \approx 0.845$. □

As described earlier, the terminal points of the clause wires (and the spinal path) are collinear with wires of the k-star. Since we never place these terminal points

[3] D_{i+2} may be more generally indexed as D_{i+2+4c}, where there is a block of $4c$ disks in the k-flower joined to the k-star in a truth configuration opposite of that of the neighboring disks in the k-flower. This does not affect the analysis, it simply relocates the singleton disk. Recall that by Lemma 2, such singletons would exist rather than having three disks connected by a path to a single edge of the k-star.

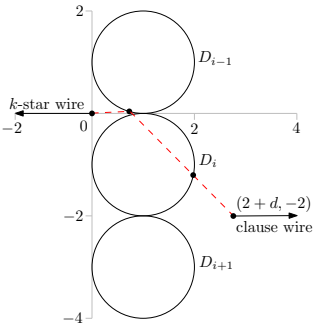


Fig. 5. The shortest possible path is shown (the dashed line) from the end of the clause wire to points in D_i and D_{i-1}, and finally connecting to the k-star wire for D_i and D_{i-1}.

on adjacent wires of the k-star, the wires need not lie within 4 units of one another, and so there will not be edges directly between different clause wires or between a clause wire and the spinal path.

Reduction. We would like to reduce a given instance of planar 3-SAT to the MSTN problem. Note that the given 3-SAT instance is assumed to be embedded on the plane, and there exists a spinal path $P = (V_v, E_P)$ that passes all variable vertices without crossing any edge of G, such that all variable vertices but 2 have degree 2 in P (as mentioned at the beginning of Section 3.2, this restricted version is also NP-hard).

To create the instance of the MSTN problem, we fix the spinal path as a part of the MST, using wires consisting of disks of radius 0. We replace each variable node with a variable gadget as explained. Each clause gadget includes three wires, which we place so that they approach the associated variable gadgets as described.

The wires forming the spinal path, the m clause gadgets, and each of the n k-stars have a fixed weight, call the total weight of all these wires w_{wires}. The remaining weight of the MST is that of connecting to a point from each disk in the k-flowers, and that of connecting each clause gadget. Suppose there exists a satisfying assignment for the 3-SAT instance. Each pair of disks in the k-flowers can be connected with weight d; this will be the case for all but m pairs. The remaining m pairs will be connected with edges that also join to the clause gadgets in the manner described in Section 3.2 with weight $2 + d$. Therefore, assuming that there is a total of i pairs of disks in the k-flowers of the construction, the remaining weight of the MST is $w_{\text{disks}} = id + 2m$. Thus, if there exists a satisfying assignment to the 3-SAT instance, the total optimal weight of the MST is $w_{\text{tot}} = w_{\text{wires}} + w_{\text{disks}}$.

If there is no satisfying assignment, at least one of the clause gadgets must be connected to the MST in the manner described in Lemma 3, which requires an additional weight of at least 0.845. Now suppose there exists an FPTAS for MSTN. Given an instance of planar 3-SAT, we build the MSTN construction and determine w_{tot}. We choose a value of ε so that $\varepsilon < 0.845/w_{\text{tot}}$, and so a $(1 + \varepsilon)$-approximate solution to the MSTN problem may be used to determine

whether there is a satisfying assignment for the planar 3-SAT instance. Since the latter problem is NP-hard, we conclude that MSTN does not admit an FPTAS unless P=NP.

4 Conclusions

We considered geometric MST with neighborhoods problems, and established that computing the MST of minimum or maximum weight is hard to approximate in this setting by proving that there is no FPTAS for either problem, assuming $\mathsf{P} \neq \mathsf{NP}$. We provided a parameterized algorithm for the MSTN problem based upon how well separated the disks are from one another. For MAX-MSTN, we showed that a deterministic algorithm that selects disk centers gives an approximation ratio of $1/2$. Furthermore, we showed that when the instance of the problem satisfies k-separability, the same approach achieves a constant approximation ratio of $1 - \frac{2}{k+4}$.

For further research, it will be interesting to study this problem under different models of imprecision. Depending on the application, the regions of uncertainty may consist of other shapes, e.g., line segments, rectangles, etc., or they may be composed of discrete sets of points.

References

1. Arkin, E., Hassin, R.: Approximation algorithms for the geometric covering salesman problem. Discrete Applied Mathematics 55(3), 197–218 (1994)
2. de Berg, M., Gudmundsson, J., Katz, M., Levcopoulos, C., Overmars, M., van der Stappen, A.: TSP with neighborhoods of varying size. Journal of Algorithms 57(1), 22–36 (2005)
3. Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. J. ACM 42(1), 67–90 (1995)
4. Dorrigiv, R., Fraser, R., He, M., Kamali, S., Kawamura, A., López-Ortiz, A., Seco, D.: On minimum- and maximum-weight minimum spanning trees with neighborhoods. Tech. Rep. CS-2012-14, University of Waterloo (2012)
5. Dumitrescu, A., Mitchell, J.S.: Approximation algorithms for TSP with neighborhoods in the plane. Journal of Algorithms 48(1), 135–159 (2003)
6. Erlebach, T., Hoffmann, M., Krizanc, D., Mihalák, M., Raman, R.: Computing minimum spanning trees with uncertainty. In: Symposium on Theoretical Aspects of Computer Science, pp. 277–288 (2008)
7. Fiala, J., Kratochvíl, J., Proskurowski, A.: Systems of distant representatives. Discrete Applied Mathematics 145(2), 306–316 (2005)
8. Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. IEEE Annals of the History of Computing 7(1), 43–57 (1985)
9. Lichtenstein, D.: Planar formulae and their uses. SIAM J. on Computing 11(2), 329–344 (1982)
10. Löffler, M., van Kreveld, M.: Largest and smallest convex hulls for imprecise points. Algorithmica 56, 235–269 (2010)

11. Nešetřil, J., Milková, E., Nešetřilová, H.: Otakar Borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. Discrete Mathematics 233(13), 3–36 (2001)
12. Yang, Y.: On several geometric network design problems. Ph.D. thesis, State University of New York at Buffalo (2008)
13. Yang, Y., Lin, M., Xu, J., Xie, Y.: Minimum spanning tree with neighborhoods. In: Kao, M.-Y., Li, X.-Y. (eds.) AAIM 2007. LNCS, vol. 4508, pp. 306–316. Springer, Heidelberg (2007)

Asymptotically Optimal Online Page Migration on Three Points

Akira Matsubayashi

Division of Electrical Engineering and Computer Science,
Kanazawa University,
Kanazawa 920-1192, Japan
mbayashi@t.kanazawa-u.ac.jp

Abstract. This paper addresses the page migration problem: given online requests from nodes on a network for accessing a page stored in a node, to output online migrations of the page. Serving a request costs the distance between the request and the page, and migrating the page costs the migration distance multiplied by the page size $D \geq 1$. The objective is to minimize the total sum of service costs and migration costs. Black and Sleator conjectured that there exists a 3-competitive deterministic algorithm for every graph. Although the conjecture was disproved for the case $D = 1$, whether or not an asymptotically (with respect to D) 3-competitive deterministic algorithm exists for every graph is still open. In fact, we did not know if there exists a 3-competitive deterministic algorithm for an extreme case of three nodes with $D \geq 2$. As the first step toward an asymptotic version of the Black and Sleator conjecture, we present 3- and $(3+1/D)$-competitive algorithms on three nodes with $D = 2$ and $D \geq 3$, respectively, and a lower bound of $3 + \Omega(1/D)$ that is greater than 3 for every $D \geq 3$. In addition to the results on three nodes, we also derive ρ-competitiveness on complete graphs with edge-weights between 1 and $2 - 2/\rho$ for any $\rho \geq 3$, improving the previous 3-competitive algorithm on uniform networks.

1 Introduction

The problem of computing an efficient dynamic allocation of data objects stored in nodes of a network commonly arises in network applications such as memory management in a shared memory multiprocessor system and Peer-to-Peer applications on the Internet. In this paper, we study one of the classical varieties of the problem, the *page migration problem*, in which a request issued on a node for accessing a single data object (called a *page* in this problem) must be served using unicast communication. After serving each request, we are allowed to migrate the page. Serving a request costs the distance of the communication, and migrating the page costs the migration distance multiplied by the page size $D \geq 1$. The objective is to minimize the total sum of the service and migration costs. The page migration problem has been extensively studied (e.g., [2,3,8,10,4,13,15]) and generalized to several settings such as k-page migration [3], file allocation

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 107–119, 2013.

problem, e.g., [2,4,13], and data management on dynamic networks, e.g, [1,7]. See [6] for a recent survey.

1.1 Related Results

We focus on deterministic online page migration algorithms. Black and Sleator [8] first studied competitive analysis of the page migration problem and presented 3-competitive deterministic algorithms on trees, uniform networks, and Cartesian products of these networks, including grids and hypercubes. These algorithms are optimal because the deterministic lower bound is 3 for every network with at least two nodes [8,11]. Black and Sleator conjectured that there exists a 3-competitive deterministic algorithm for every network. The first upper bound of 7 for general networks was given by Awerbuch, Bartal, and Fiat [2] and improved to 4.086 by Bartal, Charikar, and Indyk [3]. For a special case of $D = 1$, a better bound of $2 + \sqrt{2}$ is achievable [14]. For a yet restricted case, a 3-competitive deterministic algorithm on three nodes with $D = 1$ was presented in [10]. Whether or not a 3-competitive deterministic algorithm exists on three nodes for $D \geq 2$ was left open. Concerning the lower bound, Black and Sleator's conjecture was disproved by Chrobak, Larmore, Reingold, and Westbrook [10], who proved that no deterministic algorithm has the competitive ratio less than $85/27 \approx 3.148$ on special networks with $D = 1$. This bound was refined to 3.164 [14]. It is mentioned in [10] that the lower bound is larger than 3 even on four nodes. An explicit lower bound of 3.121 on five nodes was proved in [14].

1.2 Contributions of This Paper

All the previous lower bounds larger than 3 were proved only for the case $D = 1$. Therefore, an asymptotic version of the Black and Sleator conjecture with respect to D, i.e., whether or not an asymptotically 3-competitive deterministic algorithm on every network exists is still open. As the first step toward an answer for this conjecture, we present

- a $(3 + 1/D)$-competitive algorithm on three nodes with $D \geq 3$,
- a 3-competitive algorithm on three nodes with $D = 2$, and
- a lower bound of $3 + \Omega(1/D)$ that is greater than 3 for every $D \geq 3$.

These results thoroughly answer the open question of the existence of a 3-competitive algorithm on three nodes. In addition to the results on three nodes, we also derive

- ρ-competitiveness on complete graphs (of arbitrary size) with edge-weights between 1 and $2 - 2/\rho$ for any $\rho \geq 3$,

improving the previous 3-competitive algorithm on uniform networks [8].

1.3 Overview of Technical Ideas

Our $(3+1/D)$-competitive algorithm is a typical work function algorithm similar to algorithms for metrical task systems, e.g., [9], and k-server problems [5,12]. In general, an algorithm that makes online decisions using information on the optimal offline cost for processing requests that have been issued so far and ending at each configuration (page node in the page migration problem). Such a cost function with respect to configurations is called a work function. To prove a work function (i.e., optimal cost) increases enough, we introduce a probably new technique of analytically dealing with the work function extended on a continuous loop. This idea reveals a relation of increased amounts in the extended work function and its derivative. The author believes that such analysis is the technical contribution of this paper.

Since the competitive ratio on three nodes is not monotonic with respect to D, it appears to be reasonable that we need different approaches between $D = 2$ and $D \geq 3$. Our 3-competitive algorithm for $D = 2$ is based on the counter-based algorithm for uniform networks [8], which maintains a counter on each node. The counters are updated every time a request arrives so that they represent a tendency of migration. If a counter reaches a certain value, then the algorithm moves the page to the node with this counter. One can observe that the original algorithm is 3-competitive even on a complete graph with roughly the same edge-weights, and that this can be generalized to any $\rho \geq 3$. More specifically, there is a "triangle" condition on edge-weights around the page such that the original potential function used in [8] can amortize the service costs and the next migration cost. If there are three nodes, then at least one "good" node satisfies the condition. We design our algorithm by modifying the original algorithm for the page at a "bad" node. Although the modification wastes the "deposit" even worse when leaving the bad node, we can prove through careful observations that much more deposit can be saved after the possible migration to a good node or from services before the migration.

Our lower bound is based on a simple observation: If there are only two nodes, then any 3-competitive algorithm must move after exactly $2D$ requests issued by a cruel adversary, which always issues a request from the other node than the online page. If the adversary carefully adds a new node close to the existent request node and divides the $2D$ requests among these nodes, then no matter when or where the algorithm moves, it is too "impatient" or "tardy" to achieve the competitive ratio of 3.

Due to space constraints, proofs of lemmas are omitted and will be given in a journal version.

2 Preliminaries

The page migration problem can be formulated as follows: given an undirected graph $G = (V, E)$ with edge weights, $s_0, r_1, \ldots, r_k \in V$, and a positive integer D, to compute $s_1, \ldots, s_k \in V$ so that the cost function $\sum_{i=1}^{k}(d_{s_{i-1}r_i} + Dd_{s_{i-1}s_i})$ is

minimized, where d_{uv} is the distance between nodes u and v on G. The terms $d_{s_{i-1}r_i}$ and $Dd_{s_{i-1}s_i}$ represent the cost to serve the request from r_i by the node s_{i-1} holding the page and the cost to migrate the page from s_{i-1} to s_i, respectively. We call s_i and r_i a *server* and a *client*, respectively. An *online* page migration algorithm determines s_i without information of $r_{i+1}, \ldots, r_k$. We denote by $A(\sigma)$ the cost of a page migration algorithm A for a sequence $\sigma := r_1 \cdots r_k$. A deterministic online page migration algorithm ALG is *ρ-competitive* if there exists a constant value α such that $\mathrm{ALG}(\sigma) \leq \rho \cdot \mathrm{OPT}(\sigma) + \alpha$ for any σ, where OPT is an optimal offline algorithm. We denote by $\mathrm{OPT}_u(\sigma)$, called a *work function*, the minimum (offline) cost to process σ so that $s_k = u$. Obviously, $\mathrm{OPT}(\sigma) = \min_{u \in V}\{\mathrm{OPT}_u(\sigma)\}$. An online algorithm that determines the server after processing σ using the information of $\mathrm{OPT}_u(\sigma)$ for all possible nodes u is called a *work function algorithm*. A work function algorithm is well-defined because $\mathrm{OPT}_u(\sigma)$ can be computed using dynamic programming, i.e., for a request issued from r after σ, $\mathrm{OPT}_u(\sigma r) = \min_{v \in V}\{\mathrm{OPT}_v(\sigma) + d_{rv} + Dd_{uv}\}$ and $\mathrm{OPT}_u(\emptyset) = Dd_{s_0 u}$ [10], where $\emptyset$ denotes an empty sequence.

For a node u and $k \geq 1$, we write a sequence consisting of k repetitions of u as u^k. Unless otherwise stated, we suppose that graphs considered here have a node set $V := \{a, b, c\}$ and edge weights $x = d_{ab}$, $y = d_{ac}$, and $z = d_{bc}$ for edges (a, b), (a, c), and (b, c), respectively. We denote $L := x + y + z$ and assume that $\max\{x, y, z\} < L/2$.

3 $(3 + 1/D)$-Competitive Algorithm

We consider a typical work function algorithm denoted by WFA, which moves the server s, located after processing a sequence σ of clients, to a nearest node among nodes v minimizing $\mathrm{OPT}_v(\sigma) + d_{rv} + Dd_{sv}$ after servicing a new request from r. By this definition, the destination $\hat{s}$ of the migration satisfies $\mathrm{OPT}_s(\sigma r) = \mathrm{OPT}_{\hat{s}}(\sigma) + d_{r\hat{s}} + Dd_{s\hat{s}}$. Another way of understanding the algorithm is that WFA moves the server s to $\hat{s}$ when a decline of slope D from s to $\hat{s}$ appears on the work function, i.e., $\mathrm{OPT}_s(\sigma r) - \mathrm{OPT}_{\hat{s}}(\sigma r) = Dd_{s\hat{s}}$, except when s is one of the nodes v minimizing $\mathrm{OPT}_v(\sigma) + d_{rv} + Dd_{sv}$. The purpose of considering such a decline on the work function as a trigger of migration is to avoid requests from $\hat{s}$ that would increase online service cost at the server s but change neither OPT_s nor $\mathrm{OPT}_{\hat{s}}$. A similar idea is used for other work function algorithms ([9,5,12]). We prove the following theorem:

Theorem 1. WFA *is* $(3 + 1/D)$*-competitive on three nodes.*

We begin our proof by deriving a sufficient condition for Theorem 1. We suppose that WFA locates the server on s after processing σ, and that a request is issued from $r \in V$ after σ. For a function f of σ, we use the notations $f = f(\sigma)$ and $f' = f(\sigma r)$ for simplicity. For $u \in V$, let $\hat{u}$ be a nearest node to u among nodes v minimizing $\mathrm{OPT}_v + d_{rv} + Dd_{uv}$. Then,

$$\mathrm{OPT}'_s = \mathrm{OPT}_{\hat{s}} + d_{r\hat{s}} + Dd_{s\hat{s}} \geq \mathrm{OPT}_s + d_{r\hat{s}}, \text{ and} \tag{1}$$

$$\mathrm{OPT}'_s \leq \mathrm{OPT}'_{\hat{s}} + Dd_{s\hat{s}}. \tag{2}$$

These follow from $|\text{OPT}_u - \text{OPT}_v| \le Dd_{uv}$ for any $u, v \in V$ [10]. It follows from (1) and (2) that $d_{r\hat{s}} \le \text{OPT}'_{\hat{s}} - \text{OPT}_s + Dd_{s\hat{s}}$. Therefore, we have

$$\text{WFA}' - \text{WFA} = d_{rs} + Dd_{s\hat{s}} \le d_{r\hat{s}} + (D+1)d_{s\hat{s}} \le \text{OPT}'_{\hat{s}} - \text{OPT}_s + (2D+1)d_{s\hat{s}}. \quad (3)$$

By summing (3) overall requests in σr, we obtain $\text{WFA}' \le \text{OPT}'_{\hat{s}} + (2 + 1/D)M'$, where $M' = M(\sigma r)$ is D times the total sum of migration distances of WFA in processing σr. Hence, if

$$Dd_{\hat{s}u} + M' \le \text{OPT}'_u \text{ for any } u \in V, \quad (4)$$

then by choosing u minimizing OPT'_u, we have $\text{WFA}' \le \text{OPT}'_{\hat{s}} + (2 + 1/D)\text{OPT}' - (2D+1)d_{\hat{s}u} \le (3 + 1/D)\text{OPT}' - (D+1)d_{\hat{s}u}$, which completes the proof of Theorem 1.

The crux of the proof is to show (4). To prove (4), we generalize the network to a continuous ring[1] R of length L containing a, b, and c with the preserved distances. Specifically, we define R as an interval $\{p \mid 0 \le p < L\}$ modulo L, i.e., any real number p is equivalent to $p - \lfloor p/L \rfloor \cdot L$. We define an extended work function at a point $p \in R$ as

$$w'_p = \min_{q \in R}\{w_q + d_{rq} + Dd_{pq}\} \text{ and } w_p(\emptyset) = Dd_{s_0 p}.$$

For a point $p \in R$, we define $\hat{p}$ as a nearest point to p among points $q \in R$ minimizing $w_q + d_{rq} + Dd_{pq}$. Actually, $\hat{p} \in V$ for any $p \in R$ with $p \ne \hat{p}$, which will be proved later in Lemma 5. This means that $w'_p = \min_{q \in V \cup \{p\}}\{w_q + d_{rq} + Dd_{pq}\}$, and hence, $w_u = \text{OPT}_u$ for any $u \in V$. We denote the farthest point of p on R by $\bar{p}$. For $p, q \in R$, we define $[p, q]$ as the closed interval of length d_{pq} between p and q on R if $d_{pq} < L/2$, R otherwise. Notations $(p, q]$, $[p, q)$, and (p, q) are used to denote the intervals obtained from $[p, q]$ by excluding p, q, and both p and q, respectively. Lemmas 1–4 below state basic properties of w_p that will be used in the subsequent lemmas.

Lemma 1. *For any $p, q \in R$, it follows that $w_p - w_q \le Dd_{pq}$.*

Lemma 2. *For any $p \in R$ and $q \in (p, \hat{p}]$, it follows that $\hat{q} = \hat{p}$.*

Lemma 3. *For any $p \in R$ and $q \in [p, \hat{p})$, it follows that $w_q - w_{\hat{p}} > (D-1)d_{\hat{p}q}$.*

Lemma 4. *For any $p \in R$ and $q \in [r, \hat{p}]$, it follows that $w_{\hat{p}} - w_q \le (D-1)d_{\hat{p}q}$.*

To prove (4), we utilize a relation between the increased amount of the work function and its one-sided derivatives, which are defined as $m_{p-0} := \lim_{q \to p-0} \frac{w_q - w_p}{d_{pq}}$ and $m_{p+0} := \lim_{q \to p+0} \frac{w_q - w_p}{d_{pq}}$ for any $p \in R$. The following lemma guarantees that $w_u = \text{OPT}_u$ for any $u \in V$, the derivatives exist and are integers, and that w_p is not convex for any interval not containing a node of V.

[1] One might expect that a continuous tree instead of a continuous ring would be preferable in terms of scalability of the network. However, this idea would fail because such a tree has the center, i.e., a point near to three nodes, which makes a work function extended on the continuous tree smaller than the original work function at some nodes.

Lemma 5. *The following claims hold.*

1. *For any $p \in R$ with $\hat{p} \neq p$, it follows that $\hat{p} \in V$.*
2. *For any $p \in R$, m_{p-0} and m_{p+0} are integers with $-D \leq m_{p\pm 0} \leq D$.*
3. *For any $p \in R \setminus V$, it follows that $m_{p-0} + m_{p+0} \leq 0$, i.e., w_p is convex only in the region containing a node in V.*

For $u \in V \setminus \{s\}$, let $m_{s\to u} := \lim_{q\to u,\ q\in[s,u)} \frac{w_q - w_u}{d_{uq}}$ and $m_s := \min\{m_{s\to u} \mid u \in V \setminus \{s\}\}$. Now we state our main lemma, which claims (4) together with two other claims.

Lemma 6. *The following claims hold.*

1. *For $\{p, q\} := V \setminus \{s\}$, $w_p \geq D(L - d_{sp}) + M$, or $w_q \geq D(L - d_{sq}) + M$, or $w_p + w_q \geq m_s d_{pq} + DL + 2M$.*
2. *For any $u \in V$, $w_u + w_{\bar{u}} \geq w_s + \frac{DL}{2} + M$.*
3. *For any $u \in V$, $w_u \geq D d_{su} + M$.*

Proof Sketch. We describe a proof sketch for this lemma. Through the extension of networks and work functions to continuous ones, we can obtain a sufficient condition for (4):

$$w_u + w_{\bar{u}} \geq w_s + \tfrac{DL}{2} + M \text{ for any } u \in V. \tag{5}$$

Actually, if (5) holds, then it follows that $w_u \geq w_s - w_{\bar{u}} + \frac{DL}{2} + M \geq -D d_{s\bar{u}} + \frac{DL}{2} + M = -D(\frac{L}{2} - d_{su}) + \frac{DL}{2} + M = D d_{su} + M$. Here, we have used the fact $w_s - w_{\bar{u}} \geq -D d_{s\bar{u}}$ (Lemma 1). We will prove (5) by induction on events of service and migration of WFA for requests. The inductive proof for a WFA's migration is easy because a WFA's migration of distance d decreases w_s by Dd, increases M by Dd, and does not change the left hand side of (5). As for the proof for a WFA's service, (5) can inductively be proved for most cases using basic properties of w_u (Lemmas 1–5), some of which are properties of w_u's slope defined using one-sided derivatives. However, there is one exceptional case for which (5) cannot be proved inductively. For example, if $\hat{u} = u \neq s$ and a decline from $\bar{u}$ to the request node $r \in V \setminus \{s, u\}$ has slope D, then w_u increases by d_{ur}, whereas $w_{\bar{u}}$ does not increase. Therefore, if $\hat{s} = s$ and $d_{sr} > d_{ur}$, then it is the case that the increased amount d_{ur} of $w_u + w_{\bar{u}}$ is less than the increased amount d_{sr} of w_s.

To prove (5) even for such a case, we need yet another condition as follows:

$$\text{For } \{p, q\} := V \setminus \{s\}, w_p \geq D(L - d_{sp}) + M, \text{ or } w_q \geq D(L - d_{sq}) + M, \text{ or } w_p + w_q \geq m_s d_{pq} + DL + 2M. \tag{6}$$

The first and second inequalities mean that w_p or w_q are already large enough, and therefore, (5) is satisfied for p or q[2], respectively. Actually, if the first inequality holds, then it follows that $w_p + w_{\bar{p}} \geq D(L - d_{sp}) + M + w_s - D d_{s\bar{p}} =$

[2] To be accurate, we should prove (5) for both p and q. Although we do not mention the reason here, we note that one of the first and second inequalities of (6) suffices.

$w_s + \frac{DL}{2} + M$. Here, we have used the fact $w_{\bar{p}} - w_s \geq -Dd_{s\bar{p}}$ (Lemma 1). The parameter m_s in the third inequality of (6) is the smaller slope at w_p toward s and at w_q toward s. Roughly speaking, m_s is increased by requests from p or q and counts up to D how a possible situation for which (5) cannot be proved inductively is approaching. However, the third inequality of (6) with $m_s = D$ implies (5) because $w_p + w_q \geq Dd_{pq} + DL + 2M = D(2L - d_{sp} - d_{sq}) + 2M$, implying the first or second inequality of (6). Condition (6) is proved inductively, together with induction hypothesis of (4), and hence that of (5). Thus, (4)–(6) are proved simultaneously in the formal proof.

By Lemma 6, we have (4), and hence Theorem 1.

4 Counter-Based Algorithm

In this section we design a counter-based algorithm called CBA and prove the following theorems:

Theorem 2. CBA *is* 3*-competitive on three nodes if* $D \leq 2$.

Theorem 3. CBA *is* ρ*-competitive on complete graphs with edge-weights between* 1 *and* $2 - 2/\rho$ *for any* $\rho \geq 3$.

4.1 Ideas and Definition of Algorithm

To describe ideas of our algorithm, we review the 3-competitive algorithm[3] for uniform networks presented in [8]. This algorithm, called COUNT, maintains a counter $C_v \geq 0$ for each node v so that $\sum_{v \in V} C_v = 2D$, and that the server of COUNT always has a positive counter. Initially, the server has a counter of $2D$, and the other nodes have counters of 0. If a request is issued on a node other than the server, then COUNT decrements a positive counter of a node by 1 and increments the counter of the request node by 1. If a counter becomes $2D$, then COUNT moves the server to the node with this counter. The fact that COUNT is 3-competitive is proved by verifying that for each event of COUNT's migration, OPT's migration, and services of COUNT and OPT for a request,

$$f := \Delta\text{COUNT} + \Delta\Phi - \rho\Delta\text{OPT} \leq 0 \tag{7}$$

is satisfied for $\rho = 3$. Here, Φ is a *potential function* of counters and the servers s and t of CBA and OPT, respectively, and defined as follows:

$$\Phi := \tfrac{\rho}{2} \sum_{v \in V} C_v d_{tv} + \left(\tfrac{\rho}{2} - 1\right) \sum_{v \in V} C_v d_{sv}.$$

ΔCOUNT, ΔOPT, $\Delta\Phi$ are the amounts of change of COUNT's cost, OPT's cost, and Φ in the event, respectively. Since $\Phi \geq 0$, by summing (7) overall events, we can prove that COUNT is ρ-competitive.

[3] Although the algorithm described here is slightly modified, it is essentially same as the original version.

Our algorithm is based on the following observations. COUNT is ρ-competitive even if the network has edge-weights between 1 and $2-2/\rho$. This can be proved by verifying that for the service event of COUNT and OPT for a request on r, if COUNT decrements the counter of a node u with $d_{sr} \leq (1-2/\rho)d_{su}+d_{ur}$, then (7) is satisfied. As for the migration event of COUNT or OPT, (7) is satisfied regardless of the structure of the network because COUNT always moves the server from a node of counter 0 to a node with counter $2D$. Therefore, if the server is located at a node s satisfying

$$d_{sv} \leq (1-\tfrac{2}{\rho})d_{su} + d_{uv} \text{ for any } u, v \in V \setminus \{s\}, \tag{8}$$

then (7) is satisfied for any event considered here.

If the server is located at a node s not satisfying (8), then it may be the case that $f > 0$. We shall amortize the excessive debt. Let A be the set of nodes satisfying (8) and B be the set of nodes not contained in A. If the network is uniform, then all the nodes are contained in A. Now we recall the assumption in Sect. 2 that the graph has a node set $V = \{a, b, c\}$ and edge-weights $x = d_{ab}$, $y = d_{ac}$, and $z = d_{bc}$. In this section, moreover, we assume without loss of generality that $y \geq \max\{x, z\}$. Then, it follows that $b \in A$, and hence, $B \subseteq \{a, c\}$. This is because $x - (\frac{L}{2} - \frac{z}{\rho}) = x - (\frac{L}{2} - \frac{L-x-y}{\rho}) \leq (1-\frac{2}{\rho})(x - \frac{L}{2}) \leq 0$, and similarly, $z - (\frac{L}{2} - \frac{x}{\rho}) \leq 0$.

We design our algorithm CBA by introducing the following policy to COUNT. If the server is in B, say $a \in B$, then CBA always decrements a's counter for a request on b or c and increments the counter of the request node. With this policy, (7) is satisfied for any service event. However, this policy may cause a situation that the counters of both b and c are less than $2D$ when a's counter becomes 0. This situation forces CBA to move the server to b or c, which may cause $f > 0$. Precisely, f depends on the position of the server t of OPT and distribution of values of the counters. If the counter of c is sufficiently large, then the excessive debt for the migration from a to c can entirely be amortized by the sum of f associated with service events between the previous and current migrations. Otherwise, although the excessive debt for the migration from a to b may still remain through the previous service events, it can be amortized by the sum of f associated with service events and a possible OPT's migration between the current and next migrations of CBA. CBA determines the destination of the migration by estimating the excessive debt for the migration and the amount that can amortize the debt.

Now we formally define CBA. We divide the input sequence of clients into phases so that a migration of CBA ends the current phase. When a new phase begins, CBA sets the counter of the previous server to 0. We define a function $\Psi_{st} \leq 0$ of counters of the the servers s and t of CBA and OPT, respectively, at the end of a phase, i.e., just after the migration of CBA to s. If $B = \emptyset$, then $\Psi_{st} := 0$ for any s and t. Otherwise,

$$\Psi_{st} := 0 \text{ if } s \in \{a, c\},$$
$$\Psi_{bv} := \max\{C_{\bar{v}}(-(\tfrac{\rho}{2} - 1)d_{b\bar{v}} - \tfrac{\rho}{2}(d_{v\bar{v}} - d_{bv})),$$
$$C_b\tfrac{\rho}{2}(d_{b\bar{v}} - d_{vb} - d_{v\bar{v}}) - (\rho - 3)Dd_{b\bar{v}})\}, \text{ and}$$
$$\Psi_{bb} := \Psi_{b\bar{v}} := \max\{C_{\bar{v}}(-(\tfrac{\rho}{2} - 1)d_{b\bar{v}} - \tfrac{\rho}{2}(d_{v\bar{v}} - d_{bv})), -(\rho - 3)Dd_{b\bar{v}})\},$$

where $\{v, \bar{v}\} = \{a, c\}$ with $C_v = 0$.

If a request is issued from a node r, then CBA performs the following procedure unless $r = s$.

1. If $s \in A$ and there exists $\bar{r} \in V \setminus \{s, r\}$ with $C_{\bar{r}} \geq 1$, then $C_{\bar{r}}{-}{-}$ and $C_r{+}{+}$. Otherwise, $C_s{-}{-}$ and $C_r{+}{+}$.
2. If $C_s = 0$, then move the server as follows:
   (a) If $s \in A$, then move the server to r. Step 1 implies $C_r = 2D$ in this case.
   (b) If $s \in B$ and $F_b \leq F_{\bar{s}}$ (F is defined later), move the server to b, where $\bar{s} \in V \setminus \{s, b\}$. It should be noted that $\{s, \bar{s}\} = \{a, c\}$.
   (c) If $s \in B$ and $F_b > F_{\bar{s}}$, then move the server to $\bar{s}$, and set $C_b := 0$ and $C_{\bar{s}} := 2D$.

Here, for $p \in \{b, \bar{s}\}$,

$$F_p := \max_{t,q \in V}\{M_{pq} + S_q + \Psi_{pq} - \Psi'_{st}\},$$
$$M_{bq} := -(\rho - 3)Dd_{sb} + (\rho - 2)C_{\bar{s}}(\tfrac{L}{2} - d_{s\bar{s}}) \text{ for } q \in V,$$
$$M_{\bar{s}q} := -(\rho - 3)Dd_{s\bar{s}} + C_b((\tfrac{\rho}{2} - 1)(d_{s\bar{s}} - d_{sb}) + \tfrac{\rho}{2}(d_{\bar{s}q} - d_{bq})) \text{ for } q \in V,$$
$$S_s := 0, \text{ and}$$
$$S_q := \max\{-\rho C_{\bar{s}}(\tfrac{L}{2} - d_{s\bar{s}}), -\rho C_b(\tfrac{L}{2} - d_{sb}), -\rho C_b(\tfrac{L}{2} - d_{b\bar{s}})\} \text{ for } q \in \{b, \bar{s}\}.$$

We have used Ψ' to denote Ψ associated with the previous phase and migration. If the current phase is the first phase, then Ψ' is defined using the initial server and counters. Moreover, Ψ_{pq} is associated with the current phase and migration. It should be noted that Ψ_{pq} can be computed just before the migration of CBA to p. This is because CBA changes no counters if $p = b$, and because $\Psi_{aq} = \Psi_{cq} = 0$.

The intuitions of Φ, Ψ, M, and S are as follows: Φ is actually a refined version of the potential function used in [8]. Ψ is the deposit saved in the next phase that can be used to amortize the current phase. M and S are corrections of Φ in a phase, i.e., upper bounds of increase of (CBA's cost) $+ \Phi - \rho \cdot$ (OPT's cost) for services and migration of CBA, respectively. We also note that if all the nodes are contained in A, then CBA can be viewed as a kind of COUNT, and hence, it works even on a network with more than three nodes.

4.2 Competitiveness

For any event e, let $\Delta\text{CBA}(e)$ and $\Delta\text{OPT}(e)$ be the costs of CBA and OPT for e, respectively. Moreover, let $\Delta\Phi(e)$ be the amount of change of Φ for e. Furthermore, let $f(e) := \Delta\text{CBA}(e) + \Delta\Phi(e) - \rho\Delta\text{OPT}(e)$. We will omit e in the notations

if e is clear from the context. Lemmas 7–9 below prove $f \leq 0$ with $\rho = 3$ for most cases, except for the case that CBA moves the server in Step 2b or 2c. These lemmas also imply that we can save some deposit (as Ψ and S), and will be used to prove that the deposit can entirely amortize the excessive dept (M) for the migration in Step 2b or 2c.

Lemma 7. *Suppose that* CBA *and* OPT *serve a request from $r \in V$ with the servers on s and t, respectively. If $r = s$, then $f = -\rho d_{rt} \leq 0$. If $r \neq s$, $s \in A$, and $C_{\bar{r}} \geq 1$, then $f \leq \frac{\rho}{2}(d_{rs} - d_{r\bar{r}}) - (\frac{\rho}{2} - 1)d_{s\bar{r}} \leq 0$, where $\bar{r} \in V \setminus \{s, r\}$. Otherwise, $f = \frac{\rho}{2}(d_{rs} - d_{rt} - d_{st}) \leq 0$.*

Lemma 8. *If* OPT *moves the server from t to q, then $f = \frac{\rho}{2}\sum_{v \in V} C_v(d_{qv} - d_{tv} - d_{tq}) \leq 0$.*

Lemma 9. *Suppose that* CBA *moves the server from s to p. If the server is moved in Step 2a, then $f = -(\rho - 3)Dd_{sp}$. If the server is moved in Step 2b or 2c, then $f = M_{pq}$, where q is the server of* OPT *at the migration of* CBA*. In particular, if $C_p = 2D$, then $f = -(\rho - 3)Dd_{sp}$ for any case.*

Fix a phase, and let ϕ be the sequence of events in the phase consisting of services of CBA and OPT for a request, migrations of OPT, and a migration of CBA. Suppose that CBA and OPT locate the servers s and t, respectively, at the beginning of the phase, and at p and q, respectively, at the end of the phase. We will prove $g := \sum_{e \in \phi} f(e) + \Psi_{pq} - \Psi'_{st} \leq 0$. If this holds, then because both Φ and Ψ can be bounded from below independently of the number of requests, we can prove that CBA is ρ-competitive by summing up the inequalities overall phases. In what follows, C_v denotes the counter of $v \in V$ just before CBA moves the server to p. This means that $C_s = 0$.

If $B = \emptyset$ or $s \in \{a, c\} \cap A$, then $C_p = 2D$ as mentioned in Step 2a of the definition of CBA, and $\Psi'_{st} = 0$. Therefore, $g \leq 0$ by $\Psi_{pq} \leq 0$ and Lemmas 7–9. These arguments imply the following lemma:

Lemma 10. CBA *is ρ-competitive on networks with three or more nodes all of which are contained in A.*

If a complete graph has edges of weights between 1 and $2 - 2/\rho$, then all the nodes are contained in A. Therefore, we have Theorem 3.

To prove Theorem 2, it remains to prove that $g \leq 0$ for the case $B \neq \emptyset$ and $s \in \{b\} \cup B$. In what follows, we set $\rho := 3$ for simplicity.

Lemma 11. *If $s = b$, then $g \leq 0$.*

We prove $g \leq 0$ for the remaining case $s \in \{a, c\} \cap B$ in the following.

Lemma 12. *If $s \in B$, then $\sum_{e \in \phi} f(e) \leq M_{pq} + S_q$.*

Lemma 13. *If $D \leq 2$ and $s \in \{a, c\} \cap B$, then $F_b \leq 0$ or $F_{\bar{s}} \leq 0$.*

By Lemmas 11–13, we have $g \leq 0$ for every case. Therefore, the proof of Theorem 2 is completed.

5 Lower Bound

In this section we prove the following theorem:

Theorem 4. *If a deterministic page migration algorithm is ρ-competitive on three nodes, then $\rho \geq 3 + \Omega(1/D)$. In particular, $\rho > 3$ for any $D \geq 3$.*

5.1 Adversary

To prove Theorem 4, we design a 3-node network and an *adversary*, i.e., a strategy to generate an arbitrarily long sequence σ of clients against any deterministic online page migration algorithm ALG on the network so that $\text{ALG}(\sigma) > \rho \cdot \text{OPT}(\sigma)$ for some $\rho = 3 + \Omega(1/D)$. By using such a strategy, we obtain a lower bound of ρ, i.e., $\text{ALG}(\sigma) \geq \rho \cdot \text{OPT}(\sigma) + \alpha$ for any α independent of the number of clients because σ can be arbitrarily long. Broadly, our strategy repeatedly generates a sequence ϕ of clients so that ALG returns the server to the initial position s_0 after processing each ϕ, and that $\text{ALG}(\phi) > (3 + \Omega(1/D))\text{OPT}_{s_0}(\phi)$. The sequence ϕ begins with a sequence τ such that $\text{ALG}(\tau) > (3 + \Omega(1/D))\text{OPT}(\tau)$, or that ALG moves the server too early to achieve a competitive ratio $3 + o(1/D)$. Unless ALG locates the server at s_0 after processing τ, a subsequent sequence τ' leads ALG to return the server to s_0.

In this section we assume without loss of generality that $y \geq x \geq z$. We denote a sequence χ of clients also by χ_v if ALG leaves the server on a node v after processing χ. The following Lemma 14 provides a sufficient condition for ALG to move the server too early to achieve a competitive ratio of ρ.

Lemma 14. *Let $P \subseteq V$, $Q := V \setminus P$, and let $p \in P$ and $q \in Q$ be joined by an edge with the minimum weight w overall edges joining P and Q. If there exist $\rho > 3$ and a sequence χ_q of clients such that $(\rho - 1)\text{OPT}_p(\chi_q) + \text{OPT}_q(\chi_q) - \text{ALG}(\chi_q) + (\rho - 5)Dw < 0$, then there exists a sequence $\chi' = \chi'_p$ with $\text{ALG}(\chi_q\chi') > \rho \cdot \text{OPT}_p(\chi_q\chi')$ or a sequence $\chi'' = \chi''_q$ with $\text{ALG}(\chi_q\chi'') > \rho \cdot \text{OPT}_q(\chi_q\chi'')$.*

Lemmas 15 and 16 below are tools to generate a sequence that leads ALG to return the server to the initial position.

Lemma 15. *Let $p := a$ and $q := b$, or $p := b$ and $q := c$. Let $w := d_{pq}$. If there exist $\rho > 3$, $\beta > 0$, and a sequence χ_q of clients such that $\text{ALG}(\chi_q) > \rho \cdot \text{OPT}_q(\chi_q)$ and $\text{OPT}_q(\chi_q) \geq \beta Dw$, then there exists a sequence χ' such that $\chi' = \chi'_p$ and $\text{ALG}(\chi_q\chi') > \rho' \cdot \text{OPT}_p(\chi_q\chi')$, or that χ' is an arbitrarily long sequence with $\text{ALG}(\chi_q\chi') > \rho' \cdot \text{OPT}(\chi_q\chi')$, where $\rho' := \frac{\beta}{\beta+4}(\rho - 3) + 3$.*

Lemma 16. *Let $\{p, q\} := \{a, b\}$ and $w := d_{pq}$. If there exist $\rho > 3$, $\beta > 0$, and a sequence χ_q of clients such that $(\rho - 1)\text{OPT}_p(\chi_q) + \text{OPT}_q(\chi_q) - \text{ALG}(\chi_q) + (\rho - 5)Dw < 0$ and $\text{OPT}_q(\chi_q) \geq \beta Dw$, then there exists a sequence χ' such that $\chi' = \chi'_a$ and $\text{ALG}(\chi_q\chi') > \rho' \cdot \text{OPT}_a(\chi_q\chi')$, or that χ' is an arbitrarily long sequence with $\text{ALG}(\chi_q\chi') > \rho' \cdot \text{OPT}(\chi_q\chi')$, where $\rho' := \frac{\beta}{\beta+4}(\rho - 3) + 3$.*

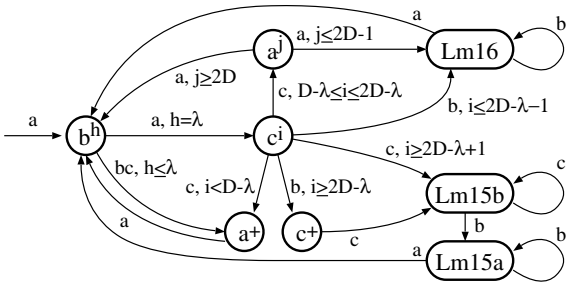


Fig. 1. Strategy to generate σ

We set the initial server $s_0 := a$. Our strategy to generate σ is defined using a state machine as shown in Fig. 1. In this state machine, a transition represents a server selected by ALG, together with optional conditions on the number of requests generated in the source state. The parameter $1 \leq \lambda < D$ will be defined later. A state with the form of u^k (i.e., b^h, a^j, and c^i) represents a sequence of requests from u until one of the outgoing arcs from the state meets the server of ALG and the conditions on the number k of the requests. A state with the form of u^+ (i.e., a^+ and c^+) represents a sequence of requests from u until ALG locates the server on u. The states Lm15b and Lm15a represent sequences of requests obtained by applying Lemma 15 with $p := b$ and $q := c$, and with $p := a$ and $q := b$, respectively. The state Lm16 represents a sequence of requests obtained by applying Lemma 16 with $p \in \{a, b\} \setminus \{s\}$ and $q := s$, where $s \in \{a, b\}$ is the server of ALG at the beginning of the state.

5.2 Analysis

Now we prove Theorem 4. Suppose that $y = x + \delta$ and $z = \gamma\delta$ with $\delta > 0$ and $3 \leq \gamma \leq x/\delta$. We will choose γ and δ later. We divide σ into phases so that entering the state b^h begins a new phase. ALG locates the server on a at the beginning of each phase. Therefore, Theorem 4 is proved if for each phase $\phi = \phi_a$, $\text{ALG}(\phi) > \rho \cdot \text{OPT}_a(\phi)$ with the initial server on a, and if for a phase $\phi \neq \phi_a$ (i.e., an arbitrarily long sequence), $\text{ALG}(\phi) > \rho \cdot \text{OPT}(\phi)$ with the initial server on a. This can be done for $\rho = 3 + \Omega(1/D)$ by carefully choosing $\delta = O(x/D)$, $\gamma = O(1)$, and $\lambda = \Theta(D)$. The detailed analysis is omitted here and will be given in a journal version.

6 Future Work

It would be interesting to answer whether or not there exists an asymptotically 3-competitive deterministic algorithm on a broader class of networks. Unfortunately, even 4-node ring networks do not allow WFA as it is to have such a competitive ratio. In fact, our proof of Theorem 1 depends on the fact that an

extended work function is not convex on the interval between two nodes other than the server on a continuous ring with three nodes (Claim 3 of Lemma 5). However, this fact does not follow for four nodes. On the other hand, there might exist a lower bound of $3 + \Theta(1)$ on general networks. For such a lower bound, however, we would need at least four nodes and have to overcome the difficulty of designing and analyzing a much more complicated adversary mainly due to increase of nodes. In any case, improving the currently best upper bound of 4.086 on general networks is still an important open problem.

References

1. Awerbuch, B., Bartal, Y., Fiat, A.: Distributed paging for general networks. J. Algorithms 28(1), 67–104 (1998)
2. Awerbuch, B., Bartal, Y., Fiat, A.: Competitive distributed file allocation. Information and Computation 185(1), 1–40 (2003)
3. Bartal, Y., Charikar, M., Indyk, P.: On page migration and other relaxed task systems. Theoret. Comput. Sci. 268(1), 43–66 (2001)
4. Bartal, Y., Fiat, A., Rabani, Y.: Competitive algorithms for distributed data management. J. Comput. Sys. Sci. 51(3), 341–358 (1995)
5. Bein, W.W., Chrobak, M., Larmore, L.L.: The 3-server problem in the plane. Theoret. Comput. Sci. 289, 335–354 (2002)
6. Bienkowski, M.: Migrating and replicating data in networks. Comput. Sci. Res. Dev. (2011), doi:10.1007/s00450-011-0150-8
7. Bienkowski, M., Byrka, J., Korzeniowski, M., Meyer auf der Heide, F.: Optimal algorithms for page migration in dynamic networks. J. Discrete Algorithms 7(4), 545–569 (2009)
8. Black, D.L., Sleator, D.D.: Competitive algorithms for replication and migration problems. Tech. Rep. CMU-CS-89-201, Department of Computer Science, Carnegie Mellon University (1989)
9. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
10. Chrobak, M., Larmore, L.L., Reingold, N., Westbrook, J.: Page migration algorithms using work functions. J. Algorithms 24(1), 124–157 (1997)
11. Karlin, A., Manasse, M., Rudolph, L., Sleator, D.: Competitive snoopy caching. Algorithmica 3(1), 79–119 (1988)
12. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. J. ACM 42, 971–983 (1995)
13. Lund, C., Reingold, N., Westbrook, J., Yan, D.: Competitive on-line algorithms for distributed data management. SIAM J. Comput. 28(3), 1086–1111 (1999)
14. Matsubayashi, A.: Uniform page migration on general networks. International Journal of Pure and Applied Mathematics 42(2), 161–168 (2007)
15. Westbrook, J.: Randomized algorithms for multiprocessor page migration. In: DIMACS. Discrete Mathematics and Theoretical Computer Science, vol. 7, pp. 135–150 (1992)

R–LINE: A Better Randomized 2-Server Algorithm on the Line

Lucas Bang, Wolfgang Bein, and Lawrence L. Larmore

Department of Computer Science,
University of Nevada Las Vegas, Nevada 89154, USA
{bang,beinw}@unlv.nevada.edu, larmore@cs.unlv.edu

Abstract. A randomized on-line algorithm is given for the 2-server problem on the line, with competitiveness less than 1.901 against the oblivious adversary. This improves the previously best known competitiveness of $\frac{155}{78} \approx 1.987$ for the problem.

1 Introduction

In the k-server problem, there are k identical mobile servers in a metric space M. At any time, a point $r \in M$ can be "requested," and must be "served" by moving one of the k servers to the point r. The cost of that service is defined to be the distance the server is moved; for a sequence of requests the goal is to serve the requests at small cost. An *online algorithm* for the server problem decides, at each request, which server to move, but does not know the sequence of future requests. We analyze an online algorithm for the server problem in terms of its *competitive ratio*, which essentially gives the ratio of its cost over the cost of an optimal (offline) algorithm which has knowledge of the entire request sequence before making any decisions. More precisely, we say that an online algorithm $\mathcal{A}$ for the server problem is C*-competitive*, if there is a constant K, such that, given any request sequence σ, $cost_{\mathcal{A}}(\sigma) \leq C \cdot cost_{\mathcal{OPT}}(\sigma) + K$, where $cost_{\mathcal{OPT}}(\sigma)$ is the minimum possible cost of any service of σ. If $\mathcal{A}$ is a randomized online algorithm, we express the inequality in terms of expected cost, *i.e.*, $E(cost_{\mathcal{A}}(\sigma)) \leq C \cdot cost_{\mathcal{OPT}}(\sigma) + K$. In the analysis of an online algorithms it is customary to think of the optimal service as performed by an *oblivious adversary*. The optimal cost is then also referred to as the "cost of the adversary," and the movement of the servers in the optimal algorithm as "adversary moves."

The server problem was first proposed by Manasse, McGeoch and Sleator [13] and the problem has been widely studied since then. They also introduced the now well-known k*-server conjecture*, which states that, for each k, there exists an online algorithm for k servers which is k-competitive for any metric space. The conjecture was immediately proved true by the same researchers for $k = 2$, but for larger k, the conjecture remains open, although it has been proved for a number of special classes of metric spaces, such as trees [10], spaces with at most $k + 2$ points [12], and the Manhattan plane for $k = 3$ [6].

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 120–130, 2013.

In the randomized case, little is known. Bartal *et al.* [4] have an asymptotic lower bound, namely that the competitiveness of any randomized online algorithm for an arbitrary metric space is $\Omega(\log k / \log^2 \log k)$. It is conjectured that there is an $O(\log k)$ competitive algorithm for general metric spaces. A recent breakthrough is the algorithm by Bartal *et al.* [2], which gives a poly-logarithmic competitive algorithm for finite metric spaces.

Surprisingly, no randomized competitive algorithm for the 2-server problem for general spaces is known to have competitiveness less than 2, although that barrier has been broken for a number of classes of spaces. The competitiveness is known to be $\frac{3}{2}$ for uniform spaces, and Bein *et al.* [7] have shown that there is a randomized algorithm with competitive ratio of at most 1.5897 for all 3-point spaces. Bein *et al.* [8] have recently given a "better than 2" competitive algorithm for crosspolytope spaces using knowledge states [9]. A lower bound of $1 + e^{-1/2} \approx 1.606$ has been shown [11].

We define the (m, n)-*server problem*, for $m > n$, to be the variation where there are m mobile servers in the metric space, and each request must be served by at least n of them. For the 2-server problem on the line, Bartal *et al.* give a barely random online algorithm for the 2-server problem on the line, with competitive ratio $\frac{155}{78} \approx 1.987$ [5]; their method is to define a deterministic online algorithm for the $(6, 3)$-server problem with that competitiveness, from which three deterministic online algorithms are defined. The randomized algorithm is simply to pick one of those three at random, each with probability $\frac{1}{3}$, and then use the chosen algorithm for the entire request sequence.

Our contribution. In this paper, we give a randomized online algorithm for the $(2n, n)$-server problem on the line, for every $n \geq 3$. By Theorem 1 below, we obtain a randomized algorithm for the 2-server problem on the line. As n increases, the competitiveness of our algorithm decreases, and the limiting value is less than 1.901.

2 The Algorithm R–LINE

Our algorithm, R–LINE, is defined to be a randomized algorithm for the $(2n, n)$-server problem, for $n \geq 3$. We make use of the following two theorems from [5]:

Theorem 1. *Given any C-competitive online algorithm for the $(2n, n)$-server problem, we can derive a randomized online algorithm for the 2-server problem that is C-competitive.*

Theorem 2. *Any optimal offline strategy for the $(2n, n)$ server problem keeps the servers in two blocks of n each, assuming that the servers are together in two blocks in the initial configuration.*

By Theorem 2, without loss of generality we can assume that the adversary is using an optimal 2-server algorithm, but serves with cost equal to n times the distance moved. We will use the notation s_i both to refer to the i^{th} server and its location, when no confusion arises. We assume that $s_1 \leq s_2 \leq ... \leq s_{2n-1} \leq s_{2n}$.

We also refer to the adversary's servers as a_1 and a_2, and assume that $a_1 \leq a_2$. The algorithm thus knows the location of one of the adversary's servers, which we call the *visible* server, and which, by a slight abuse of notation, we also call r. We denote the adversary's other server by a, and refer to it as the *hidden* server, since the algorithm does not know where it is.

We define a configuration of servers (R–LINE's as well as the adversary's) to be *satisfying* if at least n of R–LINE's servers are at r. We refer to a satisfying configuration as an S-configuration, and we assume that the initial configuration is an S-configuration.

Every round begins by the adversary choosing a new request point r and moving one of its two servers to r. R–LINE then moves as many of its servers as necessary to r, and the resulting configuration is once again an S-configuration. No R–LINE server will pass another R–LINE server that does not serve. In general, R–LINE deterministically moves zero or more servers to r, and then uses randomization to decide which additional servers to move. R–LINE is *lazy*, meaning that it never moves any server that does not serve the request.

2.1 The Potential

The algorithm R–LINE is given based on a suitable potential, which is used in Section 3 to prove competitiveness. For each fixed $n \geq 3$, we define a competitiveness C for R–LINE as well as a potential ϕ on configurations. This potential will satisfy the following property:

Property 1. If ϕ is the potential at the configuration before a round and ϕ' the potential after the round, and if $cost_{\text{R–LINE}}$ and $cost_{\mathcal{A}dv}$ are the costs incurred by R–LINE and the adversary, respectively, then

$$E(cost_{\text{R–LINE}} + \phi' - \phi) \leq C \cdot cost_{\mathcal{A}dv}$$

where E denotes expected value.

Isolation Indices and Coefficients. For $0 \leq i \leq 2n$ and $0 \leq j \leq 2$, if $1 \leq i + j \leq 2n + 1$, we define $\alpha_{i,j}$, the $(i,j)^{\text{th}}$ *isolation index* of a configuration, to be the length of the longest interval that has exactly i algorithm servers to the left and exactly j adversary servers to the left. More formally,

$$\alpha_{i,j} = \max \begin{cases} \min\{s_{i+1}, a_{j+1}\} - \max\{s_i, a_j\} \\ 0 \end{cases}$$

where we let $s_0 = a_0 = -\infty$ and $s_{2n+1} = a_3 = \infty$ by default. Isolation indices are part of T-theory and a more general definition of isolation indices is given in [1].

For each $0 \leq i \leq 2n$ and $0 \leq j \leq 2$, we define a constant $\eta_{i,j}$, the $(i,j)^{\text{th}}$ *isolation index coefficient*. The isolation index coefficients satisfy a symmetry property, namely $\eta_{i,j} = \eta_{2n-i,2-j}$; furthermore, $\eta_{0,0} = \eta_{2n,n} = 0$. We formally define the potential of a configuration to be

$$\phi = \sum \{\eta_{i,j} \cdot \alpha_{i,j} \ : \ (0 \leq i \leq 2n) \wedge (0 \leq j \leq 2) \wedge (1 \leq i + j \leq 2n + 1)\}$$

Intuitively, $\eta_{i,j}$ is a weight on the isolation index $\alpha_{i,j}$ for any configuration. For each given n, the competitiveness C and the isolation index coefficients $\{\alpha_{i,j}\}$ must satisfy a system of inequalities given in Section 3.

We will first define R–LINE in terms of those constants, and then show that R–LINE is C-competitive if the system of inequalities is satisfied. In Section 4 we outline how to find a solution to these inequalities, and give the values of the constants for $n = 3$.

2.2 Algorithm Description

We now define R–LINE. Between rounds, the configuration of servers is always an S-configuration. When the adversary makes a request at a point r, R–LINE responds by making a sequence of *moves*, each consisting of the movement of one or more servers to r. Thus, during a round, R–LINE makes at most n moves. Not all configurations can arise during execution of R–LINE; in fact, we define two classes of configurations, D-configurations and R-configurations, such that every intermediate configuration of R–LINE belongs to one of those two classes. If the current configuration is a D-configuration, then R–LINE's next move is to move one or more servers deterministically to r, while if the current configuration is an R-configuration, then R–LINE's next move is to choose, using randomization, a set of servers to move to r. In this case there are always two choices – to move one or more servers from the previous request point to r, completing the round, or to move just one server from the other side, possibly not completing the round.

We now define the classes of configurations. Note that, before the current round began, there must have been n algorithm servers at the previous request point, which we call r'. Without loss of generality, $r' \neq r$.

1. **S-Configuration:** there are n algorithm servers at r.
2. **D-Configuration:** the following two conditions hold.
   (a) There are more than n algorithm servers either strictly to the left or strictly to the right of r; that is, $r > s_{n+1}$ or $r < s_n$.
   (b) If there are fewer than n algorithm servers at r', then there is no algorithm server strictly between r' and r, and furthermore, there are at least n algorithm servers at the points r' and r combined.
3. **R-Configuration:**
   (a) There are exactly n algorithm servers on the same side of r as r', that is, either $r' = s_n < r$ or $r < r' = s_{n+1}$.
   (b) There is no algorithm server strictly between r' and r, and furthermore, there are at least n algorithm servers at the points r' and r combined.

We now give an explicit definition of R–LINE. By symmetry, we can assume, without loss of generality, that $r' < r$. The reader might also consult Figure 1 where we illustrate R–LINE through a single round, in a case where $n = 3$.

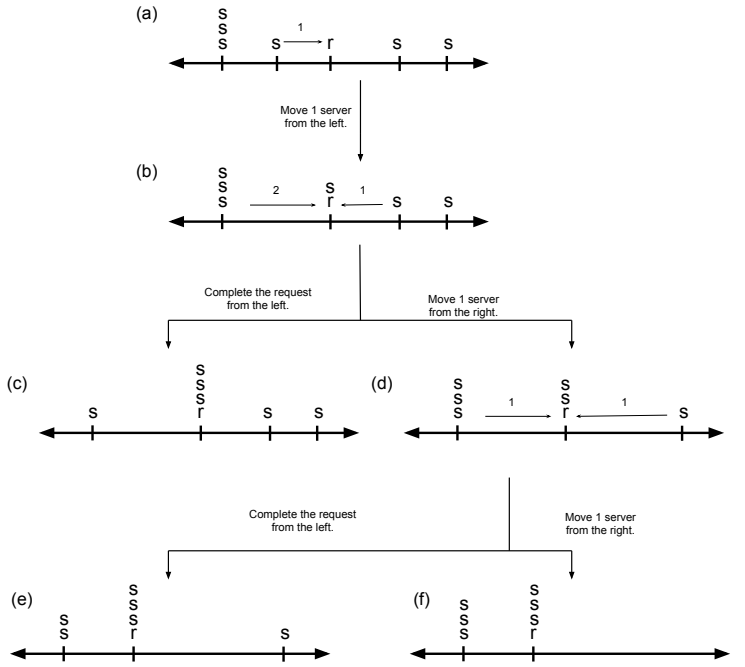


Fig. 1. (a) A D-configuration, where $n = 3$. The request is r, there are three servers located at $r' < r$. The next move is deterministic. (b) An R-configuration. One server has moved to r from the left. The next move is randomized; either move two servers from the left or one from the right. (c) An S-configuration, after two servers moved from the left. The round is over. (d) An R-configuration, after one server moved from the right. The next move is randomized; either move one server from the left or one from the right. (e) An S-configuration, after one server moved from the right. The round is over. (f) An S-configuration, after one server moved from the left. The round is over.

1. If the current configuration is a D-configuration, then there are m algorithm servers to the left of r for some $m > n$. Move the servers $s_{n+1}, \ldots s_m$ to r. If the resulting configuration is an S-configuration, the round is over. Otherwise, the resulting configuration is an R-configuration, and proceed to the next step.
2. If the current configuration is an R-configuration, then $r' = s_n < r \leq s_{n+1} < s_{2n}$. Let p be the number of algorithm servers at r. Then $s_{n+p+1} > r$. R–LINE executes one of two moves; each move is executed with a probability that is determined by solving a 2-person zero-sum game. We compute those probabilities below. The two choices of move are:

   (a) Move s_{n+p+1} to r.
   (b) Move the servers $s_{p+1} \ldots s_n$ to r.

   If the resulting configuration is an S-configuration, the round is over. Otherwise, the resulting configuration is an R-configuration, and repeat this step.

For the randomized step, one of the two choices is selected by using the optimum strategy for a 2-person zero sum game, where R–LINE is the column player, and $\mathcal{A}dv$ is the row player; the choice of the row player is where to place the hidden server. As we show later, we can assume, without loss of generality, that the hidden server is located at either s_n or s_{n+p+1}. Thus, each player has exactly two strategies. Each entry of the payoff matrix is equal to $\Delta\phi + cost = \phi' - \phi + cost$, where ϕ and ϕ' are the potentials before and after the move; and *cost* is the cost of the move, which is equal to the number of servers moved times distance moved, either $(s_{n+p+1} - r)$ or $(n-p)(r-s_n)$.

The payoff matrix is as follows:

$$G =$$

	Move s_{n+p+1}	Move $s_{p+1} \dots s_n$
$a = s_n$	$(\eta_{n+p+1,2} - \eta_{n+p,2} + 1)(s_{n+p+1} - r)$	$(\eta_{p,1} - \eta_{n,1} + n - p)(r - s_n)$
$a = s_{n+p+1}$	$(\eta_{n+p+1,1} - \eta_{n+p,1} + 1)(s_{n+p+1} - r)$	$(\eta_{p,0} - \eta_{n,0} + n - p)(r - s_n)$

3 Proof of Competitiveness

We now present a system of inequalities, which we denote $\mathbb{S}$, which suffice for R–LINE to be C-competitive. We will prove, in Theorem 3, that $\mathbb{S}$ implies C-competitiveness of R–LINE.

$$\forall\ 0 \le i \le 2n:\ |\eta_{i,1} - \eta_{i,0}| \ \le\ n \cdot C \tag{1}$$

$$\forall\ 1 \le i \le n \text{ and } \forall\ 1 \le j \le 2:\ \eta_{i,j} + 1 \ \le\ \eta_{i-1,j} \tag{2}$$

$$\forall\ 1 \le i \le n \text{ and } \forall\ 1 \le j \le 2:\ \eta_{i-1,j-1} \ \le\ \eta_{i,j-1} + 1 \tag{3}$$

$$\forall\ 1 \le i \le n:\ (\eta_{i-1,1} - \eta_{i,1} + 1)(\eta_{n-i,1} - \eta_{n,1} + i) \ \le\ (\eta_{i-1,0} - \eta_{i,0} + 1)(\eta_{n-i,0} - \eta_{n,0} + i) \tag{4}$$

Theorem 3. *For any assignment of values to C and $\eta_{i,j}$ for $0 \le i \le 2n$ and $0 \le j \le 2$ that satisfies the system $\mathbb{S}$,* R–LINE *is C-competitive.*

We prove Theorem 3 with a sequence of lemmas. We will prove that if the system of inequalities $\mathbb{S}$ is satisfied, then the following properties hold. We write $\Delta\phi = \phi' - \phi$, where ϕ is the potential before the move and ϕ' is the potential after the move.

1. For any move by the adversary, $\Delta\phi \le C \cdot cost_{\mathcal{A}dv}$. (Recall that the adversary pays n times the distance moved.)
2. For any deterministic move by R–LINE, $\Delta\phi + cost \le 0$.
3. We may assume the adversary's hidden server is at one of at most two possible locations during a given round, namely at the closest algorithm server to either the left or the right of r.
4. For any randomized move by R–LINE, $\mathrm{E}(\Delta\phi + cost) \le 0$.

We say that a move is *simple* if the move consists of moving a single server (either an algorithm or an adversary server) across an interval, and there is no other server (of either type) located strictly between the end points of that interval.

We also refer to a simple move as a *step*; in general, every movement of servers is a concatenation of steps.

Lemma 1. *If $\mathbb{S}$ holds, then Property 1 holds.*

Proof. By the symmetry of the $\eta_{i,j}$, inequality (1) implies that $|\eta_{i,j} - \eta_{i,j-1}| \leq n \cdot C$ for $j = 1, 2$. Without loss of generality the move is simple, since every move which is not simple is the concatenation of steps. Without loss of generality, the adversary server a_j moves to the right, from x to y, where $x < y$. Since the move is simple, $s_i \leq x$ and $y \leq s_{i+1}$ for some $0 \leq i \leq 2n$,. (Recall the default values $s_0 = -\infty$ and $s_{2n+1} = \infty$.) Thus, $\alpha_{i,j}$ decreases by $y - x$ and $\alpha_{i,j-1}$ increases by $y - x$. The cost to the adversary of this move is $n(y - x)$. By definition of the potential, $\Delta\phi = (\eta_{i,j} - \eta_{i,j-1})(y - x) \leq n \cdot C \cdot (y - x) \leq C \cdot cost_{\mathcal{A}dv}$.

Lemma 2. *If $\mathbb{S}$ holds, then Property 2 holds.*

Proof. For convenience, we assume that $r < r' = s_{n+1}$. There are exactly m algorithm servers to the right of r, for some $m > n$. Servers $s_{2n-m+1} \ldots s_n$ move to r. The move is the concatenation of steps, and it suffices to show that $\Delta\phi \geq cost_{\text{R–LINE}}$ for each of those steps.

Fix one step. During the step, s_i moves from x to y, where $y < x$, for some $2n - m + 1 \leq i \leq n$. The algorithm cost of the step is $x - y$. Pick the maximum j such that $a_j \leq y$. Since $r \leq y$, j is either 1 or 2. The move causes $\alpha_{i,j}$ to decrease by $x - y$ and $\alpha_{i-1,j}$ to increase by the same amount. By inequality (1), and the definition of the potential: $\Delta\phi + cost_{\text{R–LINE}} = (x - y)(\eta_{i,j} - \eta_{i-1,j} + 1) \leq 0$.

Lemma 3. *If $1 \leq i \leq 2n$ and $j = 1, 2$, then $\eta_{i,j} + \eta_{i-1,j-1} \leq \eta_{i,j-1} + \eta_{i-1,j}$*

Proof. Suppose $i \leq n$. Then $1 + \eta_{i,j} \leq \eta_{i-1,j}$ by (2), while $-1 + \eta_{i-1,j-1} \leq \eta_{i,j-1}$ by (3). Adding the two inequalities, we obtain the result.

If $i > n$, then $\eta_{2n-i+1,3-j} + \eta_{2n-i,2-j} \leq \eta_{2n-i+1,2-j} + \eta_{2n-i,3-j}$ by the previous case. By symmetry, we are done.

Lemma 4. *If $\mathbb{S}$ holds, then Property 3 holds.*

Proof. Since a could be any point on the line, the payoff matrix of the game has infinitely many rows. We need to prove that just two of those rows, namely $a = s_n$ and $a = s_{n+p+1}$, dominate the others.

By batching the row strategies, we illustrate the $\infty \times 2$ payoff matrix below.

		Move s_{n+p+1}	Move $s_{p+1} \ldots s_n$
I	$a \leq s_n$	$(\eta_{n+p+1,2} - \eta_{n+p,2} + 1)(s_{n+p+1} - r)$	$(\eta_{p,1} - \eta_{n,1} + n - p)(r - s_n)$
II	$s_n \leq a \leq r$	$(\eta_{n+p+1,2} - \eta_{n+p,2} + 1)(s_{n+p+1} - r)$	$(\eta_{p,1} - \eta_{n,1} + n - p)(r - a)$ + $(\eta_{p,0} - \eta_{n,0} + n - p)(a - s_n)$
III	$r \leq a \leq s_{n+p+1}$	$(\eta_{n+p+1,2} - \eta_{n+p,2} + 1)(s_{n+p+1} - a)$ + $(\eta_{n+p+1,1} - \eta_{n+p,1} + 1)(a - r)$	$(\eta_{p,0} - \eta_{n,0} + n - p)(r - s_n)$
IV	$a \geq s_{n+p+1}$	$(\eta_{n+p+1,1} - \eta_{n+p,1} + 1)(s_{n+p+1} - r)$	$(\eta_{p,0} - \eta_{n,0} + n - p)(r - s_n)$

The row strategy $a = s_n$ trivially dominates all row strategies in Batch I. It also dominates all row strategies in Batch II, because

$$\begin{aligned}\eta_{p,1} - \eta_{n,1} &= \sum_{i=p+1}^{n} (\eta_{i-1,1} - \eta_{i,1}) \\ &\geq \sum_{i=p+1}^{n} (\eta_{i-1,0} - \eta_{i,0}) \quad \text{by Lemma 3} \\ &= \eta_{p,0} - \eta_{n,0}\end{aligned}$$

The row strategy $a = s_{n+p+1}$ trivially dominates all row stages in Batch IV. It also dominates all row strategies in Batch III, because $\eta_{n+p+1,1} - \eta_{n+p,1} \geq \eta_{n+p+1,2} - \eta_{n+p,2}$, which we can similarly prove using Lemma 3.

We make use of a standard game theory lemma taken from [3]. To this end we remind the reader that a *saddle point* of a zero-sum game is defined to be an entry $a_{i,j}$ of the payoff matrix that is both a maximum of its row and a minimum of its column. If a game has a saddle point, then the value the game is the value of the saddle point, and it is optimum for the row player to always play the i^{th} row, and for the column player to always play the j^{th} column.

Lemma 5. *Suppose If* $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ *is the payoff matrix for a 2-person zero sum game* G*, and there is no saddle point. Then*

$$v(G) = \frac{\det A}{a_{11} - a_{12} - a_{21} + a_{22}}$$

Furthermore, the optimum strategy for the row player is:

Play row 1 with probability $\frac{a_{22}-a_{21}}{a_{11}-a_{12}-a_{21}+a_{22}}$

Play row 2 with probability $\frac{a_{11}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}}$

While the optimum strategy for the column player is:

Play column 1 with probability $\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}}$

Play column 2 with probability $\frac{a_{11}-a_{21}}{a_{11}-a_{12}-a_{21}+a_{22}}$

Lemma 6. *If* $\mathbb{S}$ *holds, then Property 4 holds.*

Proof. Consider the 2×2 payoff matrix G of Section 2.2. By $\mathbb{S}$, the upper left and lower right entries of G are negative, while the upper right and lower left entries are positive. By Theorem 5, the value of our game is

$$\frac{\det(G)}{(\eta_{n+p+1,2} + \eta_{n+p+1} - \eta_{n+p,2} - \eta_{n+p+1,1}) \cdot (s_{n+p+1} - r) + (\eta_{p,0} + \eta_{n,1} - \eta_{n,0} - \eta_{p,1}) \cdot (r - s_n)}$$

The numerator is non-negative by inequality 4. The denominator is negative, which we can prove by combining inequalities of $\mathbb{S}$ labeled (2) and (3). Thus, $E(\Delta\phi + cost_{\text{R–LINE}}) = v(G) \leq 0$ as claimed.

Theorem 3 follows immediately from Lemmas 1, 2, 4, and 6.

4 Finding a Solution to the Inequalities

We need to find a solution to the system $\mathbb{S}$ for which C is smaller than 2, preferably as small as possible. We first reduce $\mathbb{S}$ to a system, which we call $\mathbb{S}'$, that is easier to work with.

Let $n \geq 3$ be fixed. We introduce variables ϵ_i and δ_i for all $0 \leq i < n$. Our system $\mathbb{S}'$ consists of the following constraints.

$$\begin{aligned}
(2i + \epsilon_{n-i})(2 - \epsilon_i + \epsilon_{i-1}) &= 4i && \forall\ 0 < i < n \\
(2n + \epsilon_0)(2 + \epsilon_{n-1}) &\geq 4n \\
\delta &= -\epsilon_0/2 \\
C &= (2n - \delta)/n \\
\delta_i &= \epsilon_i + 2\delta && \forall\ 0 \leq i < n \\
\alpha_{i,0} &= 3i - \delta_i && \forall\ 0 \leq i < n \\
\alpha_{i,0} &= 2n + i - 2\delta && \forall\ n \leq i \leq 2n \\
\alpha_{i,1} &= 2n - i - \delta && \forall\ 0 \leq i < n \\
\alpha_{i,1} &= i - \delta && \forall\ n \leq i \leq 2n \\
\alpha_{i,2} &= \alpha_{2n-i,0} && \forall\ 0 \leq i \leq 2n
\end{aligned}$$

A solution to the system $\mathbb{S}'$. also provides a solution to $\mathbb{S}$.

4.1 The Case $n = 3$

For $n = 3$ the competitiveness of R–LINE can be calculated in closed form. The solution to $\mathbb{S}'$ for which C is minimum can be obtained by 4th degree polynomial. For $n = 3$, R–LINE has competitiveness $1 + \frac{\sqrt{71+17\sqrt{17}}}{12} \approx 1.98985407$. We also provide the values of the isolation index coefficients. The values of the constants $\delta, \delta_1, \delta_2$, and the $\eta_{i,j}$ are shown in the following tables.

Constants
$\delta = 3 - \frac{\sqrt{71+17\sqrt{17}}}{4} \approx 0.030437789$
$\delta_1 = 2 - \frac{\sqrt{7+\sqrt{17}}}{2} \approx 0.332433987$
$\delta_2 = 4 - \frac{\sqrt{79-7\sqrt{17}}}{2} \approx 0.459581218$

$\eta_{i,j} =$

	0	1	2
0	0	$6 - \delta$	$12 - 2\delta$
1	$3 - \delta_1$	$5 - \delta$	$11 - 2\delta$
2	$6 - \delta_2$	$4 - \delta$	$10 - 2\delta$
3	$9 - 2\delta$	$3 - \delta$	$9 - 2\delta$
4	$10 - 2\delta$	$4 - \delta$	$6 - \delta_2$
5	$11 - 2\delta$	$5 - \delta$	$3 - \delta_1$
6	$12 - 2\delta$	$6 - \delta$	0

The analytic methods used to find the above constants do not easily generalize and so we utilize approximation methods to determine the values of the constants for larger values of n. It is worth noting that Bartal et. al. provided an algorithm for the $(6, 3)$-server problem in [5] with competitiveness $\frac{155}{78} \approx 1.9871795$ which

is better than the result shown here. However, by using larger values of n we achieve a better upper bound on the competitiveness of the 2-server problem.

4.2 The Program

For $n > 3$, we approximate the value of C numerically, using a program to find a solution to $\mathbb{S}'$. Our program computes a function f, where $\delta = f(\epsilon_{\lfloor n/2 \rfloor})$. To find the maximum value of δ, we assumed that f is bimodal,[1] that is, there is some $x^* > 0$ for which $f(x)$ is maximum, and that $f(x)$ is monotone increasing for $0 < x < x^*$ and monotone decreasing for $x > x^*$. We then use a divide and conquer algorithm similar to binary search to find $f(x^*)$.

1. Guess $\epsilon_{\lfloor n/2 \rfloor}$, using our search algorithm.
2. If n is odd, then solve the following equation for $\epsilon_{(n+1)/2}$:

$$(n + 1 + \epsilon_{(n-1)/2})(2 - \epsilon_{(n+1)/2} + \epsilon_{(n-1)/2})$$

3. For all $0 < i < \lfloor \frac{n}{2} \rfloor$ in decreasing order:
   (a) Solve the following equation for ϵ_i:

$$(2(i + 1) + \epsilon_{n-i-1})(2 - \epsilon_{i+1} + \epsilon_i) = 4(i + 1)$$

   (b) Solve the following equation for ϵ_{n-i}:

$$2(n - i) + \epsilon_i 2 - \epsilon_{n-i} + \epsilon_{n-i-1} = 4(n - i)$$

4. Solve the following equation for δ:

$$(2 + \epsilon_{n-1})(2 - \epsilon_1 - 2\delta) = 4$$

5. Verify the following inequality:

$$(2n - 2\delta)(2 + \epsilon_{n-1}) \geq 4n$$

6. If our search interval is small enough, proceed to the last step. Otherwise, return to step 1.

7. $C \leftarrow \frac{2n-\delta}{n}$.

Our calculations show that $C \approx 1.90098671$ for $n = 2000$. Running the program for larger n leads us to believe that

$$\lim_{n\to\infty} C \approx 1.9007617$$

[1] However, the validity of the program does not depend on the bimodality of f.

5 Future Work

There are two possible directions in which to improve the results in this paper. We conjecture that the set of possible distributions can be expanded in order to obtain an even lower competitiveness. But the real gist of our work is to get a "better than 2" result for general spaces. Since our R–LINE is based on a potential defined in terms of isolation indices it is natural that a generalization to arbitrary metric spaces could make use of T-theory, where a more complex potential will be utilized.

Though not claimed in this paper, our preliminary investigation indicates that R–LINE – unlike the Bartal *et al.* algorithm – generalizes to trees.

Furthermore, it is known that Theorem 1 does not extend to $k \geq 3$ for all metric spaces. However, it does extend to $k \geq 3$ for the line, and should extend easily to the circle.

References

1. Bandelt, H.-J., Dress, A.: A canonical decomposition theory for metrics on a finite set. Adv. Math. 92, 47–105 (1992)
2. Bansal, N., Buchbinder, N., Madry, A., Naor, J.: A polylogarithmic-competitive algorithm for the k-server problem. In: Proc. 52nd Symp. Foundations of Computer Science (FOCS), 10 pages. IEEE Computer Society (2011)
3. Barron, E.N.: Game Theory: An Introduction. John Wiley & Sons, New Jersey (2008)
4. Bartal, Y., Bollobás, B., Mendel, M.: A Ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In: Proc. 42nd Symp. Foundations of Computer Science (FOCS), pp. 396–405. IEEE (2001)
5. Bartal, Y., Chrobak, M., Larmore, L.L.: A randomized algorithm for two servers on the line (Extended Abstract). In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 247–258. Springer, Heidelberg (1998)
6. Bein, W., Chrobak, M., Larmore, L.L.: The 3-server problem in the plane. Theoret. Comput. Sci. 287, 387–391 (2002)
7. Bein, W., Iwama, K., Kawahara, J.: Randomized competitive analysis for two-server problems. Algorithms 1, 30–42 (2008)
8. Bein, W., Iwama, K., Kawahara, J., Larmore, L.L., Oravec, J.A.: A randomized algorithm for two servers in cross polytope spaces. Theoretical Computer Science 412(2), 563–572 (2011)
9. Bein, W., Larmore, L., Noga, J., Reischuk, R.: Knowledge state algorithms. Algorithmica 60(3), 653–678 (2011)
10. Chrobak, M., Larmore, L.L.: An optimal online algorithm for k servers on trees. SIAM J. Comput. 20, 144–148 (1991)
11. Chrobak, M., Larmore, L.L., Lund, C., Reingold, N.: A better lower bound on the competitive ratio of the randomized 2-server problem. Inform. Process. Lett. 63, 79–83 (1997)
12. Koutsoupias, E., Papadimitriou, C.: Beyond competitive analysis. In: Proc. 35th Symp. Foundations of Computer Science (FOCS), pp. 394–400. IEEE (1994)
13. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for server problems. J. Algorithms 11, 208–230 (1990)

Black and White Bin Packing

János Balogh[1,⋆], József Békési[1,⋆], Gyorgy Dosa[2,⋆⋆], Hans Kellerer[3,⋆], and Zsolt Tuza[4,⋆⋆⋆]

[1] Department of Applied Informatics, Gyula Juhász Faculty of Education, University of Szeged, H-6701 Szeged, P.O.B. 396, Hungary
{balogh,bekesi}@jgypk.u-szeged.hu
[2] Department of Mathematics, University of Pannonia, H-8200 Veszprém, Egyetem u. 10, Hungary
dosagy@almos.vein.hu
[3] Institut für Statistik und Operations Research, Universität Graz, Universitätsstraße 15, 8010 Graz, Austria
hans.kellerer@uni-graz.at
[4] Alfréd Rényi Institute of Mathematics, Hungarian Academy of Sciences, H-1053 Budapest, Reáltanoda u. 13–15; and Department of Computer Science and Systems Technology, University of Pannonia, H-8200 Veszprém, Egyetem u. 10, Hungary
tuza@dcs.vein.hu

Abstract. We introduce the following version of bin packing. The items are of two types (black and white), and in each bin the item types must alternate. We mostly investigate the online scenario. We study the competitiveness of some classical algorithms (First/Best/Worst/Next Fit, Harmonic) — they do not perform very well — and for all online algorithms we also prove the universal *lower* bound $1+\frac{1}{2\ln 2} \approx 1.7213$ which significantly exceeds the known *upper* bound 1.58889 on classical online bin packing. We also design an online algorithm which is 3-competitive in the absolute sense. A 2.5-approximation algorithm and an APTAS is also given for the offline version.

1 Introduction

We deal with a new problem, that we call *Black and White Bin Packing*, abbreviated as *B&W-BP*. In the usual sense of bin packing there are items characterized by their sizes $p_1, p_2, \ldots, p_n$, and the goal is to pack them into the minimum number of unit capacity bins. This well-known problem is NP-hard (see [12]), and there are lots of results in this field, we will list some classical ones of them later.

In our problem the items are divided into two classes, they are either black or white. Beside the capacity constraint, we require that no two items of the same color can be packed into a bin right after each other.

⋆ Research supported in part by Stiftung Aktion Österreich-Ungarn, project No. 82öu9.
⋆⋆ Research supported in part by the project K-TET: 10-1-2011-0115.
⋆⋆⋆ Research supported in part by the Hungarian Scientific Research Fund, grant OTKA 81493.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 131–144, 2013.

There are three possible ways (at least) to look at the problem, that can clearly be distinguished from each other, as follows:

1. Offline: In this setting the entire set of items is known from the very beginning, and "black and white bin packing" is equivalent to the assumption that the numbers of the black and white items being packed into the same bin can differ from each other by at most one.

2. Online: The items arrive one by one according to a list L, and no information is given in advance; the next item can be packed only into a bin where it fits and the last item already packed into that bin has the opposite color. If there is no such bin, the new item must be packed into a new bin.

3. Restricted Offline: We also deal with a third model, which we call "restricted offline" (terminology taken from [2]). In this case the items still are given in a list L, and they have to be packed sequentially according to this list, but the order of items and also their sizes are known in advance.

Some Motivation. An application of black and white bin packing can be the optimized distribution of TV or radio programs and their commercial breaks, or music and other kind of program contents on a radio channel, mainly at online radios. The bins correspond to the blocks of the program (at many stations they are one-hour intervals) and the black and white items represent the two kinds of contents, item sizes meaning program duration. The model also makes it possible to optimize similar online contents (as an example, information and advertisement alternately, like on many video-content sharing portals, e.g. Youtube) onto today's mobile phone devices (for example onto "smartphones"). On mobile phones the contents often are observed in a band-like arrangement because on the tiny display they fit under each other only. Here bin size means the maximum amount of information fitting on one screen.

The problem seems to be interesting also in the sense that deleting some items from the sequence, the value of the *optimum can increase* (in case of all the three subproblems). This cannot happen in the pure bin packing problems. To illustrate the difference, let us consider a long sequence where very small black and white items alternate. These items can be packed into one bin, but deleting the white items, a feasible packing needs many bins.

Let the value of the optimal solution in the offline or restricted offline case be denoted simply as OPT. For any set S of items let $P(S)$ denote the total size of items in the set S.

Lower Bound LB_0. It is obvious that the total size of the items is a lower bound for any of the three (offline, restricted offline, online) versions of the problem. Let this lower bound be denoted as LB_0. Thus, for any input L, the value $LB_0(L) = P(L)$ is a valid lower bound. In each of the three models, LB_0 is just a sum; the assumption that L is an ordered list has its relevance only concerning the way of packing the items under the online and restricted offline scenarios.

At any moment of the packing procedure, we call a bin with black item on the top as *black bin*, and analogously *white bin* if the item on the top is white.

As usual, the *level* of a bin means the sum of the sizes of items already packed into the bin.

To evaluate the efficiency of an *online* algorithm, *competitive ratio* is one of the standard measures. If an online algorithm always achieves a solution within a factor ρ of the offline optimum, we say that the online algorithm is ρ-*competitive*. More explicitly, for any input list L and an online algorithm A, let $OPT(L)$ and $A(L)$ denote, respectively, the number of bins used by an optimal offline algorithm and the number of bins used by algorithm A to pack the list L. Then the *absolute competitive ratio* of A is defined as

$$R_{A,abs} = \sup_L \{A(L)/OPT(L)\},$$

while

$$R_{A,as} = \lim_{n\to\infty} \sup_L \{A(L)/OPT(L) \mid OPT(L) \geq n\},$$

is called the *asymptotic competitive ratio* of A. In the *offline* setting the analogous measures are called *approximation ratio*. For both settings together we simply use the term *performance ratio*.

It is worth noting here that in case of the restricted offline or online model, the efficiency of any algorithm should be compared to the *restricted offline* optimum. Otherwise no algorithm can be constant-competitive, since keeping the order of the items through the packing is a very strong condition. (If n small white items come first and then n small black items follow them, this list needs n bins in the restricted offline or online model, while all items may fit into one bin in the offline case.)

Some Classical Algorithms and Results for the Bin Packing Problem

In his seminal PhD work [14], David Johnson defined several classical (online) algorithms, as follows. The Next Fit algorithm keeps only one open bin at any time. If the next item cannot be packed into the open bin, the bin gets closed and the actual item is packed into a newly opened bin. With a somewhat different approach where opened bins do not get closed, in case of the First Fit, Best Fit, or Worst Fit algorithms, the next item is always packed into the first bin where it fits, into a bin with highest level where it fits, or into a bin with lowest level where it fits, respectively; and if there is no such bin, the item is packed into a new bin. The generalization of the latter three "Fit" algorithms is called Any Fit; this algorithm is allowed to pack the new item into any open bin where it fits, and if there is no such bin, the item is packed into a new bin. The algorithms are abbreviated as NF, FF, BF, WF, and AF, respectively. Then FF, BF and WF are restricted versions of AF. Moreover, FFD (abbreviation for First-Fit Decreasing) is the ordered version of FF where first the items are put into nonincreasing order and then FF is applied for the ordered list.

A totally different packing idea is used by algorithm $Harmonic(K)$ [19]. The items here are classified according to their sizes, items from interval $(\frac{1}{i+1}, \frac{1}{i}]$ belong to *Class* i for $1 \leq i \leq K-1$, where $K \geq 2$ is a fixed integer. The smaller

items, i.e. items with sizes at most $\frac{1}{K}$ belong to *Class* K. Then through the packing process in any specific bin only items from the same class are packed. If an incoming item fits into a bin of the class of the item then the item is packed there, otherwise a new bin is opened for this class of items.

The asymptotic performance ratio of $Harmonic(K)$ is approximately 1.69103 if K is chosen sufficiently big, and the performance ratio improves by increasing K.

Regarding the "Fit" algorithms, FF has asymptotic performance ratio 1.7, and parametric performance ratio $1 + 1/d$ for any parameter value of $d \geq 2$, see [17]. Recently two papers [26,4] proved independently that for any instance L, $FF(L)/OPT(L) \leq 12/7 \approx 1.7143$ holds. Thus the asymptotic ratio 1.7 is tight for algorithm FF, but regarding the absolute performance bound the question is still open (the lower bound is 1.7, the best upper bound is 12/7). For the ordered version of FF called FFD, where first the items are put into a nonincreasing list, the tight asymptotic ratio is 11/9 (see [5]), and the tight value of the additive constant is found by Dosa in [6] as $FFD(L) \leq \frac{11}{9}OPT(L) + 6/9$.

For off-line algorithms, Fernandez de la Vega and Lueker [20] provided an APTAS (Asymptotic Polynomial-Time Approximation Scheme), while Karmarkar and Karp [18] developed the first AFPTAS (Asymptotic Fully Polynomial-Time Approximation Scheme). In [20] for any $\varepsilon > 0$ an algorithm A_ε is given such that each A_ε runs in polynomial time in the length of the input list L (although exponential in $1/\varepsilon$) and has $A_\varepsilon(L) \leq (1 + \varepsilon)OPT(L) + 1$. In [18], a more complex algorithm is given. The running time of this algorithm A depends on n and $1/\varepsilon$ polynomially and $A(L) \leq OPT(L) + O(\log^2 OPT(L))$ holds for this algorithm.

Regarding classical online bin packing, the current champion algorithm is Harmonic++ by Seiden [24], which has an asymptotic competitive ratio of 1.58889. The best lower bound was 1.5401 [25] for a long time, while recently Balogh et al. improved it to $\frac{248}{161} \approx 1.5403$ [1].

Special Models Related to B&W-BP : LIB Constraint, Bin Packing with Conflicts, Graph-Bin Packing

There are many special versions of bin packing from the early seventies, for details see for example [5]. We review here only three special problems. The *bin packing problem with LIB constraint* (Largest Item in Bottom) was introduced by Manyem [21,22]. In this model the items still arrive according to a list, but an item with bigger size is not allowed to be packed into a bin that already contains an item with smaller size. In the cited papers it is shown that NF is not constant competitive, but the competitive ratio of First Fit (FF) is at most 3. Note that, when we deal with this LIB constraint, the solution of the online algorithm is always compared to a restricted kind of offline optimum, namely where the offline algorithm must pack the items in the order of their given list, otherwise in case of the offline solution the LIB constraint could be totally neglected.

Epstein [9] gave an improved analysis of FF for the problem so that its competitive ratio is at most 2.5, and proved that the parametric competitive ratio

(where the size of each item is at most $1/d$) is at most $2 + 1/d$, for any integer $d \geq 2$. In the same paper, it was shown further that the competitive ratio of any online algorithm for bin packing with LIB constraints is at least 2, for any value of the parameter d.

Later, Dosa et al. [7] proved that the (absolute) competitive ratio of algorithm FF is not worse than 2+1/6 for the problem; moreover, the (absolute) parametric competitive ratio of FF is at most $2 + 1/d(d + 2)$. In the same paper a problem with the generalized LIB constraint is also defined and analysed.

Another version of the classical bin packing problem, called *bin packing with conflicts*, is where items are regarded as nodes of a graph, and there is an edge between two nodes if the corresponding two items cannot be packed together into the same bin. For general graphs the optimum is hard to approximate, due to the fact that the chromatic number of a graph cannot be approximated within $n^{1-\epsilon}$ for any $\epsilon > 0$, where n is the number of nodes (or items) [13,27]. Many papers investigate this problem on some special conflict graphs. For perfect graphs, Jansen and Öhring [16] proved that First Fit has competitive ratio at most 2.7; more recently Epstein and Levin [10] improved the upper bound to 2.5. For other classes of graphs, we refer to the following papers for details: clique graphs [23], bipartite graphs [10,16], interval graphs [10], cographs and partial K-trees [16], d-inductive graphs [15]; multi-dimensional bin packing with conflicts [11].

In [3] a very general problem, named *graph-bin packing* is introduced, which is a common generalization of many fundamental problems studied to a great extent separately in thousands of papers in the literature of combinatorial optimization, graph theory, and operations research. The paper [3] treats many subproblems, and proposes efficient approximation and online algorithms.

2 Results on Black and White Bin Packing

We study the problem from various aspects. Our main goal is to prove lower and upper bounds on the performance ratio of algorithms. Below we list the results in a structured way; due to space limitations, most of the proofs had to be omitted, they will appear elsewhere. Although we include some proofs in later sections, we mostly try to give a general picture about the status of the problem.

2.1 Offline Approximation Algorithms

We currently do not know how efficiently the problem can be approximated. Nevertheless, the following upper bound can be proved.

Theorem 1. *There is a* 2.5*-approximation algorithm for offline Black and White Bin Packing, which runs in* $O(n \log n)$ *time.*

The main issue here is the explicit bound on running time; actually, our algorithm is almost linear, in the sense that the '$\log n$' factor occurs just because a sorting subroutine has to be applied at some points. Moreover, we expect that 2.5 can be improved to 2 as a valid upper bound on competitiveness.

In the asymptotic sense, a much stronger approach can be carried out:

Theorem 2. *There exists an asymptotic polynomial time approximation scheme for offline Black and White Bin Packing.*

2.2 Lower Bounds for Online and Restricted Offline Algorithms

There is a natural way to define the appropriate versions of NF, FF, BF, and WF algorithms for Black and White Bin Packing, too. They work in the same way as in the pure online case, the next item must be packed into a new bin if it is not allowed to be packed into any open bin by the packing rule of the algorithm. The only difference is that now an item is allowed to be packed into some bin only if it fits and also its color is not the same as that of the top item in the bin.

On the negative side, we observe that neither of the algorithms mentioned above can work well. The situation is worst in case of Next Fit:

Proposition 1. *NF is not constant-competitive, neither in the online, nor in the restricted offline case.*

Concerning the other 'Fit' algorithms, lists forcing competitive ratio at least 2 are not hard to design, but this bound is not tight. After a sequence of unpublished improvements, Leah Epstein [8] constructed problem instances which push the lower bound up to 3. We include this result by her kind permission.

Theorem 3. *The asymptotic competitive ratios of algorithms FF, BF, and WF are at least* 3*. Moreover, in the parametric case, if all items are at most* $1/d$*, the parametric asymptotic competitive ratio of algorithm WF is at least* $1 + \frac{d}{d-1}$*.*

Regarding the $Harmonic(K)$ algorithm, we can see very easily that it works even more poorly. Let $K \geq 2$ be fixed, then there are at least two classes. In the wrong list, let white and black items arrive alternatingly, from the two smallest classes. More exactly, for $i = 1, \ldots, (K-1)n$ let

- $A_i = \frac{1}{K-1} - \varepsilon$, a white item, and
- $B_i = \varepsilon$, a black item,

where n is a big integer, and $\frac{1}{K-1} - \varepsilon > \frac{1}{K}$. Then obviously in the optimum packing the items can be packed into n bins, while $Harmonic(K)$ packs each item into dedicated bins, thus $Harmonic(K) = (K-1)n$. It follows that the asymptotic performance ratio of $Harmonic(K)$ is at least $K-1$, thus it becomes weaker and weaker as K grows.

We conjecture that the parametric competitive ratio of the algorithms FF, BF, and WF is really 3, 3, and $1 + \frac{d}{d-1}$, respectively, but the analysis seems to be hard.

Concerning lower bounds, the most general result of our paper is:

Theorem 4. *There is no online algorithm for Black and White Bin Packing with asymptotic competitive ratio smaller than* $1 + \frac{1}{2 \ln 2} \approx 1.7213$*.*

For the construction we use a special list of very small items and some other lists concatenated with it. Formally the lists look like this. Let $a = \frac{1}{k}$, where k is fixed, (big) number, which is dividable by 4, and let $y_i = \frac{i}{k}$ for $0 < i \leq k$. Now let n be a positive integer. Define the first list, L_0 as a list of kn items of size $a = \frac{1}{k}$. Thus L_0 contains very small elements with equal, suitably chosen size. The cumulative size of the small items is n. The colors of the items are alternating, i.e. the the items with odd indexes are white, while the items with even indexes are black. Thus the first small item is white.

It is easy to see that $OPT(L_0) = n$. It is also easy to see that n can be defined so that $\frac{n}{y_i}$ is integer for any i. Thus let n be such (sufficiently big) integer number.

Define L_{iw} as a list of $n_i = \frac{n}{y_i}$ white items of size $0 < 1 - y_i \leq 1/2$. We choose i here to satisfy that i is even. The sizes of the elements are equal in list L_{iw}. Similarly define L_{ib} as a list of $n_i = \frac{n}{y_i}$ black items of size $1 - y_i$. The proof of the lower bound is based on the analysis of the behavior of an arbitrary online algorithm for the next lists:

- L_0,
- $L_0 L_{iw}$,
- $L_0 L_{ib}$

An important aspect of this lower bound occurs in comparison to the known 1.58889-competitive algorithm (that we mentioned earlier) on classical online bin packing. It follows that B&W-BP is conceptually harder.

2.3 Lower Bounds for Optima of Restricted Online Instances

The lower bound LB_0 is computed from item sizes only. Here we present two further bounds, one of them determined by the color pattern of list L, while the other one takes both the color patterns and the item sizes into account.

Lower bound LB_1. Suppose that $L = p_1, p_2, \ldots, p_n$ is the list of the items (which is not known in advance in the online case). Let $s_i = 1$ if the ith item is black and let $s_i = -1$ in the opposite case, if the ith item is white. We define

$$LB_1 := \max_{1 \leq i < j \leq n} \left| \sum_{k=i}^{j} s_k \right|$$

The following assertion can be proved.

Proposition 2. *For any problem instance, LB_1 is a lower bound on the optimum, both in the online and restricted offline cases. Moreover, if all items have zero sizes (or the bins have infinite capacity), then LB_1 is equal to the restricted offline optimum, and also to the online optimum. In these cases $AF = LB_1$ holds, i.e. algorithm Any Fit attains an optimal solution.*

For any list $L = \{p_1, \ldots, p_n\}$ of items, an instance of the problem, we denote by $L = L_B \cup L_W$ the partition into the sets of black and white items.

Definition 1. *The conflict graph of black items in an instance of the online or restricted offline problem is an undirected graph, denoted by G_B. It has vertex set L_B. Two items $p_i, p_j \in L_B$ with $1 \leq i < j \leq n$ are joined by an edge if and only if, for every white item p_k in the range $i < k < j$ we have $p_i + p_j + p_k > 1$. The conflict graph of white items, denoted by G_W, is defined analogously on the vertex set L_W.*

For any graph G, we use the standard notation $\omega(G)$ for the *clique number* (largest number of mutually adjacent vertices) and $\chi(G)$ for the *chromatic number* (smallest number of independent sets into which the vertex set can be partitioned). Since no two vertices adjacent by an edge of G_B or G_W can be packed into the same bin, it is immediate by definition that

$$opt(L) \geq \max\{\chi(G_B), \chi(G_W)\} \geq \max\{\omega(G_B), \omega(G_W)\} =: LB_2 \qquad (1)$$

holds for all instances $L = L_B \cup L_W$.

Theorem 5. *For any instance of Black and White Bin Packing, we have $\chi(G_B) = \omega(G_B)$ and $\chi(G_W) = \omega(G_W)$. Moreover, the lower bound given in (1) is computable in $O(n^2)$ time, where n denotes the number of items.*

An interesting aspect of the time bound $O(n^2)$ is that already checking the adjacency of a single vertex pair in the conflict graph G_B or G_W may take $\Omega(n)$ time.

The following problem may be of interest on its own right.

Problem 1. Characterize the structure of graphs that can occur as conflict graphs of black (or white) items.

2.4 An Efficient Online Algorithm

Our main positive result for the online problem is the following one.

Theorem 6. *There exists an online algorithm which is 3-competitive in the absolute sense, and also $(1 + \frac{d}{d-1})$-competitive in the parametric version where all items have sizes at most $1/d$.*

We describe the 3-competitive algorithm in the next section. It should be noted that the parametric upper bound $(1 + \frac{d}{d-1})$ approaches 2 from above as $d \to \infty$, while we have seen that FF can never be better than 3-competitive, no matter that the items are small or big.

Remark 1. Taking into account the online algorithm guaranteed by Theorem 6 and the general lower bound of Theorem 4, we see that the tight absolute competitive ratio for Black and White Bin Packing is between 1.7213 and 3.

3 Some Proofs and Constructions for B&W-BP

Space limitation does not allow us to present too much of the proofs. We have selected a relatively short argument verifying one online lower bound and the construction of the APTAS without proof of the theorem.

3.1 A 3-Competitive Online Algorithm

Here we describe an online algorithm, which actually satisfies all requirements of Theorem 6. Due to space limitation, however, we only prove now that the algorithm has competitive ratio at most 3, and omit the analysis for parametric absolute competitiveness.

Algorithm Pseudo

1. First we substitute each item in the sequence with a "pseudo" item, with the same color but with size zero. It does not matter that we do not know the items in advance, this substitution can be done at the moments when the items are revealed. Then, as the items come, by using algorithm AF we pack the pseudo items optimally, as claimed in a part of Proposition 2.
2. When the sequence stops, we divide the content of each overloaded bin (i.e. bin with level more than one) into subsequent bins as follows. We keep the order of the subsequent items. As soon as the level of a bin would exceed 1, we start to pack a new bin.

We prove that the competitive ratio of the algorithm is 3. Let LB' be the strongest lower bound obtained from LB_1. The first part of the algorithm finds a packing into LB' bins, as guaranteed in the corresponding part of Proposition 2. Instead of shrinking the items to size zero, we may equivalently view this step so as to increase bin capacities to infinite. Then each bin in the sequence of incoming items is split into bins packed up to at most 1; in fact this splitting can be done online.

Suppose that bin i is split into exactly m_i small bins. Then we have a packing of value $\sum_{i=1}^{LB'} m_i$. On the other hand, in each infinite bin, any two consecutive small bins have total load more than 1. Hence, bin i contains items of total size more than $\lfloor m_i/2 \rfloor$. Note that $\lfloor m_i/2 \rfloor = m_i/2$ if $m_i = 2$ and $\lfloor m_i/2 \rfloor \geq m_i/2 - 1/2$ if $m_i \geq 3$. Let S_1, S_2 and S_3 be the sets of bins for which $m_i = 1$, $m_i = 2$ or $m_i \geq 3$, respectively. Let us denote $L_k = |S_k|$, for $k = 1, 2, 3$. Then $L_1 + L_2 + L_3 = LB'$. Also, let $M = \sum_{i=1, i \in S_3}^{LB'} m_i$. Then $M \geq 3L_3$. We write $M = 3L_3 + 2x$, where $x \geq 0$.

Then $OPT \geq LB'$ holds, and further

$$OPT \geq \sum_{i=1}^{LB'} \lfloor m_i/2 \rfloor \geq \sum_{i=1, i \in S_2}^{LB'} m_i/2 + \sum_{i=1, i \in S_3}^{LB'} m_i/2 - L_3/2 = L_2 + M/2 - L_3/2.$$

Thus, the competitiveness of the solution is better than

$$\begin{aligned}\frac{\sum_{i=1}^{LB'} m_i}{\max(LB', L_2 + M/2 - L_3/2)} &= \frac{L_1 + 2L_2 + M}{\max(L_1 + L_2 + L_3, L_2 + M/2 - L_3/2)} \\ &= \frac{L_1 + 2L_2 + 3L_3 + 2x}{\max(L_1 + L_2 + L_3, L_2 + L_3 + x)} \\ &\leq \frac{L_1 + 2x + 3(L_2 + L_3)}{\max(L_1, x) + L_2 + L_3} \leq 3\end{aligned}$$

Consequently, the competitive ratio of the algorithm is not worse than 3 (in the absolute sense).

In fact, it can be shown [8] that the analysis of the algorithm is tight, i.e. the absolute, and even the asymptotic competitive ratio of the algorithm is equal to 3. The sequence verifying this claim consists of $3N$ items for a large integer N. For $i = 1, \ldots, N$, item $3i - 2$ is white and items $3i - 1$ and $3i$ are black (as W,B,B,W,B,B,...). Thus Pseudo creates one bin with items $1, 2, 4, 5, 7, 8, \ldots$ (items $3i - 2$ and $3i - 1$ for all i are in this bin), and N bins with items $3, 6, \ldots$ (items $3i$ are in dedicated bins). Now the sizes of items $3i - 2$ and $3i$ are $1/(2N)$ and the size of item $3i - 1$ is $1 - 1/(2N) + 1/N^2$ (i.e. the middle item in any triplet is big and the two other items are small). As a result, when the first bin of Pseudo is split into valid bins, the algorithm packs every item in a separate bin. So $Pseudo = 3N$. An optimal offline solution, however, packs all items of size $1/(2N)$ into one bin, and every larger item into a separate bin. Thus $OPT = N + 1$. □

3.2 An Asymptotic PTAS for Offline Black and White Packing

In this section we will present an APTAS for offline B&W Packing. In fact for a given accuracy ε, we will find a solution which uses at most $(1 + \varepsilon)OPT + 1$ bins with running time polynomial in n. Set $\delta := \varepsilon/6$. The items with size less than or equal to δ are called *small*, the items greater than δ are the *large* items. The large black items are denoted by B_L and the small black items by B_S, respectively. For the large and small white items we use W_L and W_S. Hence, we get $B = B_L \cup B_S$ and $W = W_L \cup W_S$. Assume that both B and W are sorted in decreasing order.

Let m be the number of bins used by algorithm Master, which is an offline algorithm with a 2.5 approximation ratio. By Theorem 1 we have

$$\frac{m}{2.5} \leq OPT \leq m. \tag{2}$$

Hence, we may assume w.l.o.g. that $\varepsilon < 1.5$. Set

$$\alpha_B := \min\{|W_L| + m, |B_S|\}. \tag{3}$$

We will form the sets B_λ, $0 \leq \lambda \leq \alpha_B$, from B in the following way.

B_λ consists of the λ smallest items from B_S and all items from B_L. From B_λ a new instance $B_\lambda^{(1)}$, with only a *constant* number of different sizes, is introduced. The basic idea for this is the *shifting technique*, an idea that is used in approximation algorithms for classical bin packing [20]. Note that using the shifting technique for classical bin packing only the "large" items are split into a finite number of classes. This is not possible for B&W packing which makes the algorithm and its analysis much more complicated.

Set

$$k := \left\lceil \frac{39}{\varepsilon^2} \right\rceil$$

and

$$h_\lambda^B := \left\lfloor \frac{|B_\lambda|}{k} \right\rfloor. \tag{4}$$

If $h_\lambda^B = 0$, simply set $B_\lambda^{(1)} = B_\lambda$ and B_λ contains only a constant number of items which can be packed optimally in polynomial time by complete enumeration. Thus, assume in the following that $h_\lambda^B > 0$.

Denote by $q_1, \ldots, q_r$ the items in B_λ and assume that they are sorted in decreasing order of sizes, i.e. $q_1 \geq q_2 \geq \ldots \geq q_r$. Partition B_λ into the $k+1$ groups

$$G_j := \{q_{jh_\lambda^B+1}, \ldots, q_{(j+1)h_\lambda^B}\}, \quad j = 0, \ldots, k-1,$$

containing h_λ^B items each and $G_k := \{q_{kh_\lambda^B+1}, \ldots, q_r\}$ containing $\tilde{h}_\lambda^B \leq h_\lambda^B$ items.

Define the set $B_\lambda^{(1)}$ by discarding the first group G_0 and replacing it by h_λ^B *dummy items* of size zero, and for each other group form a new group by rounding the size of its items up to the size of its largest item. Formally, $B_\lambda^{(1)}$ is a list consisting of h_λ^B items of size $q_{jh_\lambda^B+1}$, $j = 1, \ldots, k-1$, $\tilde{h}_\lambda^B$ items of size $q_{kh_\lambda^B+1}$ and h_λ^B items of size zero. Denote by $s_1, \ldots, s_r$ the items in $B_\lambda^{(1)}$ and assume that they are sorted in decreasing order of sizes, i.e. $s_1 \geq s_2 \geq \ldots \geq s_r$. From the construction of $B_\lambda^{(1)}$ follows that

$$s_j \leq q_j, \quad j = 1, \ldots, r. \tag{5}$$

An analogous construction (with corresponding notations) is done for the white items to obtain instances W_μ, $W_\mu^{(1)}$, $0 \leq \mu \leq \alpha^W$, with $\alpha^W = \min\{|B_L| + m, |W_S|\}$. Thus, we get instances $I_{\lambda,\mu}$, $I_{\lambda,\mu}^{(1)}$ with

$$I_{\lambda,\mu} = B_\lambda \cup W_\mu,$$

and

$$I_{\lambda,\mu}^{(1)} = B_\lambda^{(1)} \cup W_\mu^{(1)}.$$

Note that $I_{\lambda,\mu} \subseteq I_{\lambda',\mu'}$ and $I_{\lambda,\mu}^{(1)} \subseteq I_{\lambda',\mu'}^{(1)}$ for $\lambda \leq \lambda', \mu \subseteq \mu'$. Moreover, define $I^{(1)} := I_{\alpha_B,\alpha_W}^{(1)}$ as the maximal set among the sets $I_{\lambda,\mu}^{(1)}$.

The set of black and white dummy items in a set $I^{(1)}_{\lambda,\mu}$ shall be denoted by $D_{\lambda,\mu}$. Note that

$$|D_{\lambda,\mu}| = h^B_\lambda + h^W_\mu \leq h^B_{\alpha_B} + h^B_{\alpha_W},$$

and we abbreviate the maximum set D_{α_B,α_W} by D. In the following, we will distinguish between large, small and dummy items.

We say that a set of items L *dominates* a set of items L' if there is a bijective mapping ϕ from L to L' such $x \geq \phi(x)$ and x and $f(x)$ have the same color for all $x \in L$. Obviously, for any feasible packing A of L there is a *corresponding feasible packing* $\phi(A)$ of L' by replacing every item x by $\phi(x)$. Especially, $OPT(L') \leq OPT(L)$ holds. We conclude from (5) that $I_{\lambda,\mu}$ dominates $I^{(1)}_{\lambda,\mu}$ for arbitrary values of λ and μ. Note that in contrast to classical bin packing $OPT(L') \leq OPT(L)$ does not hold in B&W Packing for $L' \subset L$.

The *color sum* $C(L)$ of a set L of items is the number of black items in L minus the number of white items. The *color sum of a set of bins* corresponds to the color sum of the items packed in the bins. Hence, a set of items L with total size not greater than 1 can be packed in a bin if and only if $C(L) \in \{-1, 0, 1\}$. We say that two items have the same *item type* or call them *identical* if they have the same size and the same color. The number of different item types in $I^{(1)}$, and thus all sets $I^{(1)}_{\lambda,\mu}$, is bounded by $2(k+1)$ and thus a constant.

A set of items of $L \subseteq I^{(1)}$ forms a *feasible packing configuration* if the following conditions hold

(i) L contains at most $\lceil 1/\delta \rceil$ small or dummy items.
(ii) $P(L) \leq 1$, i.e., the items in L fit in a bin.
(iii) $C(L) \in \{-1, 0, 1\}$.

By definition every feasible packing configuration can be packed in a bin. Moreover, there is a one-to-one correspondence between feasible packing configurations and feasible packings of items of $I^{(1)}$ in a bin.

Let f denote the number of different feasible packing configurations. By a rough estimate f can be bounded by

$$f \leq \left\lceil \frac{1}{\delta} \right\rceil^{2(k+1)},$$

and hence by a constant. This holds due to property (i) and since at most $\lceil 1/\delta \rceil - 1$ large items can be packed in a bin.

Formally, our asymptotic polynomial time approximation scheme reads as follows:

Algorithm A_ε

1. For a given instance I of items and a given accuracy ε calculate the sets B_L, W_L, B_S, W_S.
2. Construct all possible sets $I^{(1)}_{\lambda,\mu}$, $0 \leq \lambda \leq \alpha_B, 0 \leq \mu \leq \alpha_W$.
3. For every set $I^{(1)}_{\lambda,\mu}$ find all possible feasible packings in at most m bins.
4. Every packing is converted into a solution for the original instance I in the following way.

4.1 For each group of $B_\lambda^{(1)}$ and $W_\mu^{(1)}$ replace the ν items of the same type by the ν largest items of the corresponding group in B_λ and W_μ, respectively.
4.2 Assign the h_λ^B largest items of B_L and the h_μ^W largest items of W_L to separate bins, pairing black and white items if possible.
4.3 If the color sum of the remaining unpacked small items from I is not equal to zero, STOP. Otherwise, these items are grouped arbitrarily into pairs of black and white items. Pack these pairs in the bins using First Fit.
4.4 Remove the dummy items and make the obtained solution feasible by removing a minimum number of items of each bin and packing them separately.

5. Output the solution which uses the smallest number of bins.

4 Conclusion

We have introduced and studied a variant of the bin packing problem, in which the items are of two types and any two items of the same type packed into the same bin have to be separated by one item of the other type. We have investigated the possible effectiveness of algorithms in both the offline and online settings, moreover we have considered a "restricted offline" scenario, too, where the entire problem instance is known in advance but the items are given in a fixed order. Lower bounds have been proved and algorithms have been designed with guaranteed approximation or competitive ratio. It remains for future research to determine the best possible ratio achievable.

Acknowledgments. G. Dosa acknowledges the financial support of the Hungarian State and the European Union under the grant no.: TAMOP-4.2.2.A-11/1/KONV-2012-0072. J. Balogh and J. Békési were supported by the European Union and the European Social Fund through project " Supercomputer, the national virtual lab" grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0010.

References

1. Balogh, J., Békési, J., Galambos, G.: New lower bounds for certain classes of bin packing algorithms. Theoretical Computer Science 440-441, 1–13 (2012)
2. Benko, A., Dosa, G., Tuza, Z.: Bin Covering with a general profit function: approximability results. Central European Journal of Operations Research (2012) (online published)
3. Bujtas, C., Dosa, G., Imreh, C., Nagy-György, J., Tuza, Z.: The Graph-Bin Packing Problem. Int. J. Found. Comput. Sci. 22(8), 1971–1993 (2011)
4. Boyar, J., Dosa, G., Epstein, L.: On the absolute approximation ratio for First Fit and related Results. Discrete Applied Mathematics 160(13-14), 1914–1923 (2012)
5. Coffman Jr, E., Garey, M., Johnson, D.: Approximation algorithms for bin packing: A survey. In: Approximation Algorithms for NP-hard Problems, pp. 46–93. PWS Publishing Co. (1996)

6. Dósa, G.: The tight bound of first fit decreasing bin-packing algorithm is $FFD(I) \leq 11/9OPT(I) + 6/9$. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 1–11. Springer, Heidelberg (2007)
7. Dosa, G., Tuza, Z., Ye, D.: Bin packing with "Largest In Bottom" constraint: Tighter bounds and generalizations. Journal of Combinatorial Optimization (to appear), doi: 10.1007/s10878-011-9408-0
8. Epstein, L.: Personal communication (2011)
9. Epstein, L.: On online bin packing with LIB constraints. Naval Research Logistics 56(8), 780–786 (2009)
10. Epstein, L., Levin, A.: On bin packing with conflicts. In: Erlebach, T., Kaklamanis, C. (eds.) WAOA 2006. LNCS, vol. 4368, pp. 160–173. Springer, Heidelberg (2007)
11. Epstein, L., Levin, A., van Stee, R.: Multi-dimensional packing with conflicts. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 288–299. Springer, Heidelberg (2007)
12. Garey, M.R., Johnson, D.S.: Computer and Intractability: A Guide to the theory of NP-Completeness. Freeman, New York (1979)
13. Håstad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. Acta Mathematica 182, 105–142 (1999)
14. Johnson, D.S.: Near-optimal bin-packing algorithms, Doctoral Thesis, MIT, Cambridge (1973)
15. Jansen, K.: An approximation scheme for bin packing with conflicts. Journal of Combinatorial Optimization 3(4), 363–377 (1999)
16. Jansen, K., Öhring, S.: Approximation algorithms for time constrained scheduling. Information and Computation 132, 85–108 (1997)
17. Johnson, D., Demers, A., Ullman, J., Garey, M., Graham, R.: Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on Computing 3, 299–325 (1974)
18. Karmarkar, N., Karp, R.: An efficient approximation scheme for the one-dimensional bin packing problem. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982), pp. 312–320 (1982)
19. Lee, C.C., Lee, D.T.: A simple on-line packing algorithm. J. ACM 32, 562–572 (1985)
20. de la Vega, W.F., Lueker, G.S.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica 1, 349–355 (1981)
21. Manyem, P.: Uniform sized bin packing and covering: Online version. In: Misra, J.C. (ed.) Topics in Industrial Mathematics, pp. 447–485. Narosa Publishing House, New Delhi (2003)
22. Manyem, P., Salt, R., Visser, M.: Approximation lower bounds in online LIB bin packing and covering. Journal of Automata, Languages and Combinatorics 8(4), 663–674 (2003)
23. McCloskey, B., Shankar, A.: Approaches to bin packing with clique-graph conflicts, Technical Report UCB/CSD-05-1378, EECS Department, University of California, Berkeley (2005)
24. Seiden, S.: On the online bin packing problem. Journal of the ACM (JACM) 49(5), 640–671 (2002)
25. van Vliet, A.: An improved lower bound for on-line bin packing algorithms. Information Processing Letters 43(5), 277–284 (1992)
26. Xia, B.Z., Tan, Z.Y.: Tighter bounds of the First Fit algorithm for the bin-packing problem. Discrete Applied Mathematics 158(15), 1668–1675 (2010)
27. Zuckerman, D.: Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. Theory of Computing 3, 103–128 (2007)

Minimizing Cache Usage in Paging

Alejandro López-Ortiz and Alejandro Salinger

David R. Cheriton School of Computer Science, University of Waterloo,
200 University Ave. West, Waterloo, ON, Canada, N2L3G1
{alopez-o,ajsalinger}@uwaterloo.ca

Abstract. Traditional paging models seek algorithms that maximize their performance while using the maximum amount of cache resources available. However, in many applications this resource is shared or its usage involves a cost. In this work we introduce the Minimum Cache Usage problem, which is an extension to the classic paging problem that accounts for the efficient use of cache resources by paging algorithms. In this problem, the cost of a paging algorithm is a combination of both its number of faults and the amount of cache it uses, where the relative cost of faults and cache usage can vary with the application. We present a simple family of online paging algorithms that adapt to the ratio α between cache and fault costs, achieving competitive ratios that vary with α, and that are between 2 and the cache size k. Furthermore, for sequences with high locality of reference, we show that the competitive ratio is at most 2, and provide evidence of the competitiveness of our algorithms on real world traces. Finally, we show that the offline problem admits a polynomial time algorithm. In doing so, we define a reduction of paging with cache usage to weighted interval scheduling on identical machines.

1 Introduction

The efficient management of a computer memory hierarchy is a fundamental problem in both computer architecture and software design. A program's data and instructions reside in various levels of the hierarchy, in which memories at higher levels have higher capacities, but slower access times. Simplified to a two-level memory system, the paging problem models a slow memory of infinite size and a fast memory of limited size, usually known as the cache. The input consists of a sequence of page requests. If the page of a request is in the cache then the request is a *hit*; otherwise it is a *miss* or *fault* and the requested page must be brought from slow memory to cache, possibly requiring the eviction of one or more pages currently residing in cache. A paging algorithm must decide which pages to maintain in the cache at each time in order to minimize a defined cost measure. In the classic page fault model the cost of an algorithm is measured in terms of its number of faults and hits have no cost, reflecting the fact that an access to slow memory is orders of magnitude slower than an access to cache.

As computer architectures and applications evolve, other cost models have arisen to reflect, for example, varying fetching costs and sizes in web-caches

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 145–158, 2013.

[11,18,29], or multi-threaded applications sharing a cache [4,9,16,17,21]. In this paper we consider a generalization of the classic page fault model whose performance objective function is a combination of both the number of faults and the amount of cache used by an algorithm. Thus in addition to the fault cost, at each step we charge a cost proportional to the number of pages in cache. In general, the model seeks algorithms with good performance in terms of number of faults while at the same time using available resources efficiently. Naturally, minimizing the number of faults and the cache usage of a paging algorithm are conflicting goals.

Paging strategies that minimize cache usage are relevant in multi-core architectures where multiple cores share some level of cache. In this context, multiple request sequences compete for the use of this shared resource. While traditional models of paging encourage algorithms to use the entire cache so as to minimize the faults incurred, a model that charges for cache usage can make a paging algorithm in a shared cache scenario be "context aware". Varying the parameters of the model for each sequence can be used to achieve a cooperative global strategy with better overall performance.

The cache minimizing model can also be used as an energy efficient paging model. Several applications use caches implemented with Content-Addressable Memories (CAMs), most notably networking routers and switches, and Translation Lookaside Buffers (TLBs). CAMs provide a single clock cycle throughput, making them faster than other hardware alternatives [24]. However, speed comes at a cost of increased power consumption, mainly due to the comparison circuitry. Reducing this power without sacrificing capacity or speed is an important goal of research in circuit design [24]. Power consumption could be reduced if inactive cache lines are turned off, thus our model can provide a framework for paging strategies that achieve good performance in terms of faults while contributing to energy savings.

1.1 Paging and Cost Models

The paging problem has been extensively studied; some well-known page replacement policies are Least-Recently-Used (LRU), which evicts the page in the cache whose last access time is furthest in the past; First-In-First-Out (FIFO), which evicts the page that has been longest in the cache; and Flush-When-Full (FWF), which when required to evict a page evicts all pages from the cache.

The performance of paging algorithms has been traditionally measured using competitive analysis [26]. A paging algorithm A has competitive ratio r or is r-competitive if its cost $A(\mathcal{R})$ over any sequence $\mathcal{R}$ satisfies $A(\mathcal{R}) \leq r \cdot OPT(\mathcal{R}) + \beta$, where $OPT(\mathcal{R})$ is the optimal cost of serving $\mathcal{R}$ offline, and β is a constant. In the page fault model, where a fault has cost 1 and hits have no cost, the algorithms above are k-competitive, where k is the size of the cache, which is optimal for deterministic algorithms [6]. A competitive ratio of k is achieved by all *marking* and *conservative* algorithms. An algorithm is conservative if it incurs at most k faults on any consecutive subsequence of requests that contains at most k distinct pages [6]. A marking algorithm associates a mark with each

page in its cache (either explicitly or implicitly) and marks a page when it is brought to cache or if it is unmarked and requested. Upon a fault with a full cache, it only evicts unmarked pages if there are any, and unmarks all pages in cache otherwise. The latter event marks the start of a phase, which defines a k-phase partition of a request sequence. LRU and FWF are marking algorithms, while LRU and FIFO are conservative algorithms [6]. Randomized algorithms with optimal competitive ratio $\Theta(\log k)$ exist for this problem [6]. The offline problem can be solved optimally by Belady's algorithm [5]: evict the page in cache that is going to be requested furthest in the future (FITF).

Other cost models for paging differ in the assumptions of applications with respect to the cost of bringing a page into the cache, and the size of pages [18,11,10,29]. Unlike these models, which consider only the cost of faults, the *full access cost* model [27] charges a cost of 1 for a hit, and a cost of $s \geq 1$ for a fault. In this model, marking algorithms achieve a competitive ratio of $1+\frac{(k-1)s}{L+s}$, where L is the average phase length in the k-phase partition of a sequence. In the worst case, $L = k$ and the ratio is $k(s+1)/(k+s)$, which is optimal. The model coincides with the classic model when $s \to \infty$, but can yield competitive ratios that are significantly smaller if s is small or if a sequence has high locality [6], properties that, as we shall see, are also shared by our model.

A related paging model that also includes the amount of cache used in the cost of algorithms is described in [14]. In this model an algorithm can purchase cache slots, and the overall cost of the algorithm is the number of faults plus the cost of purchased cache. As cache may only be bought, the cache size can only increase (with no bound on the maximum size). In our model, however, an algorithm is charged for the number of pages it has in the cache at every step, which can both increase or decrease. In this sense our model charges algorithms for renting cache, while keeping the upper bound k on the maximum cache available. Finally, we note that the idea of memory renting for reducing RAM power consumption was previously mentioned in [12].

1.2 Our Contributions

This work introduces a generic model of efficient cache usage in paging that can be applied to any scenario in which it is desirable for a paging algorithm to minimize the amount of cache it uses.

We define a family of online algorithms that combine the eviction policies of traditional marking or conservative algorithms with cache saving policies. The performance of the algorithms adapts to the relative cost of faults and cache. More precisely, they achieve a competitive ratio of 2 if $\alpha < k$, where $\alpha = f/c$ is the ratio between fault and cache cost, and $\min\left\{k, \frac{\alpha(k+1)}{\alpha+k-1}\right\}$ if $\alpha \geq k$, thus matching the performance of classical algorithms when $f \gg c$. We further parametrize the analysis by considering the locality of reference of the sequence, and show that for sequences with high locality of reference the competitive ratio of our algorithms is at most 2. Simulations on real-world inputs show that our

algorithms are close to optimal in terms of the total cost, and both its cache usage and number of faults are close to those of the optimal offline.

Lastly, we show that the offline problem admits a polynomial time algorithm via a reduction to interval weighted interval scheduling on identical machines.

The rest of this paper is organized as follows. Section 2 introduces the Minimum Cache Usage model and problem. We present an optimal offline algorithm in Sect. 3, and present our results related to online algorithms and simulations in Sect. 4. Due to space constraints, we include only some of the proofs and charts, while the rest appear in the full version [20].

2 Paging with Cache Usage

The paging model we consider in this paper extends classic paging to a model in which the cost of a paging algorithm on a request sequence is a weighted function of the number of faults and the total amount of cache used by the algorithm. An instance of paging with minimum cache consists of a sequence $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ of page requests and a maximum cache size k. Each request r_i is associated with a page σ_j, for $1 \leq j \leq N$, where N is the size of the universe of pages that can be requested. We denote by $page(r_i)$ the page associated with request r_i. A paging algorithm can hold at most k pages in its cache, but can also choose to hold fewer pages, in order to reduce its cache usage.

Definition 1 (Total cache usage). *Let A be a paging algorithm and $\mathcal{R}$ a request sequence. Let $k(i) \leq k$ denote the number of pages in A's cache immediately before request r_i, where k is the maximum cache size. The total cache usage of A when serving $\mathcal{R}$ is defined as $C_A(\mathcal{R}) = \sum_i k(i)$.*

Given a request sequence $\mathcal{R}$ and maximum cache size k, the cost of an algorithm A on $\mathcal{R}$ is defined as $A(\mathcal{R}) = fF_A(\mathcal{R}) + cC_A(\mathcal{R})$, where $F_A(\mathcal{R})$ and $C_A(\mathcal{R})$ are the number of faults and total cache usage of A when serving $\mathcal{R}$, respectively, and $f \geq 0$ and $c \geq 0$ are parameters. The *Minimum Cache Usage* problem is then the problem of serving a request sequence with minimum cost.

In reality a request sequence is revealed in an online fashion, thus our focus is on the performance of online algorithms in terms of their competitive ratio. An online algorithm has competitive ratio r if, given a maximum cache size k, and parameters f and c, for all request sequences $A(\mathcal{R}) \leq r \cdot OPT(\mathcal{R}) + \beta$, where OPT is the optimal offline, r is a function of k, f and c, and β is a constant that does not depend on $\mathcal{R}$. As in classic paging, the steps involved in serving a request r_i are as follows: the page associated with the request is revealed to the algorithm, after which the algorithm acts by possibly evicting one or more pages, and finally the request is served. Thus, all pages evicted in cache in step i were held in cache up to time $i - 1$. A paging algorithm is said to be *lazy* or *demand paging* if it only evicts a page when a page fault occurs. Observe that unlike classic paging, in which any algorithm can be made demand paging without sacrificing performance [6], in our model algorithms can benefit from evicting pages even when there is no page fault.

The relation between the parameters f and c can vary according to the application to emphasize the importance of minimizing faults or using the cache efficiently, or a combination of both. Naturally, an instance with $c = 0$ and $f > 0$ is an instance of the classical model, in which the cost of an algorithm is its number of faults. On the other hand, if $f < c$ then the problem is trivial: an optimal algorithm always evicts the page of each request immediately after serving it. We assume in general that $f \geq c > 0$.

2.1 Applications

The cost model described above provides incentives for an eviction policy to be efficient not only in terms of its faults but also with respect to the use of the resources that are available to it. Thus, the model can be used in any environment where the latter has significance. We mention the following applications.

Shared Cache Multiprocessors. Multi-core processors are equipped with both private and shared caches, with threads running in each core usually competing for the latter type. While there are schedulers that seek to achieve cooperative use of a shared cache, in general paging strategies for individual threads do not act cooperatively but use as much of the available cache as possible. The cost model we propose provides incentives for paging algorithms to balance their own benefits—a fast execution due to a small number of faults—and the benefits they can provide to concurrently running threads. Depending on the values of f and c, an algorithm will favour one or the other.

Energy Efficient Caching. Content Addressable Memories (CAMs) are used in many applications that require high speed searches, and whose primary applications are in network routers [24]. CAMs are indexed by stored data words instead of memory addresses, as in regular caches. Each cell has a matchline that indicates if the stored word in the cell and the searched word match. A search for an input data word first precharges all matchlines, then each cell compares its bits against the searched bits, and matchlines corresponding to non-matching entries are discharged. The overall missing matchline dynamic power consumption for a system with w matchlines can be modeled as $P = wCV^2f$, where C is the matchline capaticance, V is the supply of a matchline and f is the frequency of misses (the power associated with a matchline in a match is small and can be neglected) [24]. The power involved in this operation can be therefore reduced if matchline precharging is controlled based on the valid bit status of each entry [22]: on a search, only valid entries require the precharging of matchlines, thus the power cost of a search can be proportional to the number of valid entries in the cache. In this scenario, a paging algorithm that uses its cache efficiently will contribute to power savings.

3 Offline Optimum

In the next section we describe a simple family of online algorithms for the cache usage problem and analyze their competitiveness. In order to provide a better

intuition for that analysis we first describe a solution to the offline problem. We recast the paging instance as an instance of weighted interval scheduling on identical machines, and use an algorithm for this problem to obtain an optimal polynomial time paging algorithm.

An instance of Weighted Interval Scheduling on Identical Machines consists of a set J of jobs and a number m of available identical machines. Each job has a starting time, a duration, and a weight. In order to be processed, a job must be assigned to a machine immediately after its start time and cannot be interrupted. A machine can process only one job at a time. The goal is to process a subset $J' \subseteq J$ of jobs such that the total weight of jobs in J' is maximized. Equivalently, each job corresponds to an interval in the real line, and we seek to schedule the maximum weight subset of intervals such that at most m intervals overlap at any time. This problem can be solved in polynomial time by reduction to minimum cost flow [3,7].

It will be useful to see a paging problem instance as an instance of interval scheduling on identical machines: each pair of consecutive requests to the same page defines an interval whose start and end times are the times of the requests. In each pair of requests, the second request results in a hit if and only if the corresponding page is kept in the cache since the previous request, or equivalently, if the interval is scheduled.

The connection between interval scheduling and paging has been noted before in [28,8] where it is used to study cache policies in non-standard caches. It is assumed, however, that the reduction applies only when bypassing is allowed. More recently, [13] used this connection to show that offline paging in the fault and bit models is NP-hard by reducing interval packing problems to paging. Unlike our model, these models consider pages (and hence intervals) of different sizes. The reduction we introduce in this paper is from paging to interval scheduling, and it is defined as follows.

Definition 2 (Interval representation of a sequence). *An interval representation of a request $\mathcal{R}$ of length n is a set of intervals $\mathcal{I}(\mathcal{R}) = \{I_1, I_2, \ldots, I_n\}$ where each interval I_i corresponds to request r_i in $\mathcal{R}$. The starting time of each interval I_i is $s(I_i) = i + 1$ and the end time is $e(I_i) = j - 1$, where $j > i$ is the smallest index such that $page(r_j) = page(r_i)$, or $e(I_i) = n$ if no such j exists. We say that an interval I_i is* feasible *if $e(I_i) < n$ and* unfinished *otherwise. Thus the length of interval I_i is $|I_i| = e(I_i) - s(I_i) + 1$.*

An example of a sequence and its interval representation is shown in Fig.1. Intuitively, an interval corresponding to request r_j represents the time interval in which $page(r_j)$ must reside in the cache in order for the next request to this page to result in a hit. Note that each first request to a page has no preceding interval thus cannot be a hit. Similarly, a page that is requested for the last time in a sequence can be held in cache, but as the interval does not finish in the corresponding page, it cannot result in a hit. Note that intervals do not overlap with the times in which their corresponding pages are requested, thus using this reduction theres is no need to assume that bypassing is allowed. All requests are served, but only the ones whose interval was scheduled will result in hits. Note as

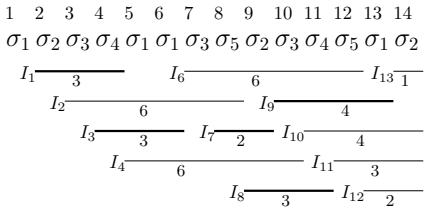


Fig. 1. A request sequence and its interval representation. The length of each interval is shown below the interval (I_5 of length 0 is not shown). Feasible intervals are $\{I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9\}$ while $\{I_{10}, I_{11}, I_{12}, I_{13}\}$ are unfinished. The request can be served with a cache of size 3 with 8 faults and a cache usage of 29 by scheduling intervals $\{I_1, I_3, I_5, I_7, I_8, I_9\}$ on 2 machines (thus requests 5,6,7,10,12,14 are hits and the rest are faults), which is the optimal cache cost for the minimum number of faults.

well that two consecutive requests to the same page define an interval of length 0 that does not overlap any other interval, and thus it is always scheduled. The following lemma formalizes the reduction[1].

Lemma 1. *Let $\mathcal{R}$ be a request sequence. Let $\mathcal{I}' = \mathcal{I}(\mathcal{R}) \setminus \{I_i : I_i \text{ is unfinished}\}$. Let $S \subset \mathcal{I}'$ be a feasible schedule of $\mathcal{I}'$ on $k-1$ machines. Then $\mathcal{R}$ can be served with a cache of size k such that all requests r_j with $I_i \in S$ and $j = i + |I_i| + 1$ are hits, with a total cache usage of $|\mathcal{R}| + \sum_{I_i \in S} |I_i|$.*

In light of Lemma 1, when describing the actions of an algorithm while serving a request $\mathcal{R}$, we sometimes use the terminology related to interval scheduling. Thus we say that an algorithm schedules an interval I_i to mean that it keeps a page $page(r_i)$ in cache until request r_j with $j = i + |I_i| + 1$ (and $page(r_j) = page(r_i)$). We define the *cache cost* of a request r_j as the number of requests that $page(r_j)$ was kept in cache for after r_i, which equals $|I_i|$ if r_j is a hit, and is smaller otherwise.

If we are only interested in minimizing faults then the problem corresponds to Maximal Interval Scheduling. This problem can be solved by sorting intervals in increasing order of end time, and then greedily scheduling intervals while there are available machines. Minimizing the number of faults while at the same time using the least possible cache can be solved instead by computing the maximum weight schedule in the corresponding interval representation. Weighted interval scheduling on identical machines can in turn be solved by formulating the problem as a minimum cost flow problem [3,7]. Since we are interested in minimizing cache usage (equivalently, minimizing processing time in the interval schedule), for a given instance $\mathcal{R}$ we assign weights to intervals using the following corollary from [7]:

Corollary 1. *[7, Cor. 2] For each interval $I_j \in I(\mathcal{R})$ with processing time $|I_j|$, define a weight $w_j = M - |I_j| + 1$ [2], where M is a positive real number such that*

[1] See full version [20] for full proofs.

[2] We add 1 to the weight of each interval so that intervals have non-zero weight if all intervals have length 0.

Algorithm 1. Minimum Cache Usage Cost($\mathcal{R}, k, f, c$)

```
1: {Compute interval representation of R without unfinished intervals}
2: I = ∅ ; M ← 0
3: for j = 1 to |R| do
4:    lastRequest[page(r_j)] = −1
5: for j = 1 to |R| do
6:    i ←lastRequest[page(r_j)]
7:    if i ≠ −1 then
8:       s(I_i) ← i + 1 ; e(I_i) ← j − 1
9:       if c · |I_i| ≤ f then
10:         add I_i to I; M ← M + |I_i|
11:   lastRequest[page(r_j)] = j
12: for i = 1 to |I| do
13:   w(I_i) = M − |I_i| + 1
14: S ←MaxWeightSchedule(I, k − 1)
15: return f(|R| − |S|) + c(Σ_{I_i∈S} |I_i| + |R|)
```

$M \geq \sum |I_j|$. *Then a solution to maximum weight interval scheduling gives an optimal solution to maximal interval scheduling with minimum total processing time.*

Using the above weight assignment and a maximum weight scheduling algorithm we obtain a way of serving request $\mathcal{R}$ with the minimum number of faults, and with minimum cache usage. Recall that in general we seek to minimize the total cost of serving a sequence $\mathcal{R}$, defined as $fF(\mathcal{R}) + cC(\mathcal{R})$, which does not necessarily imply minimizing the number of faults F. However, we can still use the same reduction to interval scheduling and subsequently to minimum cost flow by first eliminating from $\mathcal{I}(\mathcal{R})$ all intervals whose cache cost is higher than the fault cost. It is easy to see that any solution that includes an interval I_i such that $c|I_i| > f$ could be modified to obtain a smaller cost by not scheduling that interval and paying for the fault instead. Hence, an optimal algorithm does not schedule any interval whose cost is higher than that of the fault cost. The resulting optimal offline algorithm is shown in Algorithm 1, where MaxWeightSchedule is an algorithm for maximum weight interval scheduling. Clearly, computing the interval representation of a request $\mathcal{R}$ of n pages (lines 2-13) takes $O(n)$ time, while MaxWeightSchedule takes time $O(m^2 \log m)$ [3], where m is the number of intervals of the weighted interval scheduling problem. Naturally, $m = O(n)$, which yields an $O(n^2 \log n)$ total running time. However, in general m might be much smaller than n, depending on the number of different pages in $\mathcal{R}$ and the number of intervals whose length is greater than f/c.

Theorem 1. *Given a request sequence $\mathcal{R}$ of length n and a cache size k, and constants $f \geq 0$, and $c \geq 0$, an optimal way of serving $\mathcal{R}$ that minimizes $fF(\mathcal{R}) + cC(\mathcal{R})$, where $F(\mathcal{R})$ and $C(\mathcal{R})$ are the number of faults and cache usage when serving $\mathcal{R}$, can be computed in $O(n^2 \log n)$ time.*

4 Online Algorithms

In this section we present a family of online algorithms that adapt to the relative cost of a fault versus the cache cost. These algorithms are k-competitive in the worst case (when $f \gg c$), but can achieve significant cache savings and smaller cost when the cache cost is closer to the fault cost. As a warm-up, we show that while classical optimal paging algorithms are also k-competitive, this ratio does not improve when the cache cost is high relative to the fault cost.

Lemma 2. *Let A be any marking or conservative paging algorithm. The competitive ratio of A is at most k.*

Proof. Let $\mathcal{R}$ be any sequence and consider its k-phase partition. Since A is marking or conservative, it faults at most k times per phase. In addition, in a phase of m requests any algorithm has a cache cost of at most cmk. On the other hand, any algorithm must fault at least once per phase, and must pay at least cm for a phase of m requests. Thus $A(\mathcal{R})/OPT(\mathcal{R}) \leq (fk+cmk)/(f+cm) = k$.

Lemma 3. *Let A be any lazy paging algorithm. Then the competitive ratio of A is at least k.*

Proof. Let $\alpha = f/c$ and $c \neq 0$. Suppose that α is finite. Let $\mathcal{R} = \{\sigma_1, \sigma_2, \ldots, \sigma_{k-1}, (\sigma_k)^x\}$, with $\sigma_i \neq \sigma_j$ for all $i \neq j$, and $(\sigma)^x$ denotes a sequence of x consecutive requests to σ. Since A is a lazy algorithm, it will not evict any page from the cache, thus only faulting in the first k requests but using the entire cache until the end of the sequence. Hence, $A(\mathcal{R}) \geq fk + xkc$. An optimal algorithm can use only one cell of cache for a cost of $OPT(\mathcal{R}) = fk + xc$. Since x can be made arbitrarily large and f/c is bounded, the result follows. In the case of an unbounded α, the same sequence used in the classic lower bound of k applies: request the page in $\{\sigma_1, \ldots, \sigma_{k+1}\}$ not currently in the cache. Thus, $A(\mathcal{R}) \geq n(f + c)$ and $OPT(\mathcal{R}) \leq (n/k)f + nkc$ and the ratio approaches k as $\alpha \to \infty$.

4.1 A Family of Cost-Sensitive Online Algorithms

Definition 3. *For any online paging algorithm A, we define A_d as the algorithm that acts like A, except that for each r_i, it evicts page(r_i) at time $i + d$ if this page has not been requested by that time and is still in the cache. In this case, we say that page(r_i) expires at time $i + d$. We say that a page suffers an early eviction if it is evicted as a result of a capacity miss, according to A's eviction policy. Thus, if page(r_i) is not requested or evicted early within $[i, i + d]$, it will reside in cache for $d + 1$ requests.*

We restrict our choice of online algorithms in the definition above to marking and conservative algorithms and set $d = \lfloor \alpha \rfloor = \lfloor f/c \rfloor$. Consider A=LRU. For some instances LRU could have a better cost than LRU_α[3]. We now show, however, that the cost of LRU_α is always at most twice the cost of LRU, while

[3] To keep notation simple, we refer to $A_{\lfloor \alpha \rfloor}$ as A_α.

there exists a sequence for which the cost of LRU is k times worse than the cost of LRU_α, which is the worst possible ratio for a marking algorithm. This direct comparison of two algorithms can be seen as a variation of *relative interval analysis* [15] that uses the cost ratio instead of the cost difference: for algorithms A and B let $Min(A,B) = \liminf_{n\to\infty}(\min_{|\mathcal{R}|=n}\{A(\mathcal{R})/B(\mathcal{R})\})$ and $Max(A,B) = \limsup_{n\to\infty}(\max_{|\mathcal{R}|=n}\{A(\mathcal{R})/B(\mathcal{R})\})$. Then the relative interval of A and B is $\mathcal{I}(A,B) = [Min(A,B), Max(A,B)]$, and $\mathcal{I}(A,B) \subseteq [\gamma,\delta]$ if $\gamma \leq Min(A,B)$ and $Max(A,B) \leq \delta$. Thus, if $\mathcal{I}(A,B) \subseteq [\gamma \geq 1, \delta > 1]$ we say that B dominates A, since on any sequence B is no worse than A and there is at least one sequence for which B is better than A. Lemma 5[1] and Theorem 2 show that $\mathcal{I}(\text{LRU},\text{LRU}_\alpha) \subseteq [1/2, k]$. Thus, although LRU does not properly dominate LRU_α, the latter is generally preferable to the former. Throughout the proofs in this section we use the following lemma:

Lemma 4. *[25, Cor. 11] Let two vectors* $\boldsymbol{x} = (x_1,\ldots,x_n) \geq \mathbf{0}$ *and* $\boldsymbol{y} = (y_1,\ldots,y_n) > \mathbf{0}$ *be given. Let* π *denote a permutation of* $(1,\ldots,n)$*. Then*

$$\frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} \leq \min_\pi \max\left\{\frac{x_i}{y_{\pi(i)}} : 1 \leq i \leq n\right\} \leq \max\left\{\frac{x_i}{y_{\pi(i)}} : 1 \leq i \leq n, \text{ and fixed } \pi\right\}$$

Lemma 5. *Let* $\alpha = f/c$ *be finite. Then* $Max(LRU,LRU_\alpha) = k$*.*

Theorem 2. *Assume* $k \geq 2$*. Then, for all* $\mathcal{R}$*,* $LRU_\alpha(\mathcal{R}) \leq 2\ LRU(\mathcal{R})$*, and thus* $Min(LRU,LRU_\alpha) \geq 1/2$*.*

Proof. Let $\mathcal{R}$ be any sequence. Let F and C denote the faults and cache cost of LRU on $\mathcal{R}$ and let F_α and C_α denote the corresponding costs for LRU_α. Let $C_\alpha = C_{fh} + C_{hh} + C_{ff} + C_{hf} + \gamma$, where C_{fh} is the cache cost of requests that are faults for LRU_α and hits for LRU, and C_{ff}, C_{hh}, and C_{hf} are defined analogously. γ is the cost of keeping unfinished intervals. We will use the following properties: (1) every page of a request sequence is kept in LRU's cache for at least as long as in LRU_α's cache; and (2) any request that is a fault for LRU_α and is a hit for LRU corresponds to a page that expired in LRU_α's cache.

To see that Property (1) holds, note that if LRU evicts a page σ upon request r_i, then either σ has also expired in LRU_α's cache, or it is evicted at this point on request r_i as well. The latter holds because if σ was evicted from LRU's cache, then there are k distinct requests since the last request to σ, and since σ has not expired in LRU_α's cache, there are $k-1$ pages in LRU_α's cache that have not expired either and are younger than σ. Hence upon request r_i, LRU_α evicts σ as well. Property (1) implies that every request that is a hit for LRU_α is a hit for LRU, and thus $C_{hf} = 0$. Property (2) follows from the fact if LRU_α evicts a page σ due a capacity miss, then its cache is full and since all pages stay longer in LRU's cache, then LRU's cache holds the same pages and evicts σ as well, hence the next request to σ is also a fault for LRU.

Property (1) implies as well that LRU's cache cost is $C \geq C_{fh} + F_\alpha - F + C_{ff} + C_{hh} + \gamma$. Moreover, both properties imply that $C_{fh} = \lfloor\alpha\rfloor(F_\alpha - F)$. Hence,

$$\begin{aligned}\frac{LRU_\alpha(\mathcal{R})}{LRU(\mathcal{R})} &\leq \frac{fF_\alpha + c(\lfloor\alpha\rfloor(F_\alpha - F) + C_{hh} + C_{ff} + \gamma)}{fF + c(\lfloor\alpha\rfloor(F_\alpha - F) + F_\alpha - F + C_{ff} + C_{hh} + \gamma)} \\ &\leq \frac{fF_\alpha + c\lfloor\alpha\rfloor(F_\alpha - F)}{fF + c(\lfloor\alpha\rfloor(F_\alpha - F) + F_\alpha - F)} \qquad \text{(by Lemma 4)} \\ &= \frac{\alpha F_\alpha + \lfloor\alpha\rfloor(F_\alpha - F)}{\alpha F + \lfloor\alpha\rfloor(F_\alpha - F) + F_\alpha - F}\end{aligned}$$

The above expression is bounded above by 2 if $\alpha \geq 2$. The case $\alpha < 2$ is covered by the upper bound on the competitive ratio of A_α in Theorem 3. □

4.2 Upper Bound on the Competitive Ratio of A_α

We now show that for any marking or conservative algorithm A, the competitive ratio of A_α adapts to the relative costs of faults and hits, being at most 2 when the cost of faults is relatively small, and matching the competitiveness of traditional paging algorithms when the cost is negligible.[1]

Theorem 3. *Let A be any marking or conservative algorithm and let $\alpha = f/c$. Assume $k \geq 2$. The competitive ratio of A_α is at most $2 - \frac{1+\alpha-\lfloor\alpha\rfloor}{\alpha+1}$ if $\alpha < k$ and $\min\left\{k, \frac{\alpha(k+1)}{k+\alpha-1}\right\}$ if $\alpha \geq k$.*

Lemma 6 gives a lower bound on the competitive ratio for A_α, which matches the upper bound for $\alpha < k-1$. For larger values of α the gap between upper and lower bounds is reduced as α grows. Lemma 7 gives a straightforward smaller lower bound for any online algorithm.[1]

Lemma 6. *For A marking or conservative, the competitive ratio of A_α is at least $2 - \frac{1+\alpha-\lfloor\alpha\rfloor}{\alpha+1}$ if $\alpha < k-1$ and $\frac{\alpha k + k^2/2}{\alpha + k^2}$ otherwise.*

Lemma 7. *The competitive ratio of any online deterministic algorithm is at least $\frac{k(\alpha+1)}{\alpha+k^2}$.*

The classic paging cost model has been criticized for not being able to capture the benefit of online algorithms on sequences with high locality of reference [6]. Various studies have analyzed the competitiveness of paging algorithms in a parameterized manner, attempting to capture relevant characteristics of sequences such as, for example, locality and typical memory accesses [25], and attack rate [23]. We now give a parameterized competitive ratio for A_α that varies with the locality of reference of the input sequence, for which we use the definition in terms of the average phase length in its k-phase partition.[1]

Theorem 4. *Let A be any marking or conservative algorithm, let $\alpha = f/c$, and let $k \geq 2$. Let $\mathcal{R}$ be any request sequence and let ϕ be the number of phases in $\mathcal{R}$'s k-phase partition. Let $L(\mathcal{R}) = |\mathcal{R}|/\phi$. Then $A_\alpha(\mathcal{R})/OPT(\mathcal{R}) \leq 2$ if $L(\mathcal{R}) > k\alpha(\alpha-2)$, and $\frac{A_\alpha(\mathcal{R})}{OPT(\mathcal{R})} \leq 1 + \frac{\alpha k + 1 - \alpha}{\alpha + k - 1 + L(\mathcal{R})}$ otherwise.*

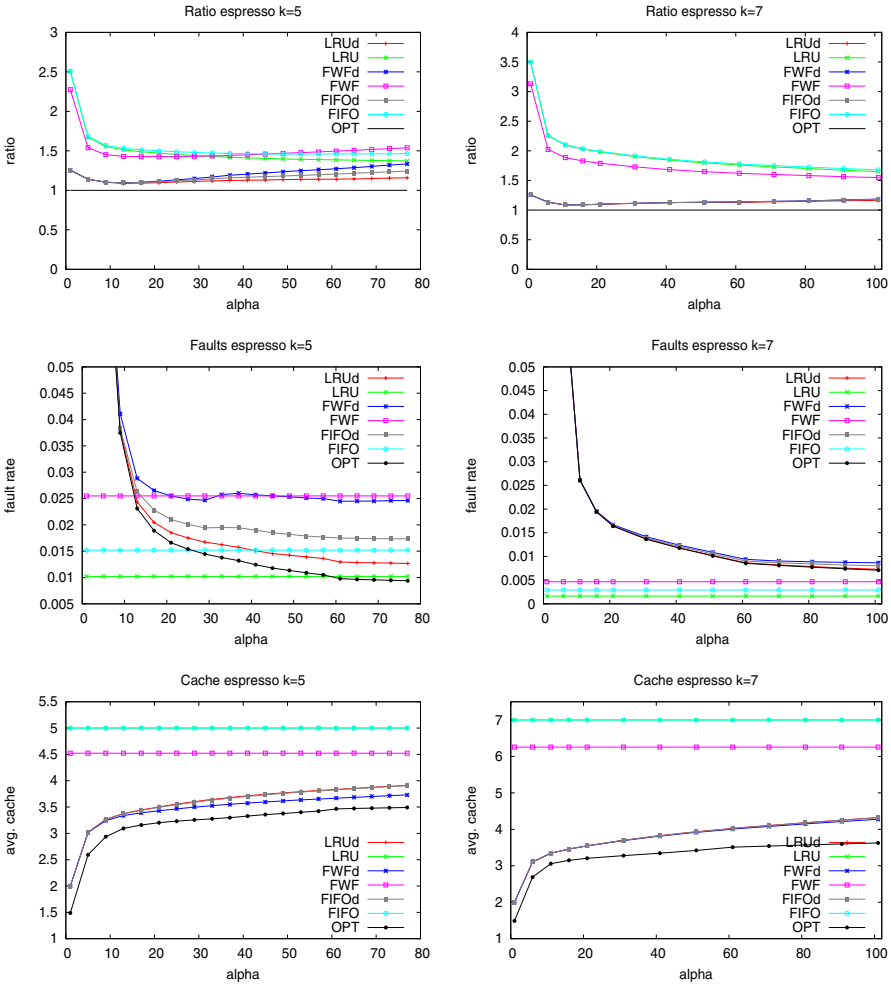


Fig. 2. Cost ratio, fault rate, and average cache used by LRU, LRUd, FWF, FWFd, FIFO, FIFOd, and OPT (with $d = \alpha$) on sequence "espresso" of length 3×10^6 with cache sizes $k = 5$ (average phase length 196) and $k = 7$ (average phase length 1502).

4.3 Real World Sequences

We measured the performance of various algorithms on real world cache traces collected from 4 applications using VMTrace (for Linux) and the Etch tool (on Windows NT)[19]. We obtained the traces from [2] and truncated them to 3×10^6 entries. We simulated LRU, LRU$_\alpha$, FWF, FWF$_\alpha$, FIFO, FIFO$_\alpha$, and OPT on these sequences. For each sequence, we used the size of cache that would yield a fault rate of 1% and 0.1% for LRU. Figure 2 shows the cost ratio compared to OPT, fault rate, and average cache usage for the *espresso* sequence (a circuit

simulator) for two cache sizes. Results for other sequences are shown in the full version [20]. For the total cost we set $c = 1$ and $f = \alpha$. We implemented the optimal offline (Algorithm 1) using the reduction to minimum cost flow in [7], and solved the minimum cost flow instances using the implementation of the cost scaling algorithm from the LEMON C++ library [1]. Results in these practical instances show that the cost of A_α algorithms adapt nicely to the value of α, and that their fault rate and cache usage approaches those ones of the optimal offline. In fact, the ratio A_α/OPT is never more than 2 and in most cases is close to 1. As suggested by Theorem 4, the cost ratio of A_α algorithms improves for sequences with higher locality. Note as well that as α grows, the performance of the traditional marking algorithms gets closer to that of its cost-sensitive counterpart, which is more noticeable for instances with smaller caches.

5 Conclusions

We introduced a model for paging with minimum cache usage and presented a cost-sensitive family of online algorithms whose performance adapts to the relative costs of cache and faults. The cost model that we propose is able to capture locality of reference, yielding a competitive ratio of at most 2 for inputs with high locality. Experiments on request sequences collected from actual programs agree with the theoretical results.

It would be interesting to show a better lower bound for online algorithms, and to propose and analyze other online algorithms, including randomized ones. A natural direction of research would be to evaluate the model in an application, either in theory or in practice. For example, it would be interesting to study and design a global shared caching strategy that varies the relative cache and fault cost for various threads so that the cooperative execution leads to an advantage in overall performance.

Acknowledgements. We would like to thank Francisco Claude, Robert Fraser, Patrick Nicholson, and Hiren Patel for insightful discussions. We are also thankful to the anonymous reviewers of this paper for their useful comments.

References

1. Lemon graph library, `http://lemon.cs.elte.hu/trac/lemon`
2. Trace reduction for virtual memory simulation, `http://www.cs.amherst.edu/sfkaplan/research/trace-reduction/index.html`
3. Arkin, E.M., Silverberg, E.B.: Scheduling jobs with fixed start and end times. Discrete Appl. Math. 18(1), 1–8 (1987)
4. Barve, R.D., Grove, E.F., Vitter, J.S.: Application-controlled paging for a shared cache. SIAM J. Comput. 29, 1290–1303 (2000)
5. Belady, L.A.: A study of replacement algorithms for virtual-storage computer. IBM Systems Journal 5(2), 78–101 (1966)
6. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)

7. Bouzina, K.I., Emmons, H.: Interval scheduling on identical machines. Journal of Global Optimization 9, 379–393 (1996)
8. Brehob, M., Wagner, S., Torng, E., Enbody, R.J.: Optimal replacement is NP-hard for nonstandard caches. IEEE Trans. Computers 53(1), 73–76 (2004)
9. Cao, P., Felten, E.W., Li, K.: Application-controlled file caching policies. In: Proceedings of USTC, vol. 1, pp. 171–182 (1994)
10. Cao, P., Irani, S.: Cost-aware WWW proxy caching algorithms. In: Proceedings of USITS, p. 18 (1997)
11. Chrobak, M., Karloff, H., Payne, T., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. 4(2), 172–181 (1991)
12. Chrobak, M.: SIGACT news online algorithms column 17. SIGACT News 41(4), 114–121 (2010)
13. Chrobak, M., Woeginger, G.J., Makino, K., Xu, H.: Caching is hard - even in the fault model. Algorithmica 63(4), 781–794 (2012)
14. Csirik, J., Imreh, C., Noga, J., Seiden, S.S., Woeginger, G.J.: Buying a constant competitive ratio for paging. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 98–108. Springer, Heidelberg (2001)
15. Dorrigiv, R., López-Ortiz, A., Munro, J.I.: On the relative dominance of paging algorithms. Theor. Comput. Sci. 410(38-40), 3694–3701 (2009)
16. Feuerstein, E., Strejilevich de Loma, A.: On-line multi-threaded paging. Algorithmica 32(1), 36–60 (2002)
17. Hassidim, A.: Cache replacement policies for multicore processors. In: Yao, A.C.C. (ed.) ICS, pp. 501–509. Tsinghua University Press (2010)
18. Irani, S.: Page replacement with multi-size pages and applications to web caching. In: Proceedings of STOC, pp. 701–710. ACM (1997)
19. Kaplan, S.F., Smaragdakis, Y., Wilson, P.R.: Flexible reference trace reduction for vm simulations. ACM Trans. Model. Comput. Simul. 13(1), 1–38 (2003)
20. López-Ortiz, A., Salinger, A.: Minimizing cache usage in paging. Tech. Rep. CS-2012-15, University of Waterloo (2012), `http://www.cs.uwaterloo.ca/research/tr/2012/CS-2012-15.pdf`
21. López-Ortiz, A., Salinger, A.: Paging for multi-core shared caches. In: Goldwasser, S. (ed.) ITCS, pp. 113–127. ACM (2012)
22. Miyatake, H., Tanaka, M., Mori, Y.: A design for high-speed low-power CMOS fully parallel content-addressable memory macros. IEEE JSSC 36(6), 956–968 (2001)
23. Moruz, G., Negoescu, A.: Outperforming LRU via competitive analysis on parametrized inputs for paging. In: Proceedings of SODA, pp. 1669–1680 (2012)
24. Pagiamtzis, K., Sheikholeslami, A.: Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. IEEE JSSC 41(3), 712–727 (2006)
25. Panagiotou, K., Souza, A.: On adequate performance measures for paging. In: Proceedings of STOC, pp. 487–496. ACM (2006)
26. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28(2), 202–208 (1985)
27. Torng, E.: A unified analysis of paging and caching. Algorithmica 20, 194–203 (1998)
28. Wagner, S.: Restricted Cache Scheduling. Ph.D. thesis, Michigan State Univ. (2001)
29. Young, N.E.: On-line file caching. In: Proc. of SODA, pp. 82–86. SIAM (1998)

Competitive-Ratio Approximation Schemes for Makespan Scheduling Problems*

Adam Kurpisz[1], Monaldo Mastrolilli[2], and Georgios Stamoulis[2]

[1] Institute of Mathematics and Computer Science,
Wrocław University of Technology, Wrocław, Poland
adam.kurpisz@pwr.wroc.pl
[2] IDSIA, Manno-Lugano, Switzerland
{monaldo,georgios}@idsia.ch

Abstract. The concept of *competitive-ratio approximation scheme* was recently proposed in [7]. Such a scheme algorithmically constructs an on-line algorithm with a competitive ratio arbitrarily close to the best possible competitive ratio for a given online problem. In this paper we continue this line of research by addressing several makespan scheduling problems and introducing new ideas: we combine the classical technique of structuring and simplifying the input instance for approximation schemes, with the new technique of guessing the end of the schedule (time after which no job is processed and released), which allows us to reduce the infinite-size set of on-line algorithms to a relevant set of finite size. This is the key idea for eventually allowing an enumeration scheme that finds a near optimal on-line algorithm. We demonstrate how this technique can be successfully applied to three basic makespan online over time scheduling problems: scheduling on unrelated parallel machines, job shop scheduling and single machine scheduling with delivery times.

1 Introduction

Scheduling problems form an important class of well studied problems in combinatorial optimization. On-line scheduling can be seen as scheduling with incomplete information. At certain points, decisions have to be made without knowing the complete instance. Depending on the way that new information becomes known, different on-line paradigms are possible. In this paper we consider the case where jobs arrive over time and their data become known at their arrival (release time); the scheduling decision for a job may be delayed. At any time, all the currently available jobs (the ones not scheduled so far) are at the disposal of the decision maker. Decisions made in the past are irreversible. When we are dealing with such on-line problems, we need to define a measurement that distinguish "good" algorithms from "bad" ones. The most popular method in comparing the performance of such on-line algorithms is the *competitive ratio*:

* Supported by the Swiss National Science Foundation Project N.200020-122110/1 "Approximation Algorithms for Machine Scheduling Through Theory and Experiments III" and by Hasler Foundation Grant 11099.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 159–172, 2013.

Definition 1 ([12], [19]). *Let Π be an on-line problem. Assume that Π is a minimization problem. An algorithm $\mathcal{A}$ for Π is said to be ρ-competitive if for every feasible instance I for Π it produces a solution of value $val^{\mathcal{A}}(I) \leq \rho \cdot opt(I)$ where $opt(I)$ is the optimal solution on the offline instance I (i.e. when we treat I as a full information instance).*

The competitive ratio of an algorithm $\mathcal{A}$ (for a minimization problem Π) is the infimum over ρ such that $\mathcal{A}$ is ρ-competitive. The optimal competitive ratio for a problem Π is defined to be the minimum ρ achievable for Π by any algorithm $\mathcal{A}$, and we denote it by ρ^*.

Observe that we do not constraint $\mathcal{A}$ to be an efficient (i.e polynomial time) algorithm. In particular, we do not impose any time or space (or any other measure) constraint on the efficiency of $\mathcal{A}$. For a comprehensive account of the theory of competitive analysis and a thorough treatment of on-line algorithms see [2].

Problems Definition and Related Work: In all the considered problems we assume that each machine can process at most one operation at a time, and each job can be processed by at most one machine at any time. We consider the following on-line scheduling problems in which jobs arrive over time and their properties become known at their release dates, denoted by r_j for each job j; each job j can start being processed after its release date r_j.

1. *Unrelated Parallel Machines ($Rm|r_j|C_{\max}$):* Each job j has a processing time of $p_{ij} \in \mathbb{Q}^{\geq 0}$ on machine i, $\forall i \in [m]$. For each job, we need to decide on which machine and when it starts processing without preemption. We consider the problem of minimizing the maximum job completion time among all jobs (i.e. *makespan*, denoted as $C_{\max}$). We assume that the number m of machines is *fixed* (i.e not part of the input).
2. *Makespan Minimization in Job Shops:* In Job Shops problem, each job J_j consists of a sequence of μ operations $O_{1j}, O_{2j}, \ldots, O_{\mu j}$ that need to be processed in this order. Operation O_{ij} must be processed without interruption on machine $m_{ij} \in M$, during $p_{ij} \in \mathbb{Q}^{\geq 0}$ time units. We also assume that the number m of machines and the number μ of operations per job are fixed. This is the on-line analogue of $Jm|r_j, op \leq \mu|C_{\max}$.
3. *Scheduling in Single Machine with Delivery Times:* Each job has processing time $p_j \in \mathbb{Q}^{\geq 0}$ and delivery time $q_j \in \mathbb{Q}^{\geq 0}$. If $s_j (\geq r_j)$ is the time that job j starts processing, it will be delivered at time $L_j = s_j + p_j + q_j$. The objective is to minimize the *maximum delivery time* $L_{\max} = \max_j L_j$. We assume that the delivery process is a non-bottleneck activity (different jobs can be delivered at the same time). This is the on-line analogue of $1|r_j|L_{\max}$.

The off-line counterparts of these problems have produced several approximation algorithms over the years (since their decision versions are strongly **NP**-complete). For example in [13] a 2 approximation was given for $R||C_{\max}$ and was shown that no better than $\frac{3}{2}$-approximation exists. When the number of machines is fixed, several FPTASes are known for $Rm||C_{\max}$, we refer to [4]

for the relevant literature and because we will use the techniques in [4] in the current paper. For Job Shop, in [20] it was proven that there does not exist a better than $\frac{5}{4}$ approximation algorithm (mod $\mathbf{NP} \neq \mathbf{P}$) which was further improved in [15] to $\Omega((\log \mathrm{lb})^{1-\epsilon})$ (mod $\mathbf{NP} \not\subseteq \mathbf{ZTIME}(2^{\log n^{\mathcal{O}(1/\epsilon)}})$) where lb is a lower bound on the optimal objective function value. For such a case, an $\mathcal{O}((\log(m\mu)\log(\min(m\mu, p_{\max}))\ /\ \log\log(m\mu))^2)$ approximation algorithm exists [5], where $p_{\max}$ is the largest processing time among all operations. For fixed m, μ Shmoys et al. [17] gave a $(2+\epsilon)$ approximation algorithm which has been further improved by Jansen et al. [11] who have designed a $(1+\epsilon)$ approximation algorithm. This has been further simplified and improved (w.r.t the running time) in [4]. For $1|r_j|L_{\max}$ many PTASes are known: in [8], [9] two PTASs of running time $\mathcal{O}((\frac{n}{\epsilon})^{\mathcal{O}(1/\epsilon)})$ and $\mathcal{O}(n\log n + n(1/\epsilon)^{\mathcal{O}(1/\epsilon 2)})$ were given, which were further improved to a PTAS with running time $\mathcal{O}(n + (1+\epsilon)^{\mathcal{O}(1/\epsilon)})$ in [14]. In this paper we will heavily use the results in [4,14].

Online makespan minimization problems have already been considered under two main online models. The first one, when jobs arrive one by one, has been extensively studied through the last 50 years. In 1966 Graham [6] showed that for m identical parallel machines scheduling, the List algorithm which always puts the next job on the least loaded machine is exactly $(2-\frac{1}{m})$-competitive. For more results see e.g. [10], [16] and the references therein. The latter model, much less studied, consider jobs arriving online over time. For the $Pm||C_{max}$ problem Chen and Vestjens [3] gave nearly tight bounds on the optimal competitive ratio $(1.347 \leq \rho^* \leq 3/2)$.

Moreover, in [18] the authors describe a general technique to convert a non-clairvoyant scheduling algorithm for a problem with all jobs released at the same time to a non-clairvoyant on-line algorithm that can handle unknown release dates. They show that the quality of the schedule thus constructed is within a factor of 2 of the quality of schedules constructed by the on-line algorithm in the simpler environment. This technique does not only apply to parallel machine scheduling, but also to the entire class of shop scheduling problems. On the other hand, for the sum of weighted completion time objective function, this online model was intensively studied recently in [7] where, to the best of our knowledge, the concept of *competitive-ratio approximation scheme* has been proposed for the first time:

Definition 2 (Competitive-Ratio Approximation Scheme [7]). *An* competitive - ratio approximation scheme *(CRAS) computes for a given $\epsilon > 0$ an online algorithm A with a competitive ratio $\rho_A \leq (1+\epsilon)\rho^*$. Moreover, it determines a value ρ' such that $\rho' \leq \rho^* \leq (1+\epsilon)\rho'$.*

In [7] the authors study the problem of scheduling jobs online to minimize the weighted sum of completion times on parallel, related, and unrelated machines, and derive both deterministic and randomized algorithms which are almost best possible among all online algorithms of the respective settings. They also generalize their techniques to arbitrary monomial cost functions and apply them to some makespan scheduling problems with fixed number of machines, namely, for

the preemptive and non preemptive related parallel machines problem, denoted as $Qm|r_j, (pmtn)|C_{\max}$, and for the unrelated parallel machines problem when preemption is allowed, denoted as $Rm|r_j, pmtn|C_{\max}$. The novel technique presented there works as follows: First the input is structured and simplified. After that, a proof is given that there are constant number of "situations" that an on-line algorithm can encounter at any decision step. Then, it is shown that once a situation had been encountered, this situation (or another equivalent to this) will appear after constant number of steps i.e. specific configurations *cycle*. This helps focus attention on a very limited number of configurations: any instance is either captured or being equivalent with such a configuration, and so an complete enumeration can be done.

Our Results: Our contribution is online approximation schemes for the three non-preemptive makespan scheduling problems described before.

We combine the classical technique of structuring and simplifying the input instance for approximation schemes, with the new technique of guessing the end of the schedule (time after which no job is processed and released), which allows us to reduce the infinite-size set of on-line algorithms to a relevant set of finite size. This is the key idea for eventually allowing an enumeration scheme that finds a near optimal on-line algorithm.

The enumeration technique from [7] was the inspiration for our work, although the details differ to tackle with the addressed basic makespan scheduling problems. In particular, we use a different approach in order to successfully enumerate all possible scenarios for the problems under consideration (for example the approach presented in [7] do not seem to generalize in a straightforward way to the case of $Rm|r_j|C_{\max}$ when preemption is not allowed). In order to (approximately) compute the best possible competitive ratio, we transform every input instance to an equivalent one with certain nice properties with only small loss. We in fact prove that we can actually restrict our attention to scenarios of "small" size and enumerate all possible situations and all possible schedules (algorithms) for these situations, efficiently. The output will be an on-line approximation scheme for $Rm|r_j|C_{\max}$, $Jm|r_j|C_{\max}$ and $1|r_j|L_{\max}$. Since the unbounded computational power is given, the approximation scheme provide an algorithm for any given number of machines m, even when the number of machines is part of the input. However, we note that computing the near optimal value of competitive ratio can be done only in the case when m is fixed.

2 General Framework

In this section we will describe the general idea we use to obtain online approximation schemes for the addressed problems. Our approach uses several classic transformations (e.g. [1]) of the input instance which may potentially increase the objective function value by a factor of *at most* $1 + \mathcal{O}(\epsilon)$. Throughout this paper, when we describe this type of transformation, we shall say it produces $1 + \mathcal{O}(\epsilon)$ loss.

Algorithm 1. Enumeration Scheme

Input: A set of jobs J_x currently available for processing at the beginning of the interval I_x.

Output: Algorithm $\mathcal{A}$ with nearly optimal competitive ratio among all possible schedules (algorithms) for J_x.

1. Set the $LB_x, UB_x = EOS_x$ accordingly (see the next section, EOS_x refers to the currently guessed end of schedule).
2. Partition the time horizon ($[LB_x, EOS_x]$) into bounded number of subintervals (again, see Section 3). This can be done since $\frac{EOS_x}{LB_x} < \infty$.
3. Let the current released instance composes of jobs J_x.
4. Define the set $\mathbb{I}'_x$ of all possible instances for the rest of the intervals that respects EOS_x (as described in Section 3).
5. For every schedule (algorithm) f for J_x together with instances from $\mathbb{I}'_x$ that respects the EOS_x **DO**:
   (a) For every possible possible reduced instance $I' \in \mathbb{I}'_x$ **DO**:
      i. Schedule $J_x \cup I'$ according to f.
      ii. Let $D_{\max}(J_x, I')$ be the completion time of this instance (under f).
      iii. Define the offline instance composed by J_x, I' and solve it optimally to obtain $D^{opt}_{\max}(J_x, I')$.
   (b) Define $\rho_f^{J_x} = \max_{I'} \frac{D_{\max}(J_x, I')}{D^{opt}_{\max}(J_x, I')}$
6. Output $\mathcal{A}$ such that $\mathcal{A} = \arg\min_f \rho_f^{J_x}$.

First, using a result from [1], we prove that the release dates and processing times can be structured to take only specific values (see Lemma 1). In particular, all the release dates of the jobs, r_j, can be structured to be integer powers of $(1+\epsilon)$. This assumption can be done with $(1+\epsilon)$ loss in the objective function value. This allows us to partition the time horizon into *intervals* $I_x = [R_x, R_{x+1})$ with $R_x = (1+\epsilon)^x$ such that "interesting things" (release of jobs) happen only at the beginning of the intervals, i.e. at time R_x.

Assume that we are at a particular time step R_x, for some integer x. Some jobs arrive (being released) at this step, plus we are possibly left with some unprocessed jobs released in previous intervals. Now we want to compute a nearly optimal schedule for those jobs available for processing at R_x, let's call them J_x. In order to do this, we introduce the technique of "guessing the end of schedule" for the current set of jobs J_x. Intuitively, we set a lower bound LB_x as a minimum amount of time needed to process all jobs available for processing at time R_x and also an *upper bound* for the completion time of whole schedule (EOS_x). In other words, EOS_x is a time after which no job is being processed. We prove that if our guess of EOS_x was indeed correct, then the scheduling of the jobs J_x can be done in nearly optimal way. Otherwise (by appropriately setting of EOS_x) we prove that if our guess was wrong, then the jobs J_x become insignificant (with respect to their contribution to the objective function value). So, even the worst possible schedule for those jobs cannot affect more than a factor $(1+\mathcal{O}(\epsilon))$ the objective function value.

We denote the infinite set of instances that may appear in the future with respect to EOS_x as $\mathbb{I}_x$. In order to prove that, under the assumption EOS_x is correct, the jobs J_x are scheduled nearly optimally, we use a series of transformations and simplifications of instances from [4], that help us focus attention on instances with specific structure.

More precisely, in order to schedule nearly optimally the jobs J_x, we show that any instance composed of jobs that may appear in the future and respect the previously described EOS_x, can be mapped to another instance I' such that in I' we have only "few" jobs for processing (basically by merging jobs together) and the data for each job (e.g. its processing time) can take only few (bounded) different values. More importantly, this transformation can be done with $(1 + \mathcal{O}(\epsilon))$ loss. So in fact, instead of considering the entire *infinite* set of instance $\mathbb{I}_x$, we can focus on the finite set of *reduced* instances $\mathbb{I}'_x$ that may appear in the future with respect to the current EOS_x, without affecting much the objective function value. The fact that $\frac{EOS_x}{LB_x} < \infty$ helps us enumerate all of them, because only a bounded number of such reduced instances can exist. For each such instance I' we enumerate all possible schedules for J_x and we output the one with the "best behavior". The final schedule for jobs in J_x is the schedule that has the best *worst* case performance among all schedules and among all instances I' (Algorithm 1).

3 Structuring and Transforming the Instance

In this section we will show how simple transformations (some of which are on the spirit of the [1]) allow us to focus attention on instances with certain "nice" properties. In particular, we will prove all the assumptions and claims of the previous section. For concreteness we will show how everything applies to on-line $Rm|r_j|C_{\max}$. The properties that we assume are the following:

1. With $1 + \epsilon$ loss all the release dates r_j and processing times p_{ij} of jobs are integer powers of $(1 + \epsilon)$.
2. Any instance $I' \in \mathbb{I}'_x$ needs bounded information in order to be fully described.
3. For fixed x, there is a bounded number of instances $\mathbb{I}'_x$ that approximates the entire set of instances $\mathbb{I}_x$ that respects the currently guessed EOS_x.
4. For a currently guessed EOS_x at some point x, if the guess is not correct then the set of jobs J_x becomes irrelevant.

Property 1 structures the release dates thus let us focus only on the beginning of intervals $I_x = [R_x, R_{x+1})$ where $R_x = (1+\epsilon)^x$. In order to guarantee these properties, we also need to show that we need bounded data in order to fully describe any instance I'. This is exactly the second requirement. The third property says that, when we fix LB_x and EOS_x, for any x, then there is only a bounded number of scenarios that can arrive in the future (where the future is bounded by EOS_x). In fact, there might be an unbounded number of situations that can appear within these two bounds, but we prove that with at most $(1 + \mathcal{O}(\epsilon))$

loss, $\forall I \in \mathbb{I}_x$, $\exists I' \in \mathbb{I}'_x$ that approximates I. The latter means that the objective function values of I and I' after applying an optimal (on-line) algorithm will not differ by more than a $1 + \mathcal{O}(\epsilon)$ factor. In other words, the bounded size set $\mathbb{I}'_x$ approximates the infinite size set $\mathbb{I}_x$ with arbitrary precision.

Finally, we need the property that if our current guess of the end of schedule EOS_x is not correct, then even the worst possible decision regarding the scheduling of jobs J_x cannot affect much (more than $(1 + \mathcal{O}(\epsilon))$ factor) the objective function value.

In fact, for any problem that the previous properties are true and can be done in finite time, the Algorithm $\mathcal{A}$ of the previous section can be applied to produce a near optimal on-line algorithm. In the next subsections we demonstrate how these properties are true for $Rm|r_j|C_{\max}$, and in the appendix for the rest of the problems mentioned in the introduction. When we want to say that a quantity (e.g. ξ) take a finite (bounded) value, we will simply write ξ-finite.

3.1 Guaranteeing Property 1

Let $\epsilon \in \mathbb{Q}^{(0,1]}$ and let us denote OPT the value of the optimal off-line schedule. Let us define $d_j := \min_{i=1,..,m}(p_{ij})$ as a *scale factor* of a job J_j. W.l.o.g we assume $d_j > 0$ for any job (otherwise it is always optimal to schedule job j on the machine i with $p_{ij} = 0$). We bound several relevant parameters by constants. If not stated differently, any mentioned constant depends only on ϵ. When we say that some parameter is finite it means that it is bounded by some function depending on input parameters i.e for $Rm|r_j|C_{max}$: ϵ and m. To fulfill Property 1 we prove the following Lemma.

Lemma 1. *With $1 + \mathcal{O}(\epsilon)$ loss, we can restrict to instances where for each job J_j the processing time on machine M_i (p_{ij}) and release date (r_j) are powers of $1 + \epsilon$ and $r_j \geq \epsilon d_j$ for every job j, thus $r_j > 0$.*

Proof. Assume that we have an optimal schedule. Let $\tilde{LB}$ be the minimum amount of time that is needed to process all already released jobs. For all these jobs job J_j, $\tilde{LB} \geq d_j$ and $\tilde{LB} \geq \frac{\sum d_j}{m}$.

First of all, we move all jobs to the right by the value $\epsilon\tilde{LB}$. Obviously the schedule is still feasible. In the new schedule every job j finishes at time $C_j + \epsilon\tilde{LB}$. Job j does not start before $C_j + \epsilon\tilde{LB} - p_{ij} \geq \epsilon\tilde{LB}$. This implies that we can increase release dates to enforce $r_j \geq \epsilon\tilde{LB} \geq \epsilon d_j$. Now we can multiply every release date and processing time by $1 + \epsilon$; this increases the objective function at most by a factor $1 + \epsilon$. Then we decrease each release date and processing time to the nearest power of $1 + \epsilon$ (which is at least the original value). The latter can only improve things. □

3.2 Guaranteeing Property 2

First we will show how we can map an arbitrary instance I to another I' such that I' has finite number of different jobs that seek processing.

We take the decision only at the beginning of each interval $I_x = [R_x, R_{x+1})$, where $R_x = (1+\epsilon)^x$, $x \in \mathbb{Z}$. There are two disjoint situations that an on-line algorithm can encounter regarding the sum $\sum_{j \in J_x} d_j$ together with the remainder of the processing time of the "crossing" jobs (jobs that started their processing in previous intervals but have not yet finished in R_x):

1. either it is smaller or equal than $|I_x|$ or,
2. it is greater than $|I_x|$.

For the first case we can just apply the PTAS in [4] to schedule jobs in this interval. Otherwise let $LB_x = \frac{|I_x|}{m} = \frac{\epsilon(1+\epsilon)^x}{m}$ (minimum amount of time needed to process jobs available for processing in interval I_x together with the remainder of the crossing jobs) and an "end of schedule", $EOS_x = \xi LB_x$, where $\xi = \frac{m}{\epsilon}\left(1+\frac{m}{\epsilon^3}\right)^2 < \infty$ (see Lemma 12). The EOS_x can be identified as an upper bound for amount of the time required to process all jobs of a whole on-line instance of scheduling problem.

At the beginning of each interval I_x, we reduce number of jobs released at time R_x to some finite number using the lemma below.

Lemma 2 ([4]). *Assuming that $\frac{EOS_x}{LB_x} = \xi$, ξ-finite, then with $(1+\mathcal{O}(\epsilon))LB_x$ loss we can reduce in linear time the number of jobs that are available for processing at time R_x to be at most $\Delta = (\log \frac{m}{\epsilon})^{\mathcal{O}(\xi)}$-finite.*

To fulfill the technicalities we prove the following Lemma:

Lemma 3. *The transformations of Lemma 1 and Lemma 2 cannot affect the optimal competitive ratio more than a $(1+\mathcal{O}(\epsilon))$ factor.*

Proof. Let us fix some instance. Let A denotes the value of objective function of optimal on-line schedule for this fixed instance. Let A_R, OPT_R, ρ_R be equivalent to A, OPT, ρ for the reduced instance. From Lemma 2 we get $A_R \leq A(1+\epsilon)$, thus:

$$\rho_R = \frac{A_R}{OPT_R} \leq \frac{A(1+\epsilon)}{OPT} = \rho$$

□

Now for each job J_j, we define the following sets of machines: $\mathcal{F}_j = \{i : p_{ij} \leq \frac{\epsilon}{m} d_j\}$ are the "fast" machines for J_j and $\mathcal{S}_j = \{i : p_{ij} \geq \frac{m}{\epsilon} d_j\}$ are the "slow" machines for J_j.

We structure the input data of any job J_j $(j = 1, \ldots, n)$ as follows

1. For any "fast" machine $i \in \mathcal{F}_j$ for job J_j, set the corresponding processing time to zero, $p_{ij} := 0$.
2. For any "slow" machine $i \in \mathcal{S}_j$ for job J_j, set the corresponding processing time $p_{ij} := +\infty$.
3. For any other machine $i \in [m] \setminus \{\mathcal{F}_j \cup \mathcal{S}_j\}$, round p_{ij} down to the nearest lower value of $\frac{\epsilon}{m} d_j (1+\epsilon)^h$, for some $h \in \mathbb{N}$.

Lemma 4 ([4]). *With $1 + \mathcal{O}(\epsilon)$ loss we can assume that any instance can be structured with the above three conditions.*

Consider the input instance after the structuring step. We define the *job profile* of a job J_j to be a (m)-tuple $< \Pi_{1,j}, \ldots, \Pi_{m,j} >$, such that $\Pi_{ij} = -\infty$ if $i \in \mathcal{F}_j$, $\Pi_{ij} = +\infty$ if $i \in \mathcal{S}_i$ and Π_{ij} is such that $p_{ij} = \frac{\epsilon}{m} d_j \, (1+\epsilon)^{\Pi_{ij}}$ otherwise. Observe that any job J_j is completely defined by its *job profile*, *scale factor* d_j and *release date* r_j.

Lemma 5 ([4]). *The number of distinct job profiles as defined above is at most* $l := \left(3 + 2\log_{1+\epsilon} \frac{m}{\epsilon}\right)^m$.

Now, we define *big* and *small* jobs for interval I_x: $L_x := \{j \in J : r_j = R_x, d_j > \frac{\epsilon^2}{m\Delta} R_x\}$, $S_x := \{j \in J : r_j = R_x, d_j \leq \frac{\epsilon^2}{m\Delta} R_x\}$.

Now in each interval I_x we partition L_x into sets of *relevant jobs* and *irrelevant jobs*. Intuitively a job is supposed to be irrelevant, if any decision concerned with this job cannot much affect the result. In other words with $1+\mathcal{O}(\epsilon)$ loss irrelevant jobs can be scheduled arbitrarily.

Lemma 6. *With $(1 + \mathcal{O}(\epsilon))$ loss all small jobs are irrelevant.*

Proof. We schedule any small job J_j on the machine with smallest processing time, d_j. From Lemma 2 and the definition of small jobs, the total processing time of small jobs in interval I_x is bounded by $\Delta \frac{\epsilon^2}{m\Delta}(1+\epsilon)^x = \epsilon \frac{|I_x|}{m} = \epsilon LB_x$. □

Lemma 7. *The number of distinct scale factors of jobs in each set L_x is bounded by* $\log_{1+\epsilon} \frac{\Delta\xi}{\epsilon^2}$.

Proof. Take a large job $J_j \in L_x$ with scale factor $d_j = (1+\epsilon)^y$. Before reducing the number of jobs (Lemma 2), from Lemma 1 we have that for J_j, $d_j \leq \frac{r_x}{\epsilon} = \frac{(1+\epsilon)^x}{\epsilon}$ and also $d_j > \frac{\epsilon^2}{m\Delta} r_x$ ($J_j \in L_x$ and so $r_j = R_x$). Thus

$$\frac{\epsilon^2}{m\Delta}(1+\epsilon)^x < (1+\epsilon)^y \leq \frac{(1+\epsilon)^x}{\epsilon}$$

The number of distinct integers y which satisfy the above inequalities is bounded by $\log_{1+\epsilon} \frac{m\Delta}{\epsilon^3}$. After reducing the number of jobs we can no longer ensure that for every job J_j, $d_j \leq \frac{r_j}{\epsilon}$ (we do not interfere inside the "reducing number of jobs" procedure, i.e. some gluing technique may increase the jobs processing time). We know that for every interval I_x, $EOS_x = \xi \frac{\epsilon(1+\epsilon)^x}{m}$, and that for every job J_j, $d_j \leq EOS_x$. For these jobs the following inequalities hold

$$\frac{(1+\epsilon)^x}{\epsilon} < (1+\epsilon)^y \leq \xi \frac{\epsilon(1+\epsilon)^x}{m}.$$

The number of distinct integers y which satisfy above inequalities is bounded by $\log_{1+\epsilon} \frac{\epsilon^2 \xi}{m}$. The total number of distinct scale factors of large jobs is bounded by $\log_{1+\epsilon} \frac{\Delta\xi}{\epsilon}$. □

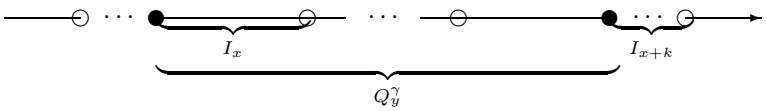
From Lemmas 2, 3, 4, 5, 6 and 7, Property 2 for online $Rm|r_j|C_{\max}$ holds.

3.3 Guaranteeing Properties 3 and 4

We partition the time line into periods $Q_y^\gamma := [\sum_{i=-\infty}^{y}(1+\frac{\gamma}{\epsilon})^i, \sum_{i=-\infty}^{y+1}(1+\frac{\gamma}{\epsilon})^i)$, $\gamma \geq 1$ is a parameter, defined later. $|Q_y^\gamma|$ is the length of period y, $|Q_y^\gamma| = (1+\frac{\gamma}{\epsilon})^{y+1}$.

Lemma 8. *For any finite $\gamma \geq 1$ each period Q_y^γ intersects with finitely many intervals. The number of intersected intervals is at most $\log_{1+\epsilon}(1+\frac{\gamma}{\epsilon})+2$.*

Proof. First assume that there exist intervals I_x, I_{x+k} such that their starting times are equal to $\sum_{i=-\infty}^{y}(1+\frac{\gamma}{\epsilon})^i$, $\sum_{i=-\infty}^{y+1}(1+\frac{\gamma}{\epsilon})^i$ respectively.



● starting, ending points of periods,
○ starting, ending points of intervals.

Let $A := 1+\frac{\gamma}{\epsilon}$. We have that

$$(1+\epsilon)^x = \sum_{i=-\infty}^{y} A^i = \frac{A^{y+1}}{A-1} \Rightarrow x = \log_{1+\epsilon}\frac{A^{y+1}}{A-1}$$

and in similar way we can get

$$x+k = \log_{1+\epsilon}\frac{A^{y+2}}{A-1} \Rightarrow k = \log_{1+\epsilon}\frac{A^{y+2}}{A^{y+1}} = \log_{1+\epsilon}A$$

Since, at the beginning of this proof, we made an assumption on starting times of intervals I_x, I_{x+k}, we have that the exact number of intervals that intersect any period is bounded by $\log_{1+\epsilon}(1+\frac{\gamma}{\epsilon})+2$ (the assumption can increase the number of intervals intersecting with the period by at most 2). □

Lemma 9. *Q_y^γ has the property that $\sum_{i=-\infty}^{y-1}\gamma|Q_i^\gamma| = \epsilon|Q_y^\gamma|$, for all $y \in \mathbb{Z}$, γ-finite.*

Proof. The following inequalities hold

$$\sum_{i=-\infty}^{y-1}\gamma|Q_i^\gamma| - \epsilon|Q_y^\gamma| = \gamma\frac{\left(1+\frac{\gamma}{\epsilon}\right)^{y+1}}{\left(1+\frac{\gamma}{\epsilon}\right)-1} - \epsilon\left(1+\frac{\gamma}{\epsilon}\right)^{y+1} =$$

$$\frac{\left(1+\frac{\gamma}{\epsilon}\right)^{y+1}}{\frac{1}{\epsilon}} - \epsilon\left(1+\frac{\gamma}{\epsilon}\right)^{y+1} = 0$$

□

Lemma 10. *In each interval I_x, with $1 + \mathcal{O}(\epsilon)$ loss, we can assume that the total sum of d_j of jobs released in this interval is bounded by $\alpha|I_x|$, for $\alpha = \frac{m}{\epsilon^2}$.*

Proof. From Lemma 1 and from definition of $\tilde{LB}$ for any interval I_x we get:

$$R_x \geq \epsilon \tilde{LB} \geq \epsilon \frac{\sum_{j \in J: r_j \leq R_x} d_j}{m},$$

thus

$$\sum_{j \in J: r_j = R_x} d_j \leq \sum_{j: r_j \leq R_x} d_j \leq m \frac{R_x}{\epsilon} = m \frac{|I_x|}{\epsilon^2}$$

and the claim follows. □

Lemma 11. *In each interval starting in period Q^α_y, with $1 + \mathcal{O}(\epsilon)$ loss, all jobs released in intervals that ends within any period $\{Q^\alpha_{-\infty}, \ldots, Q^\alpha_{y-2}\}$ are irrelevant. In each interval there is a finite number of relevant jobs, either released in this interval or postponed from previous intervals.*

Proof. The total sum of d_j of jobs released in all intervals that ends within any period between $\{Q^\alpha_{-\infty}, \ldots, Q^\alpha_{y-2}\}$ is bounded by $\sum_{i=-\infty}^{y-2} \alpha |Q^\alpha_i| \leq \epsilon |Q^\alpha_{y-1}|$ by Lemma 10 and Lemma 9. Thus only intervals which intersect the current and the previous period can have relevant jobs. From Lemma 2 and Lemma 8 we get that in each interval there is a finite number of relevant jobs, either released in this interval or postponed from previous intervals. □

Next lemma ensures that if the EOS_x is not correct it does not matter what decision will be made at this time, incorrect EOS_x cannot much affect the optimal solution.

Lemma 12. *With $1 + \mathcal{O}(\epsilon)$ loss, in each interval I_x while taking decisions, we can assume that there exists a lower bound ($LB_x = \frac{\epsilon(1+\epsilon)^x}{m}$) for jobs released in interval I_x and an "end of schedule" (EOS_x), such that $EOS_x = \xi LB_x$, $\xi = \frac{m}{\epsilon}\left(1 + \frac{m}{\epsilon^3}\right)^2$-finite, unless all jobs released in this and previous intervals are irrelevant.*

Proof. Suppose that interval I_x starts in period Q^α_y. The lower bound LB_x for jobs released in interval I_x can be set as $\frac{|I_x|}{m}$ (we have already assumed that the total sum of d_j of jobs available for processing in interval I_x is equal at least $|I_x|$). Since $R_x \geq \sum_{j=-\infty}^{y-1} |Q^\alpha_j| = \frac{A^{y+1}}{A-1}$ and analogously $LB_x \geq \frac{\epsilon}{m} \frac{A^{y+1}}{A-1}$. We set as "end of schedule" EOS_x as $\left(\sum_{j=-\infty}^{y+1} |Q^\alpha_j| - \sum_{j=-\infty}^{y-1} |Q^\alpha_j|\right) = \frac{A^{y+3} - A^{y+1}}{A-1}$. Thus we get

$$\frac{EOS_x}{LB_x} \leq \frac{m}{\epsilon} \frac{A^{y+3} - A^{y+1}}{A^{y+1}} \leq \frac{m}{\epsilon} \frac{A^{y+3}}{A^{y+1}} = \frac{m}{\epsilon} A^2 = \frac{m}{\epsilon}\left(1 + \frac{m}{\epsilon^3}\right)^2 = \xi$$

(remember that $A = (1 + \frac{\alpha}{\epsilon}) \geq 2$ and $\alpha = \frac{m}{\epsilon^2}$).

If the"end of schedule" (EOS_x) that we guess was not correct, due to Lemma 11 all jobs released in interval I_x and all previous intervals would have been irrelevant. □

From the proof of above Lemma 12 the Property 4 holds.

Now, let us suppose that in some interval I_x the total processing time of jobs available for processing (already released and not processed yet) respect the lower bound, $LB_x = \frac{|I_x|}{m}$. In the next Lemma we prove that with $(1 + \mathcal{O}(\epsilon))$ loss and with respect to EOS_x the number of distinct instances of jobs released in the future is finite. Thus we fulfill Property 3.

Lemma 13. *At the beginning of any interval I_x, when we take a scheduling decision, with $(1 + \mathcal{O}(\epsilon))$ loss we can assume that there is only finite number of distinct instances of relevant jobs that might be released in the future, unless any decision in interval I_x cannot much affect the optimal solution.*

Proof. Let us assume that jobs released in interval I_x are relevant. From Lemma 12 there exist LB_x and EOS_x, such that $EOS_x = \xi LB_x$. From Lemma 8 we get that any instance of jobs released in the future can be partitioned into finite number of subsets of jobs, such that each subset contains jobs released at the same time. To follow the claim it is enough to show that number of distinct jobs in each subset is finite. Now, using Lemma 2 with $(1 + \mathcal{O}(\epsilon))$ loss we can reduce any subset of jobs released at some time to some finite number Δ. From Lemma 7 we know that every job's scale factor can take only finite number of distinct values, combining with Lemma 5 every job can take at most finite number of distinct processing times. Finally from Property 3 for fixed processing time and fixed release date there is a finite number of distinct jobs. Since every component is finite, we can produce at most finite number of distinct instances of jobs released in the future such that they respect the EOS_x. □

4 Job Shop and Single Machine Scheduling with Delivery Times

In this section we show that two other scheduling problems described in this paper (Job Shop, Single Machine with delivery times) fulfill Property 1, Property 2, Property 3.

4.1 Online Job Shop Scheduling Problem

In this section we use the results in [4]. We show how easy it is to apply above framework using result for offline scheduling problem. Based on this paper [4] we can rewrite that any job can be fully described using *job profile*, *scale factor* and *release date*. Since the objective function is the same as for the previous problem, C_{max}, nearly all presented lemmas fit the on-line Job Shop scheduling problem. The Properties 1, 3 and 4 are fully satisfied for any online scheduling problem with maximum completion time objective function. To fulfill Property 2

we need to reprove only Lemma 2 and 5. We replace Lemma 2 with the following result.

Lemma 14 ([4]). *Assuming that* $\frac{EOS_x}{LB_x} = \xi$, $\xi = \frac{1}{\epsilon}\left(1+\frac{1}{\epsilon^3}\right)^2$*-finite, then with* $(1+\mathcal{O}(\epsilon))LB$ *loss we can reduce in linear time the number of jobs that are available for processing at time* R_x *to be at most*

$$\Delta = \left(\frac{6\mu^4 m\xi}{\epsilon}\right)^{m/\epsilon} + m^{\mu}\left(2+\log_{1+\epsilon}\frac{\mu m}{\epsilon}\right)^{\mu}$$

The Lemma 5 can be replaced with the following cited result.

Lemma 15 ([4]). *The number of distinct job profiles is bounded from above by most* $l := m^{\mu}\left(2+\log_{1+\epsilon}\frac{\mu m}{\epsilon}\right)^{\mu}$.

4.2 Online Single Scheduling Problem with Delivery Times

We refer to article [14]. Again any job can be fully described using *job profile*, *scale factor* and *release date*. In this problem the case is to minimize the maximum delivery time, L_{max}. Since this objective function in the structure is very similar to maximum completion time, C_{max} the Property 1, 3 and 4 are satisfied for any problem with L_{max}. Similar to the previous subsection we cite the following lemmas to fulfill Property 2.

Lemma 16 ([14]). *Assuming that* $\frac{EOS_x}{LB_x} = \xi$, $\xi = \frac{1}{\epsilon}\left(1+\frac{1}{\epsilon^3}\right)^2$*-const, then with* $(1+\mathcal{O}(\epsilon))LB$ *loss we can reduce in linear time the number of jobs that are available for processing at time* R_x *to be at most* $\Delta = \mathcal{O}(\xi/\epsilon^3)$ *-const.*

Lemma 17 ([14]). *The number of distinct profiles is at most*

$$l := \lceil 12\epsilon^{-2}\log_{1+\epsilon} 2/\epsilon \rceil.$$

4.3 Constant Number of Machines

Since through all the paper we use the assumption that an on-line algorithm has unbounded computational power we only care about finite values, i.e. instance size and so on. If the number of machines in the input problem is constant then any finite number in the paper becomes constant number. Thus for considered problems with constant number of machines and operations (for Job Shop), $Rm|r_j|C_{max}$, $Jm|r_j|C_{max}$, $1|r_j|L_{max}$ the algorithm $\mathcal{A}$ presented in the Section 2 has constant running time.

References

1. Afrati, F.N., et al.: Approximation schemes for minimizing average weighted completion time with release dates. In: FOCS, pp. 32–44. IEEE Computer Society (1999)

2. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)
3. Chen, B., Vestjens, A.P.A.: Scheduling on identical machines: How good is lpt in an on-line setting? Oper. Res. Lett. 21(4), 165–169 (1997)
4. Fishkin, A.V., Jansen, K., Mastrolilli, M.: Grouping techniques for scheduling problems: Simpler and faster. Algorithmica 51(2), 183–199 (2008)
5. Goldberg, L.A., Paterson, M., Srinivasan, A., Sweedyk, E.: Better approximation guarantees for job-shop scheduling. SIAM J. Discret. Math. 14(1), 67–92 (2001)
6. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell System Technical Journal 45(4), 1563–1581 (1966)
7. Günther, E., Maurer, O., Megow, N., Wiese, A.: A new approach to online scheduling: Approximating the optimal competitive ratio. CoRR abs/1204.0897 (2012)
8. Hall, L.A., Shmoys, D.B.: Approximation schemes for constrained scheduling problems. In: FOCS, pp. 134–139. IEEE Computer Society (1989)
9. Hall, L.A., Shmoys, D.B.: Jackson's rule for single-machine scheduling: making a good heuristic better. Math. Oper. Res. 17(1), 22–35 (1992)
10. Rudin III, J.F., Chandrasekaran, R.: Improved bounds for the online scheduling problem. SIAM J. Comput. 32(3), 717–735 (2003)
11. Jansen, K., Solis-Oba, R., Sviridenko, M.: Makespan minimization in job shops: A polynomial time approximation scheme. In: Vitter, J.S., Larmore, L.L., Leighton, F.T. (eds.) STOC, pp. 394–399. ACM (1999)
12. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica 3, 77–119 (1988)
13. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. 46, 259–271 (1990)
14. Mastrolilli, M.: Efficient approximation schemes for scheduling problems with release dates and delivery times. J. Scheduling 6(6), 521–531 (2003)
15. Mastrolilli, M., Svensson, O.: Hardness of approximating flow and job shop scheduling problems. J. ACM 58(5), 20 (2011)
16. Paterson, M. (ed.): ESA 2000. LNCS, vol. 1879. Springer, Heidelberg (2000)
17. Shmoys, D.B., Stein, C., Wein, J.: Improved approximation algorithms for shop scheduling problems. SIAM J. Comput. 23(3), 617–632 (1994)
18. Shmoys, D.B., Wein, J., Williamson, D.P.: Scheduling parallel machines on-line. SIAM J. Comput. 24(6), 1313–1331 (1995)
19. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28(2), 202–208 (1985)
20. Williamson, D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevast'janov, S.V., Shmoys, D.B.: Short Shop Schedules. Operations Research 45, 288–294 (1997)

Online Primal-Dual for Non-linear Optimization with Applications to Speed Scaling

Anupam Gupta[1,⋆], Ravishankar Krishnaswamy[2], and Kirk Pruhs[3,⋆⋆]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213
[2] Computer Science Department, Princeton University, Princeton, NJ 08542
[3] Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260

Abstract. We give a principled method to design online algorithms (for potentially non-linear problems) using a mathematical programming formulation of the problem, and also to analyze the competitiveness of the resulting algorithm using the dual program. This method can be viewed as an extension of the online primal-dual method for linear programming problems, to nonlinear programs. We show the application of this method to two online speed-scaling problems: one involving scheduling jobs on a speed scalable processor so as to minimize energy plus an arbitrary sum scheduling objective, and one involving routing virtual circuit connection requests in a network of speed scalable routers so as to minimize the aggregate power or energy used by the routers. This analysis shows that competitive algorithms exist for problems that had resisted analysis using the dominant potential function approach in the speed-scaling literature, and provides alternate cleaner analysis for other known results. This gives us another tool in the design and analysis of primal-dual algorithms for online problems.

1 Introduction

Speed scalable devices are now a ubiquitous energy management technology. Such devices can be run in high speed and power modes that are energy inefficient, or in low speed and power modes that are more energy efficient, where energy efficiency is the resulting processing speed divided by power investment. The resulting optimization problems involve determining when this improvement in the quality of service provided by running at high speed justifies the resulting inefficient use of significant energy. As the relationship between speed and power in current (and any conceivable) technologies is non-linear, so are the resulting optimization problems. This non-linearity explains in part why we have generally not been able to show that online algorithms are competitive in such settings by reasoning directly about optimal solutions. The dominant algorithm analysis tool in speed-scaling settings has been potential functions. But one has to often guess the "right" potential function; moreover, there are situations where the use of potential functions is problematic, most notably when there does not seem to be a simple algebraic expression for the "right" potential for an arbitrary configuration.

⋆ Research partly supported by NSF awards CCF-0964474 and CCF-1016799.
⋆⋆ Supported in part by NSF grants CCF-0830558, CCF-1115575, CNS-1253218 and an IBM Faculty Award.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 173–186, 2013.

One motivation of our research is to see if one can use duality to analyze online algorithms for speed-scaling problems for which algorithm analysis using potential functions seems problematic.

The first problem we consider involves scheduling jobs on a speed scalable processor online, with an objective of the form $\mathcal{E} + \beta\mathcal{S}$, where $\mathcal{E}$ is the energy used by the processor, $\mathcal{S}$ is a scheduling objective that is the sum over jobs of a scheduling cost of the individual jobs, and β expresses the relative value of saving energy versus decreasing the scheduling objective. The input consists of a collection of jobs that arrive over time. The j^{th} job arrives at time r_j, and has size/work p_j. There is a convex function $P(s) = s^\alpha$ specifying the dynamic power used by the processor as a function of speed s, which may be any nonnegative real number. The value of α is typically around 3 for CMOS based processors. A *fractional sum scheduling objective* $\mathcal{S}$ is of the form $\sum_j \sum_t \frac{y_{jt}}{p_j} C_{jt}$, where C_{jt} is the cost of completing a unit of work of job j at time t, and y_{jt} is the amount of work completed at time t (such that $\sum_t y_{jt} = p_j$). The corresponding *integer* sum scheduling objective is of the form $\sum_j \sum_t z_{jt} C_{jt}$, where z_{jt} indicates if job j was finished at the time instant t. For instance, a single job arrives at time r_j with $C_{jt} = (t + 1 - r_j)$, and is executed over times $t \in \{r_j, r_j + 1, r_j + 2, \cdots, r_j + p_j - 1\}$ at a uniform rate $y_{jt} = 1$, the fractional sum objective is $\sum_{t=r_j}^{r_j+p_j-1} \frac{1}{p_j}(t - r_j + 1) = \frac{p_j+1}{2}$, whereas the integer sum objective is p_j (since $z_{j(r_j+p_j-1)} = 1$ and all other $z_{jt} = 0$). The energy cost $\mathcal{E}$ in both cases is the non-linear expression $\sum_t (\sum_j y_{jt})^2$.

For linear sum scheduling objectives (where the cost C_{jt} for finishing a job j at time t is a increasing linear function of the flow time $t - r_j$, such as the one above), a potential function based on an algebraic expression for the future cost of the online scheduling algorithm (starting from a particular configuration) has proved to be widely applicable for analyzing natural speed-scaling scheduling algorithms [1]. However, the seeming lack of simple algebraic expressions for future online costs of natural online algorithms for nonlinear sum scheduling objectives (e.g., $C_{jt} = (t - r_j + 1)^2$, which gives the sum of the squares of flow time) explains in part why, despite some effort, we have not been able to analyze algorithms for such problems. It is easy to convert a solution for the fractional scheduling objective to the integer objective while losing at most $\max\{(1 + \epsilon)^\alpha, 1/\epsilon\}$ by speeding up each job by $1 + \epsilon$, so that the fractional objective pays at least ϵ for each instant the sped-up job has not yet been completed, whereas the power cost is increased only by $(1 + \epsilon)^\alpha$[2,3], so we will focus on the fractional scheduling objectives from now on.

The second problem that we consider involves online routing of virtual circuit connection requests in a network of speed scalable routers with the objective of minimizing the aggregate power used by the routers. The j^{th} request consists of a source s_j, a sink t_j, and a flow requirement f_j. In the unsplittable flow version of the problem the online algorithm must route f_j units of flow along a single (s_j, t_j)-path. In the splittable flow version of this problem, the online algorithm may partition the f_j units of flow among some collection of (s_j, t_j)-paths. In either case, we assume speed scalable network elements (routers, or links, or both), where element e use powers ℓ_e^α, where the load ℓ_e is the sum of the flows through the element. The objective of total aggregate power is then $\sum_e \ell_e^\alpha$. This problem was introduced in [4], where a poly-log-approximate *offline*

polynomial-time algorithm is given for unsplittable routing; this algorithm classifies the requests by geometrically increasing demands, and randomly rounds a convex program for each demand class.

2 Our Contributions

Very much in the spirit of the online primal dual technique for linear programs [5], we give a principled method both to design online algorithms (but for potentially nonlinear problems) using a mathematical programming formulation of the problem, and also to analyze the competitiveness of the resulting algorithm using the Lagrangian dual program. We start by considering a mathematical program for the offline problem. We then interpret the online algorithm as solving this mathematical program online, where the constraints arrive one-by-one: in response to the arrival of a new constraint, the online algorithm has to raise some of the primal variables so that the new constraint will be satisfied. We consider the most natural online greedy algorithm: one that raises the primal variables so that the increase in the primal objective is minimized.

For the analysis, we use weak duality: each feasible value of the dual is a lower bound to the optimal primal solution [6]. How should we set the duals? We set the Lagrangian dual variable corresponding to the new constraint to be *proportional to the rate of increase in the primal objective* that the online algorithm incurred at the time that the constraint was satisfied. If we could argue that the value of the dual increased by at least a constant fraction of the increase to the primal, we would be done. But what is the value of the resulting dual? Due to nonlinearity, analyzing the dual for a nonlinear program is more complicated than for a linear program, since in the dual for a nonlinear program, one can not disentangle the objective and the constraints (as one can in the linear case); the dual itself contains a version of the primal objective, and hence copies of the primal variables, within it. Consequently, the arguments for the dual in the nonlinear case not only involve setting the Lagrangian dual variables, but also relating the settings of the copies of the primal variables in the dual with the actual primal variables in the primal. The solutions we find are fractional solutions to the nonlinear program, so for the speed-scaling scheduling (with the fractional objective) and splittable routing problems that we consider, this analysis allows us to conclude that the natural online greedy algorithm is $O_\alpha(1)$-competitive (the subscript means that the constant hidden in the big-O depends on the constant α). As mentioned above, the integer scheduling objective follows from the fractional one at a small loss [2,3]. For the unsplittable routing problem, we can also show that the natural online greedy algorithm is $O_\alpha(1)$-competitive.

Before we give more details about the specific speed-scaling problems that we consider, we would like to emphasize that once we formulate the primal non-linear program in the obvious way, the design of the online algorithm and the variable settings for its Lagrangian dual are naturally derived from this program. The problem-specific aspects are confined to setting the dual variables—which in both our problems is some multiplier δ times the rate of primal change—and the analysis of the dual (which gives the

"right" value of the multiplier δ we should set).[1] Consequently, we feel that our work represents another step towards a principled design and analysis of primal-dual algorithms for online problems. We hope that this principled approach could be applied to a wider class of nonlinear online problems.

2.1 Applications to Speed-Scaling Scheduling and Routing

Our first observation is that fractional speed-scaling scheduling problems with a general sum scheduling objective can be cast more conveniently as the following *Online Generalized Assignment Problem (OnGAP)*. In OnGAP, jobs arrive online one-by-one, and the algorithm must fractionally assign these jobs to one of m machines. When a job j arrives, the online algorithm learns ℓ_{je}, the amount by which the load of machine e would increase for *each unit* of work of job j that is assigned to machine e, and c_{je}, the assignment cost incurred for each unit of work of job j that is assigned to machine e. So if x_{je} is the fraction of job j assigned to machine e, then the assignment cost is $c_{je}x_{je}$, and the load of the machine e increases by $\ell_{je}x_{je}$. The goal is to minimize the sum of the α^{th} powers of the machine loads, plus the total assignment cost. (See (4.1) for the convex programming formulation.)

To cast a speed-scaling problem with a (fractional) sum scheduling objective as OnGAP, think of each unit of time as being a separate machine. The assignment cost c_{je} then models the cost for scheduling a unit of job j at time e. Let us now illustrate this model using some examples:

Speed-Scaling with Deadline Feasibility. In this problem, each job j has a size of ℓ_j and a deadline of d_j. The goal is to devise speed-scaling and scheduling policies so that every job is scheduled within its deadline, and the total energy is minimized. Indeed, we can view this as OnGAP where each time unit is a machine, $\ell_{jt} := \ell_j$ for all times, and $c_{jt} := 0$ for $t \in [r_j, d_j]$ and is infinite otherwise. This problem has been widely studied [7,8,9,10], and different algorithms are shown to be $O_\alpha(1)$-competitive using varied potential functions.

Total Flow plus Energy. Here, given a set of jobs and a speed-scalable processor, the objective is to minimize the sum of (fractional) flowtimes plus energy of the schedule. We can cast this as OnGAP by setting c_{jt} to be $(t - r_j)$ for $t \geq r_j$ and infinite otherwise. This problem has been studied in [11,3,12,13,14,15,16], where different algorithms are shown to be $O_\alpha(1)$-competitive (and some, even $O(1)$-competitive for general power functions) using potential functions.

Total Flow Squared plus Energy. For the objective of sum of fractional flow/response times squared plus energy, we can set the assignment cost c_{je} to be $(e - r_j)^2$ for all times $e \geq r_j$, the release time of job j, and infinite otherwise.

More General Objectives. We can create much more exotic objectives: say, $c_{je} = (e - r_j)^2$ for $j \in [r_j, d_j]$ and ∞ otherwise would give squared flow time with a hard deadline, or we could create blackout dates by setting some c_{je}'s to ∞. For many of these general problems, we provide the first online algorithms with non-trivial competitive ratios.

[1] In the sequel, whenever we use the word dual, we refer to the Lagrangian dual of the primal convex program being considered.

In Section 4 we apply our primal-dual approach to solving OnGAP *fractionally*, and show that a natural greedy algorithm is α^α-competitive.[2] This immediately gives us solutions for speed scaling scheduling problems with fractional sum objectives. To analyze our algorithm for OnGAP, we show a dual solution has a particularly nice form, and for our setting of the dual variables, is within α^α of the primal solution. As an immediate consequence, the corresponding online greedy algorithm is α^α-competitive for speed-scaling scheduling problems with *any* fractional sum scheduling objective. Recall that previously, competitive analyses were known only for linear sum scheduling objectives, so this is a substantial improvement. For some speed-scaling problems, our duality analysis is cleaner than the existing potential function analyses; e.g., compare the α^α-competitive analysis of the greedy Optimal Available (OA) algorithm for energy minimization with deadline feasibility constraints given in [8] to our α^α-competitive analysis of our greedy algorithm. Lower bounds in [17,18,8] imply that no deterministic online algorithm can be better than α^α-competitive for OnGAP, and no deterministic online algorithm can be better than α^α-competitive for speed scaling scheduling to minimize energy with feasibility constraints. In Section 5, we make some further comments about the application of these results to speed-scaling problems.

Finally in Section 6, we apply our primal-dual approach to the splittable routing problem in a network of speed scalable routers. In this case, the worst-case settings of the copies of the primal variables in the dual are not easy to reason about. To facilitate this reasoning, we *relax* the dual problem in a novel way, by allowing the copies of the primal variables in the dual to take on different values for different edges. To overcome relaxing the flow-constraints, we alter the relaxed objective function (based on the edge loads of our online algorithm!) to ensure that we can still recover enough dual value. This allows us to show the online greedy algorithm is α^α-competitive with respect to the relaxed dual with the specified settings of the dual variables, and hence with respect to optimal. This extends to unsplittable routings as well.

3 Related Work

An extensive survey/tutorial on the online primal dual technique for linear problems can be found in [5]. A survey of the algorithmic power management literature in general, and the speed-scaling literature in particular, can be found in [19]. Casting the speed-scaling scheduling problems as a load balancing problem is natural in hindsight, but to the best of our knowledge this has not been observed before. This reduction allows the application of techniques from the load balancing literature to speed-scaling problems. The version of OnGAP without assignment costs was studied by [17,18], where the online greedy algorithm is shown to be $O_\alpha(1)$-competitive. In their analysis the online cost is bounded by an algebraic expression involving the product of the online cost and the optimal cost, which is disentangled by use of the Cauchy-Schwartz inequality. While this analysis shares some commonalities with both potential function analysis

[2] In the full version, we show a related greedy algorithm is $O(\alpha)^\alpha$-competitive for OnGAP *integrally* (where each job has to go to a *single* machine). This does not imply anything useful for the speed-scaling scheduling problems. On the other hand, one can use the underlying ideas to convert the splittable energy-aware routing algorithm of Section 6 to unsplittable routings.

and duality analysis, it is probably best considered a distinct technique. Upon some reflection, one can see that their potential function technique can be used to obtain an alternate analysis that achieves the same bounds as we achieve in this paper by duality. Caragiannis [20] gives some refinements to the analysis in [17,18] for OnGAP without assignment costs. An offline $O(1)$-approximation (independent of α) was given by [21,22], via solving the convex program and rounding the solution in a correlated fashion: such an result independent of α is impossible in the online setting. Finally, offline poly-log-approximation algorithms for the virtual circuit routing problem, when routers have a static power component, can be found in [23,4].

Works [24,25] show that various online algorithms are competitive, using potential function analysis, for various scheduling problems with fixed speed processors and for the ℓ_k norms of flow objective. The potential functions used in these analyses were motivated by the desire to have an algebraic expression for the future costs for the online algorithm, but required some ad-hoc features in order for the algebra to work out. Despite efforts by the authors of these papers, it is not clear how to extend these potential functions to work in a speed-scaling setting.

Independently and concurrently with this work, Anand, Garg and Kumar [26] obtained results in a similar vein to the results here. Mostly notably, they showed how to use nonlinear-duality to analyze a greedy algorithm for a multiprocessor speed-scaling problem involving minimizing flow plus energy on unrelated machines. Additionally, [26] showed how duality based analyses could be given for several scheduling algorithms that were analyzed in the literature using potential functions. However, our results are somewhat different in spirit, with our emphasis being more on a principled methodology for algorithm design and setting of the dual variables. For instance the algorithm for the speed-scaling problem in [26] is not derived from the mathematical programming formulation, and the emphasis is more on obtaining a "dual-fitting" analysis for (in some sense) pre-existing algorithms.

4 The Online Generalized Assignment Problem

In this section we consider the problem of Online Generalized Assignment Problem (OnGAP). If x_{je} denotes the extent to which job j is assigned on machine e, then this problem can be expressed by the following mathematical program:

$$\begin{aligned} \min \quad & \sum_e \left(\sum_j \ell_{je} x_{je} \right)^\alpha + \sum_e \sum_j c_{je} x_{je} \\ \text{subject to} \quad & \sum_e x_{je} \geq 1 \qquad j = 1, \ldots, n \end{aligned} \tag{4.1}$$

The dual of the primal relaxation is then

$$g(\lambda) = \min_{x \geq 0} \left(\sum_j \lambda_j + \sum_e \left(\sum_j \ell_{je} x_{je} \right)^\alpha + \sum_{j,e} c_{je} x_{je} - \sum_{j,e} \lambda_j \, x_{je} \right) \tag{4.2}$$

One can think of the dual problem as having the same instance as the primal, but where jobs are allowed to be assigned to extents less than unit. This is compensated for in the

objective function: in addition to the load cost $\sum_e \left(\sum_j \ell_{je} x_{je}\right)^\alpha$ as in the primal, a fixed cost of λ_j is paid for each job j, and a *payment* (or negative cost) of $\lambda_j - c_{je}$ is obtained for each unit of job j assigned. It is well known that each feasible value of the dual is a lower bound to the optimal primal solution; this is *weak duality* [6].

Online Greedy Algorithm Description: Let δ be a constant that we will later set to $\frac{1}{\alpha^{\alpha-1}}$. Now the algorithm works as follows when a new job j arrives: until a unit fraction is scheduled, job j is scheduled on all machines for which the increase in the cost will be the least, assuming that energy costs are discounted by a factor of δ. More formally, the value of all the primal variables x_{je} for all the machines e that minimize

$$\delta \cdot \alpha \cdot \ell_{je} (\sum_{i \le j} \ell_{ie} x_{ie})^{\alpha-1} + c_{je} \tag{4.3}$$

are increased until all the work from job j is scheduled, i.e., $\sum_e x_{je} = 1$. Notice that $\alpha \cdot \ell_{je} \left(\sum_{i \le j} \ell_{ie} x_{ie}\right)^{\alpha-1}$ is the rate at which the load cost is increasing for machine e, and c_{je} is the rate that assignment costs are increasing for machine e. In other words, our algorithm fractionally assigns the job to the machines on which the overall objective increases at the least rate. Furthermore, observe that if the algorithm begins assigning the job to some machine e, it does not stop raising the primal variable x_{je} until the job is fully assigned[3]. By this monotonicity property, it is clear that all machines e for which $x_{je} > 0$ have the same value of the above derivative when j is fully assigned. Now, for the purpose of analysis, we set the value $\widehat{\lambda}_j$ to be the rate of increase of the objective value when we assigned the last infinitesimal portion of job j. More formally, if e is any machine on which job j is run, i.e., if $x_{je} > 0$, then

$$\widehat{\lambda}_j := \delta \cdot \alpha \cdot \ell_{je} \left(\sum_{i \le j} \ell_{ie} x_{ie} \right)^{\alpha-1} + c_{je} \tag{4.4}$$

Intuitively, $\widehat{\lambda}_j$ is a surrogate for the total increase in objective value due to our fractional assignment of job j (we assign a total of 1 unit of job j, and λ_j is set to be the rate at which objective value increases). Let $\widetilde{x}$ denote the final value of the x_{je} variables for the online algorithm.

Algorithm Analysis. To establish the desired competitive ratio of $O(\alpha^\alpha)$, note that it is sufficient (by weak duality) to show that $g(\widehat{\lambda})$ is at least $\frac{1}{\alpha^\alpha}$ times the cost of the online solution. To this end, let $\widehat{x}$ be the value of the minimizing x variables in $g(\widehat{\lambda})$, namely

$$\widehat{x} = \arg\min_{x \succeq 0} \left(\sum_j \widehat{\lambda}_j + \sum_e \left(\sum_j \ell_{je} x_{je} \right)^\alpha - \sum_{j,e} \left(\widehat{\lambda}_j - c_{je} \right) x_{je} \right)$$

Observe that the values $\widehat{x}$ could be very different from the values $\widetilde{x}$, and indeed the next few lemmas try to characterize these values. Lemma 1 notes that $\widehat{x}$ only has one job $\varphi(e)$ on each machine e, and Lemma 2 shows how to determine $\varphi(e)$ and $\widehat{x}_{\varphi(e)e}$. Then, in Lemma 3, we show that a feasible choice for the job $\varphi(e)$ is the latest arriving job for which the online algorithm scheduled some bit of work on machine e; Let us denote this latest job by $\psi(e)$. Formally, $\psi(e) = \max\{j \text{ s.t } \widetilde{x}_{je} > 0\}$.

[3] It may however increase x_{je} and $x_{je'}$ at different rates so as to balance the derivatives where e and e' are both machines which minimize equation 4.3

Lemma 1. *There is a minimizing solution $\widehat{x}$ such that if $\widehat{x}_{je} > 0$, then $\widehat{x}_{ie} = 0$ for $i \neq j$.*

Proof. Suppose for some machine e, there exist distinct jobs i and k such that $\widehat{x}_{ie} > 0$ and $\widehat{x}_{ke} > 0$. Then by the usual argument of either increasing or decreasing these variables along the line that keeps their sum constant, we can keep the convex term $(\sum_j \ell_{je}\widehat{x}_{je})^\alpha$ term fixed and not increase the linear term $\sum_j (\widehat{\lambda}_j - c_{je})\,\widehat{x}_{je}$. This allows us to either set $\widehat{x}_{ie}$ or $\widehat{x}_{ke}$ to zero without increasing the objective. ∎

Lemma 2. *Define $\varphi(e) = \arg\max_j \frac{(\widehat{\lambda}_j - c_{je})}{\ell_{je}}$. Then $\widehat{x}_{\varphi(e)e} = \frac{1}{\ell_{\varphi(e)e}} \left(\frac{\widehat{\lambda}_{\varphi(e)} - c_{\varphi(e)e}}{\alpha \ell_{\varphi(e)e}} \right)^{1/(\alpha-1)}$ and $\widehat{x}_{je} = 0$ for $j \neq \varphi(e)$. Moreover, the contribution of machine e towards $g(\widehat{\lambda})$ is exactly $(1-\alpha)\left(\frac{\widehat{\lambda}_{\varphi(e)} - c_{\varphi(e)e}}{\alpha \ell_{\varphi(e)e}} \right)^{\alpha/(\alpha-1)}$.*

Proof. By Lemma 1 we know that in $\widehat{x}$ there is at most one job (say j, if any) run on machine e. Then the contribution of this machine to the value of $g(\widehat{\lambda})$ is

$$(\ell_{je}\widehat{x}_{je})^\alpha - (\widehat{\lambda}_j - c_{je})\widehat{x}_{je} \tag{4.5}$$

Since $\widehat{x}$ is a minimizer for $g(\widehat{\lambda})$, we know that the partial derivative of the above term evaluates to zero. This gives $\alpha \ell_{je} \cdot (\ell_{je}\widehat{x}_{je})^{\alpha-1} - \left(\widehat{\lambda}_j - c_{je}\right) = 0$, or equivalently, $\widehat{x}_{je} = \frac{1}{\ell_{je}} \left(\frac{\widehat{\lambda}_j - c_{je}}{\alpha \ell_{je}} \right)^{1/(\alpha-1)}$. Substituting into this value of $\widehat{x}_{je}$ into equation (4.5), the contribution of machine e towards the dual $g(\widehat{\lambda})$ is

$$\left(\frac{\widehat{\lambda}_j - c_{je}}{\alpha \ell_{je}} \right)^{\alpha/(\alpha-1)} - \frac{(\widehat{\lambda}_j - c_{je})}{\ell_{je}} \left(\frac{\widehat{\lambda}_j - c_{je}}{\alpha \ell_{je}} \right)^{1/(\alpha-1)} = (1-\alpha) \left(\frac{\widehat{\lambda}_j - c_{je}}{\alpha \ell_{je}} \right)^{\alpha/(\alpha-1)}$$

Hence, for each machine e, we want to choose that the job j that minimizes this expression, which is also the job j that maximizes the expression $(\widehat{\lambda}_j - c_{je})/\ell_{je}$ since $\alpha > 1$. This is precisely the job $\varphi(e)$ and the proof is hence complete. ∎

Lemma 3. *For all machines e, job $\psi(e)$ is a feasible choice for $\varphi(e)$.*

Proof. The line of reasoning is the following:

$$\varphi(e) = \arg\max_j \frac{(\widehat{\lambda}_j - c_{je})}{\ell_{je}} = \arg\max_j \left(\delta \cdot \alpha \cdot \left(\sum_{i \leq j} \ell_{je} x_{ie} \right)^{\alpha-1} \right)$$
$$= \arg\max_j \left(\left(\sum_{i \leq j} \ell_{ie} x_{ie} \right)^{\alpha-1} \right) = \psi(e)\,.$$

The first equality is the definition of $\varphi(e)$. For the second, observe that for any job k,

$$\widehat{\lambda}_k \leq \delta \cdot \alpha \cdot \ell_{ke} (\sum_{i \leq k} \ell_{ie} x_{ie})^{\alpha-1} + c_{ke} \implies \frac{\widehat{\lambda}_k - c_{ke}}{\ell_{ke}} \leq \delta\, \alpha\, (\sum_{i \leq k} \ell_{ie} x_{ie})^{\alpha-1}\,.$$

The above expression is monotone increasing in $\sum_{i \leq k} \ell_{ie} x_{ie}$, the load due to jobs up to and including k. Moreover, it is maximized by the last job assigned fractionally to e. Since the last job is $\psi(e)$, the last equality follows. ∎

Theorem 1. *The online greedy algorithm is α^α-competitive.*

Proof. By weak duality it is sufficient to show that $g(\widehat{\lambda}) \geq \text{ON}/\alpha^\alpha$. Applying Lemma 2 to the expression for $g(\widehat{\lambda})$ (equation (4.2)), and using Lemma 3 to replace $\varphi(e)$ by $\psi(e)$, we get that

$$g(\widehat{\lambda}) = \left(\sum_j \widehat{\lambda}_j + \sum_e (1-\alpha) \left(\frac{\widehat{\lambda}_{\psi(e)} - c_{\psi(e)e}}{\alpha \ell_{\psi(e)e}} \right)^{\alpha/(\alpha-1)} \right) \tag{4.6}$$

Now we consider only the first term $\sum_j \widehat{\lambda}_j$ and evaluate it.

$$\sum_j \widehat{\lambda}_j = \sum_{j,e} \widehat{\lambda}_j \widetilde{x}_{je} \tag{4.7}$$

$$= \sum_e \sum_j \left(\delta \cdot \alpha \cdot \ell_{je} \left(\sum_{i \leq j} \ell_{ie} \widetilde{x}_{ie} \right)^{\alpha-1} + c_{je} \right) \widetilde{x}_{je} \tag{4.8}$$

$$= (\delta \cdot \alpha) \sum_e \sum_j \ell_{je} \widetilde{x}_{je} \left(\sum_{i \leq j} \ell_{ie} \widetilde{x}_{ie} \right)^{\alpha-1} + \sum_{j,e} \widetilde{x}_{je} c_{je} \tag{4.9}$$

$$\geq \delta \sum_e \left(\sum_j \ell_{je} \widetilde{x}_{je} \right)^\alpha + \sum_{j,e} \widetilde{x}_{je} c_{je} \tag{4.10}$$

If we consider the second term of (4.6), and plug in the value of $\widehat{\lambda}_{\psi(e)}$, it evaluates to $(1-\alpha)\delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_j \ell_{je} \widetilde{x}_{je} \right)^\alpha$. Putting the above two estimates together, we get

$$g(\widehat{\lambda}) \geq \delta \sum_e \left(\sum_j \ell_{je} \widetilde{x}_{je} \right)^\alpha + \sum_{j,e} \widetilde{x}_{je} c_{je} + (1-\alpha)\delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_j \ell_{je} \widetilde{x}_{je} \right)^\alpha \tag{4.11}$$

$$= \left(\delta + (1-\alpha)\delta^{\alpha/(\alpha-1)} \right) \sum_e \left(\sum_j \widetilde{x}_{je} \ell_{je} \right)^\alpha + \sum_{j,e} \widetilde{x}_{je} c_{je} \geq \text{ON}/\alpha^\alpha \tag{4.12}$$

The last step is by setting $\delta = 1/\alpha^{\alpha-1}$ which maximizes $\left(\delta + (1-\alpha)\delta^{\alpha/(\alpha-1)} \right)$. ∎

As observed, e.g., in [18], an $O(\alpha)^\alpha$ result is the best possible, even for the (fractional) OnGAP problem without any assignment costs. In the full version, we show how to obtain an $O(\alpha)^\alpha$-competitive algorithm for *integer* solutions to OnGAP by a similar greedy algorithm, and a similar but slightly more careful analysis.

5 Application to Speed Scaling

We now discuss the application of our results for OnGAP to some well-studied speed-scaling problems. Normally one thinks of the online scheduling algorithm as having two components: a *job selection policy* to determine the job to run, and a *speed-scaling policy* to determine the processor speed. However, one gets a different view when one thinks of the online scheduler as solving online the following mathematical programming formulation of the problem (which is an instance of the fractional OnGAP problem):

$$\min \quad \sum_t \left(\sum_j p_j x_{jt} \right)^\alpha + \sum_j \sum_t C_{jt}\, x_{jt}$$

$$\text{subject to} \quad \sum_t x_{jt} \geq 1 \qquad j = 1, \ldots, n$$

Here the variables x_{jt} specify how much work from job j is run at time t, and the objective captures the fractional sum scheduling objective defined in the Introduction. (Think of $y_{jt} := x_{jt} \cdot p_j$.) The arrival of a job j corresponds to the arrival of a constraint specifying that job j must be completed. Greedily raising the primal variables corresponds to committing to complete the work of job j in the cheapest possible way, given the previous commitments. This greedy algorithm has the advantage that, at the release time of a job, it can commit to the client exactly the times that each portion of the job will be run. One can certainly imagine situations when this information would be useful to the client. The speed-scaling algorithms analyzed in the literature for total (possibly weighted) flow scheduling objectives, are some variation of the *balancing* speed-scaling algorithm that sets the power equal to the (fractional) number/weight of unfinished jobs; so for these prior algorithms, when a job is run generally depends on jobs that arrive in the future.

As mentioned earlier, this algorithm for the *fractional* sum objective can be converted to the *integer* scheduling objective by speeding up the processor by a $(1+\epsilon)$ factor and using known techniques [2,3]: the eventual competitive ratio is $\min(\alpha^\alpha(1+\epsilon)^\alpha, \frac{1}{\epsilon})$. One price we pay for the fact that we can handle *any* sum scheduling objective is that our analysis is sub-optimal for specific problems, such as when the scheduling objective is total flow plus energy. Notice that notion of integral solutions for OnGAP do not apply for the integral versions of these energy minimization scheduling problems, since the notions of integrality are different: integrality for OnGAP means each job must be assigned to a single machine (i.e., a single time unit, when we cast OnGAP as an energy minimization scheduling problem), which is different from the concept of integrality for the scheduling objective.

6 Routing with Speed Scalable Routers

Our analysis will follow the same general approach as for OnGAP: we define dual variables $\widehat{\lambda}_j$ for the demand pairs, but now the minimization problem (which is over flow paths, and not just job assignments) is not so straight-forward: the different edges on a path p might want to set $f(p)$ to different values. So we do something seemingly bad: we *relax* the dual to decouple the variables, and allow each (edge, path) pair to choose its own "flow" value $f(p,e)$. And which of these should we use as our surrogate for $f(p)$? We use a convex combination $\sum_{e \in p} h_e \, f(p,e)$—where the multipliers $h(e)$ are chosen *based on the primal loads*(!), hence capturing the importance of edges.

6.1 The Algorithm and Analysis

We first consider the splittable flow version of the problem. Therefore, we can assume without loss of generality that all flow requirements are unit, and all sources and sinks are distinct (so we can associate a unique request $j(p)$ with each path p). This will also

allow us to order paths according to in when flow was sent along the paths. We now model the problem as follows:

$$\begin{aligned} \min \quad & \sum_e \left(\sum_j \sum_{p \ni e : p \in P_j} f(p) \right)^\alpha \\ \text{subject to} \quad & \sum_{p \in P_j} f(p) \geq 1 \qquad j = 1, \ldots, n \end{aligned}$$

where P_j is the set of all (s_j, t_j) paths, and $f(p)$ is a non-negative real variable denoting the amount of flow routed on the path p. In this case, the dual function is:

$$g(\lambda) = \min_{f(p)} \left(\sum_j \lambda_j + \sum_e \left(\sum_j \sum_{p \ni e : p \in P_j} f(p) \right)^\alpha - \sum_{j, p \in P_j} \lambda_j f(p) \right)$$

One can think of the dual as a routing problem with the same instance, but without the constraints that at least a unit of flow must be routed for each request. In the objective, in addition to energy costs, a fixed cost of λ_j is paid for each request j, and a payment of λ_j is received for each unit of flow routed from s_j to t_j.

Description of the Online Greedy Algorithm: For request j, flow is continuously routed along the paths that will increase costs the least until enough flow is routed to satisfy the request. That is, flow is routed along all (s_j, t_j) paths p that minimize $\sum_{e \in p} \alpha \cdot \left(\sum_{q \leq p : q \ni e} f(q) \right)^{\alpha - 1}$. For analysis purposes, after the flow for request j is routed, we define (where δ is a constant later set to $\frac{1}{\alpha^{\alpha-1}}$):

$$\widehat{\lambda}_j = \alpha \delta \left(\sum_{e \in p} \sum_{q \leq p : q \ni e} f(q) \right)^{\alpha - 1}$$

where p is *any* path along which flow for request j was routed.

The Analysis: Unfortunately, unlike the previous section for load balancing, it is not so clear how to compute the dual $g(\widehat{\lambda})$ or its minimizer since the variables cannot be nicely decoupled as we did there (per machine). In order to circumvent this difficulty, we consider the following *relaxed* function $\widehat{g}(\widehat{\lambda}, h)$, which does not enforce the constraint that flow must be routed along paths. This enables us to decouple variables and then argue about the objective value. Indeed, let $\widetilde{f}(p)$ be the final flow on path p for the routing produced by the online algorithm. Let $h(e) = \alpha \sum_{p \ni e} \widetilde{f}(p)^{\alpha - 1}$ be the incremental cost of routing additional flow along edge e, and $h(p) = \sum_{e \in p} h(e)$ be the incremental cost of routing additional flow along path p. We then define:

$$\widehat{g}(\widehat{\lambda}, h) = \min_{f(p,e)} \left(\sum_j \widehat{\lambda}_j + \sum_e \left(\sum_j \sum_{p \ni e : p \in P_j} f(p, e) \right)^\alpha - \sum_j \widehat{\lambda}_j \sum_{P \in P_j} \sum_{e \in p} \frac{h(e)}{h(p)} f(p, e) \right)$$

Conceptually, $f(p, e)$ can be viewed as the load placed on edge e by request $j(p)$. In $\widehat{g}(\widehat{\lambda}, h)$, the scheduler has the option of increasing the load on individual edges $e \in p \in P_j$, but the income from edge e will be a factor of $\frac{h(e)}{h(p)}$ less than the income achieved in $g(\widehat{\lambda})$. In Lemma 4 we prove that $\widehat{g}(\widehat{\lambda}, h)$ is a lower bound for $g(\widehat{\lambda})$. Lemma 5 shows how the minimizer and value of $\widehat{g}(\widehat{\lambda}, h)$ can be computed, and Lemma 6 shows how to bound some of the dual variables in terms of the final online primal solution.

Lemma 4. *For the above setting of $h(\cdot)$, $\widehat{g}(\widehat{\lambda}, h) \leq g(\widehat{\lambda})$.*

Proof. We show that there is a feasible value of $\widehat{g}(\widehat{\lambda}, h)$ that is less than $g(\widehat{\lambda})$. Let the value of $f(p,e)$ in $\widehat{g}(\widehat{\lambda}, h)$ be the same as the value of $f(p)$ in $g(\widehat{\lambda})$. Plugging these values for $f(p,e)$ into the expression for $\widehat{g}(\widehat{\lambda}, h)$, and simplifying, we get:

$$\begin{aligned}\widehat{g}(\widehat{\lambda}, h) &\leq \sum_j \widehat{\lambda}_j + \sum_e \left(\sum_j \sum_{p \ni e : p \in P_j} f(p) \right)^\alpha - \sum_j \widehat{\lambda}_j \sum_{P \in P_j} f(p) \sum_{e \in p} \frac{h(e)}{h(p)} \\ &= \sum_j \widehat{\lambda}_j + \sum_e \left(\sum_j \sum_{p \ni e : p \in P_j} f(p) \right)^\alpha - \sum_j \widehat{\lambda}_j \sum_{P \in P_j} f(p) = g(\widehat{\lambda})\end{aligned}$$

The first equality holds by the definitions of $h(e)$ and $h(p)$, and the second equality holds by the optimality of $f(p)$. ∎

Lemma 5. *There is a minimizer $\widehat{f}$ of $\widehat{g}(\widehat{\lambda}, h)$ s.t for any edge e, there is a single path $p(e)$ such that $\widehat{f}(p,e)$ is positive, and $\widehat{f}(p(e), e) = \left(\frac{\widehat{\lambda}_{j(p(e))} h(e)}{\alpha \cdot h(p(e))} \right)^{1/(\alpha-1)}$.*

Lemma 6. $\widehat{\lambda}_{j(p(e))} \leq \delta \cdot h(p(e))$

Proof. $\widehat{\lambda}_{j(p(e))}$ is δ times the rate at which the energy cost was increasing for the online algorithm when it routed the last bit of flow for request $j(p(e))$. $h(p(e))$ is the rate of at which the energy cost would increase for the online algorithm if additional flow was pushed along $p(e)$ after the last request was satisfied. If $p(e)$ was a path on which the online algorithm routed flow, then the result follows from the fact the online algorithm never decreases the flow on any edge. If $p(e)$ was not a path on which the online algorithm routed flow, then the lemma follows from the fact that, when the online algorithm was routing flow for request $j(p(e))$, $p(e)$ was more costly than the selected paths (and this cost can't decrease subsequently, by the monotonicity of the online algorithm). ∎

Theorem 2. *The online greedy algorithm is α^α competitive.*

Proof. We will show that $\widehat{g}(\widehat{\lambda}, h)$ is at least ON/α^α, which is sufficient since $\widehat{g}(\widehat{\lambda}, h)$ is a lower bound to $g(\widehat{\lambda})$ by Lemma 4, and since $g(\widehat{\lambda})$ is a lower bound to optimal.

$$\widehat{g}(\widehat{\lambda}, h) = \min_{f(p,e)} \left(\sum_j \widehat{\lambda}_j + \sum_e \left(\sum_j \sum_{p \ni e : p \in P_j} f(p,e) \right)^\alpha - \sum_j \widehat{\lambda}_j \sum_{P \in P_j} \sum_{e \in p} \frac{h(e)}{h(p)} f(p,e) \right) \tag{6.13}$$

$$= \sum_j \widehat{\lambda}_j - (\alpha - 1) \sum_e \left(\frac{\widehat{\lambda}_{j(p(e))} h(e)}{\alpha \cdot h(p(e))} \right)^{\alpha/(\alpha-1)} \tag{6.14}$$

$$\geq \sum_j \widehat{\lambda}_j - (\alpha - 1) \sum_e \left(\frac{\delta \cdot h(e)}{\alpha} \right)^{\alpha/(\alpha-1)} \tag{6.15}$$

$$= \sum_j \widehat{\lambda}_j - (\alpha - 1) \delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \tag{6.16}$$

$$= \sum_j \widehat{\lambda}_j \sum_{p \in P_j} \widetilde{f}(p) - (\alpha-1)\delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \tag{6.17}$$

$$= \delta\alpha \sum_j \sum_{p \in P_j} \widetilde{f}(p) \left(\sum_{e \in p} \sum_{q \le p : q \ni e} \widetilde{f}(q) \right)^{\alpha-1} - (\alpha-1)\delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \tag{6.18}$$

$$\geq \delta \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha - (\alpha-1)\delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \tag{6.19}$$

$$= \frac{1}{\alpha^\alpha} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \geq \text{ON}/\alpha^\alpha \tag{6.20}$$

The equality in line (6.13) is the definition of $\widehat{g}(\widehat{\lambda}, h)$. The equality in line (6.14) follows from Lemma 5. The inequality in line (6.15) follows from Lemma 6. The equality in line (6.16) follows from the definition of $h(e)$. The equality in line (6.17) follows from the feasibility of $\widetilde{f}$. The equality in line (6.18) follows from the definition of $\widehat{\lambda}$. The equality in line (6.19) follows from the definition of δ. ■

7 Conclusion

The online primal-dual technique (surveyed in [5]) has proven to be a widely and systematically applicable method to analyze online algorithms for problems expressible by linear programs. This paper develops an analogous technique to analyze online algorithms for problems expressible by *nonlinear* programs. The main difference is that in the nonlinear setting one can not disentangle the objective and the constraints in the dual, and hence the arguments for the dual have a somewhat different feel to them than in the linear setting. We apply this technique to several natural nonlinear covering problems, most notably obtaining competitive analysis for greedy algorithms for uniprocessor speed-scaling problems with arbitrary sum scheduling objectives that researchers were not previously able to analyze using the prevailing potential function based analysis techniques.

References

1. Im, S., Moseley, B., Pruhs, K.: A tutorial on amortized local competitiveness in online scheduling. SIGACT News 42(2), 83–97 (2011)
2. Becchetti, L., Leonardi, S., Marchetti-Spaccamela, A., Pruhs, K.: Online weighted flow time and deadline scheduling. Journal of Discrete Algorithms 4(3), 339–352 (2006)
3. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. SIAM J. Comput. 39(4), 1294–1308 (2009)
4. Andrews, M., Antonakopoulos, S., Zhang, L.: Energy-aware scheduling algorithms for network stability. In: INFOCOM, pp. 1359–1367 (2011)
5. Buchbinder, N., Naor, J.S.: The design of competitive online algorithms via a primal-dual approach. Foundations and Trends in Theoretical Computer Science 3(2-3), 93–263 (2009)
6. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, New York (2004)

7. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. IEEE Symposium on Foundations of Computer Science, pp. 374–382 (1995)
8. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. Journal of the ACM 54(1) (2007)
9. Bansal, N., Bunde, D.P., Chan, H.L., Pruhs, K.: Average rate speed scaling. Algorithmica 60(4), 877–889 (2011)
10. Bansal, N., Chan, H.-L., Pruhs, K., Katz, D.: Improved bounds for speed scaling in devices obeying the cube-root rule. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 144–155. Springer, Heidelberg (2009)
11. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms 3(4), 49 (2007)
12. Bansal, N., Chan, H.L., Pruhs, K.: Speed scaling with an arbitrary power function. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 693–701 (2009)
13. Lam, T., Lee, L., To, I., Wong, P.: Speed scaling functions based for flow time scheduling based on active job count. In: European Symposium on Algorithms, pp. 647–659 (2008)
14. Andrew, L.L.H., Lin, M., Wierman, A.: Optimality, fairness, and robustness in speed scaling designs. In: SIGMETRICS, pp. 37–48 (2010)
15. Chan, H.L., Edmonds, J., Lam, T.W., Lee, L.K., Marchetti-Spaccamela, A., Pruhs, K.: Nonclairvoyant speed scaling for flow and energy. In: Symposium on Theoretical Aspects of Computer Science, pp. 255–264 (2009)
16. Chan, S.H., Lam, T.W., Lee, L.K.: Non-clairvoyant speed scaling for weighted flow time. In: European Symposium on Algorithms, pp. 23–35 (2010)
17. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. Journal of the ACM 44(3), 486–504 (1997)
18. Awerbuch, B., Azar, Y., Grove, E.F., Kao, M.Y., Krishnan, P., Vitter, J.S.: Load balancing in the L_p norm. In: IEEE Symposium on Foundations of Computer Science, pp. 383–391 (1995)
19. Albers, S.: Energy-efficient algorithms. Communications of the ACM 53(5), 86–96 (2010)
20. Caragiannis, I.: Better bounds for online load balancing on unrelated machines. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 972–981. ACM, New York (2008)
21. Azar, Y., Epstein, A.: Convex programming for scheduling unrelated parallel machines. In: ACM Symposium on Theory of Computing, pp. 331–337. ACM, New York (2005)
22. Anil Kumar, V.S., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: A unified approach to scheduling on unrelated parallel machines. Journal of the ACM 56(5), Art. 28, 31 (2009)
23. Andrews, M., Antonakopoulos, S., Zhang, L.: Minimum-cost network design with (dis)economies of scale. In: IEEE Symposium on Foundations of Computer Science, pp. 585–592 (2010)
24. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling jobs with varying parallelizability to reduce variance. In: ACM Symposium on Parallelism in Algorithms and Architectures, pp. 11–20 (2010)
25. Im, S., Moseley, B.: Online scalable algorithm for minimizing lk-norms of weighted flow time on unrelated machines. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 95–108 (2011)
26. Anand, S., Garg, N., Kumar, A.: Resource augmentation for weighted flow-time explained by dual fitting. In: ACM-SIAM Symposium on Discrete Algorithms (2012)

Approximating the Throughput by Coolest First Scheduling *

Christoph Dürr[1], Ioannis Milis[2], Julien Robert[3], and Georgios Zois[1,2]

[1] LIP6, Université Pierre et Marie Curie, Paris, France
{Christoph.Durr,Georgios.Zois}@lip6.fr
[2] Dept. of Informatics, Athens University of Economics and Business, Greece
milis@aueb.gr
[3] LIFO, Université d'Orléans, France
julien.robert@univ-orleans.fr

Abstract. We study a temperature-aware scheduling problem aiming in maximizing the throughput of a set of unit-length jobs, each one having its own heat contribution, on a single processor operating under a strict temperature threshold. Following a simplified model for the processor's thermal behavior, proposed by Chrobak et al. [9], we analyze the approximation factor of the natural CoolestFirst scheduling algorithm for jobs with common release dates and deadlines. We first prove a $\frac{k}{k+1}$ factor, where k depends on a partition of the jobs according to their heat contributions. Next, we refine our partition and provide a linear program that shows a lower bound of 0.72 on the approximation factor.

1 Introduction

Motivated by technical, financial and ecological incentives the issue of power and thermal management in computing systems are nowadays a critical research topic not only in the hardware and systems design, but also in the operating systems and applications levels. Focusing on the operating systems level, these issues are addressed as new challenging scheduling problems where the goal is to optimize the energy consumption or/and the processors temperature simultaneously with some other QoS measure (e.g., makespan, throughput, response time). A large body of work on these dual objective scheduling problems is based on the *speed scaling* technology, incorporated in today processors, combined with *power down* strategies. The reader is referred to [1,2,10] for excellent reviews of this work.

In this paper we focus on temperature-aware scheduling problems that aim to model the thermal and cooling behavior of processors. The temperature related objective of these problems is to keep the processors' temperature low by either minimizing its maximum or by avoiding to exceed a given threshold; the violation

* This work was partially supported by the Basic Research Funding Program of the Athens University of Economics and Business (C. Dürr, I. Milis, G. Zois) and by the European Social Fund and Greek national resources under the programs HERACLEITUS II (G. Zois) and THALES-ALGONOW (I. Milis).

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 187–200, 2013.

of such a threshold reduces the lifetime or even damages the processors. An approach to this direction is based on the Newton's law of cooling and uses speed scaling to decrease the processors temperature [5,3]. In another approach [11] the behavior of a processor is modeled as a thermal RC circuit. Here we study the simplified model proposed by Chrobak et al. in [9].

In fact, we consider a set of unit-length jobs, each one having its own heat contribution, to be scheduled on a single processor operating under a strict temperature threshold. Following [9] the temperature of a processor after executing a job becomes equal to the average of its temperature T, when started executing the current job, and the heat contribution h of this job, that is $\frac{T+h}{2}$, where 2 is the processor's cooling factor.

More general models have been studied by Birks et al., with different cooling factors and different processing times or objectives [6,8,7]. Recently, Bampis et al. [4] proposed approximation results for the multi-processor makespan minimization problem, under the same model, as well as for the problem of minimizing the maximum temperature when the threshold constraint is removed.

Since the application we have in mind is the job scheduler at the operating system level, it seems important to keep the computational overhead for the scheduler low, since it could deteriorate the performance and generate additional heating. Therefore we are particular interested in simple algorithms, which at every time slot, schedule the job with highest priority among the jobs available for execution. The priority could depend on the heat contribution of the job, as well as on its deadline.

In [9] the authors study the problem of maximizing the throughput, that is the number of jobs finishing their execution before their deadlines. They prove that it is strongly NP-hard, even for jobs with common release dates and deadlines. Moreover, for the online case with arbitrary release dates and deadlines, they prove that a large class of algorithms are 1/2-competitive and that no deterministic online algorithm achieves a better factor. This class consists of all algorithms that would never schedule a job j, when another job is available which has smaller deadline or heat contribution than j. This class includes for example the CoolestFirst and EarliestDeadlineFirst algorithms.

This paper concerns the maximization of the throughput in the offline setting. For this purpose we consider a simple model, where all jobs are available at time zero and have a common deadline. For this setting the above mentioned class of algorithms reduces to the unique CoolestFirst algorithm. Its behavior is quite simple: at any time slot, if the current temperature is cool enough to allow a job to be scheduled, then it schedules the one with the smallest heat contribution — the *coolest job* — otherwise the processor remains idle in that slot. As the goal is to maximize the number of jobs before the common deadline, it seems a fairly reasonable strategy to focus on the coolest jobs. However, the algorithm schedules these jobs in order of increasing heat contribution, while an optimal schedule might follow a different order, allowing a bigger number of jobs to be executed.

We analyze the approximation factor of CoolestFirst using a rounding scheme. The heat contribution scale is divided into classes and the heat contribution of each is rounded according to these classes, to make it harder for the algorithm and more easy for the optimum schedule. The main advantage of this technique is that the rounded instances contain now only a finite number of different jobs, and permits to describe the optimum schedule. The main contribution of this paper is not this rounding procedure, which is rather standard, but the technical lemmas behind it.

On a rounded instance the schedule produced by CoolestFirst will be partitioned according to the heat contributions of the jobs. Now, in these time intervals the schedule consists only of jobs of some heat contribution — say h — and some idle slots. Clearly, for the analysis we are interested in the proportion ρ of non-idle slots among the time interval and its relation to h. Our main contribution is a theorem stating roughly that for every proportion ρ, there is a heat contribution h_ρ such that CoolestFirst produces a schedule with density ρ when the instance consists only of jobs with heat contribution h_ρ. In addition, we show that the values h_ρ are increasing with ρ, as one would expect.

The paper is organized as follows. After a formal introduction of the notations used in the paper we propose, in Section 3, a first analysis of the CoolestFirst algorithm using a rough rounding scheme, just to explain the general technique. Then, in Section 4, we refine the analysis using a linear programming approach, and finally, in Section 5, we show our key lemma relating densities to heat contributions.

2 Notation and Preliminaries

We consider a set $J = \{1, 2, \ldots, n\}$ of n unit-length jobs to be scheduled in a single processor, each one having a heat contribution $h_j \in \mathbb{Q}^+$. All jobs are considered to be released at time 0 and have a common deadline D. Jobs are executed in some time interval of the form $[t-1, t)$, which we call the *time slot* t, for some positive integer t.

The processor's thermal behavior obeys the following rule: At time 0 its temperature is T_0; when a job j is executed in time slot t, the processor's temperature at time t is equal to $T_t = \frac{T_{t-1}+h_j}{2}$, where 2 is the processor's cooling factor and T_{t-1} its temperature at time $t-1$. The processor's temperature is not allowed to exceed a given thermal threshold, which we assume to be 1 by normalization. Therefore, w.l.o.g. we assume that the heat contribution of each job belongs to the interval $[0, 2]$. This means that at some time slot t, we can schedule only jobs of heat contribution h such that $(T_{t-1} + h)/2 \leq 1$. Idle slots can be treated as executing jobs of heat contribution 0, that is, after an idle slot the temperature is divided by 2. For the sake of simplicity, we refer to a job of heat contribution $0 \leq h \leq 2$ as an h-job. Moreover we say that this job is *hot* if $h > 1$ and *cool* if $h \leq 1$.

Our goal is maximize the throughput, which is the number of jobs that complete their execution before their common deadline D and, in fact, we analyze

the approximation factor of the natural CoolestFirst greedy algorithm: at any time schedule the job with the smallest heat contribution, if the current temperature permits it, otherwise remain idle in that time slot.

As already mentioned, in our analysis we will partition the schedule produced by CoolestFirst into time intervals containing only jobs of identical heat contributions, scheduled as soon as it is admissible. Therefore, it is useful to define the infinite schedule $G(h, T)$ to be the schedule of jobs of heat contribution h with initial temperature $T_0 = T$. For notational convenience we describe the schedule $G(h, T)$ as a binary sequence $\omega = (w_0, w_1, \ldots, w_t) \in \{0, 1\}^\star$, $t \in \mathbb{N}$, where $w_t = 0$ if time slot t is idle and $w_t = 1$ otherwise (see Figure 1 for an example). The critical part of our analysis is based on the concept of *density* of a schedule $G(h, T)$, which is the proportion of 1's in the infinite sequence $G(h, T)$.

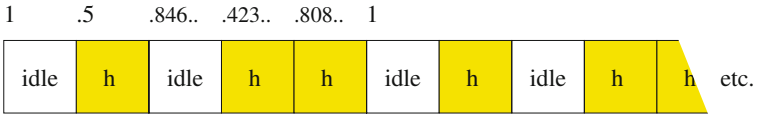


Fig. 1. Example: a prefix of the infinite schedule generated by $h = 31/26$-jobs, obtaining $G(h) = (01011)^*$ for a density of $3/5$

The following proposition analyses the sensibility of the optimal schedule to the initial temperature. According to it we can assume w.l.o.g. that $T_0 = 1$ and we write $G(h)$ as a shortcut for $G(h, 1)$.

Proposition 1. *For the optimum throughput OPT_T when the initial temperature is T, $0 \leq T < 1$, it holds that $OPT_1 \leq OPT_T \leq OPT_1 + 1$.*

Proof. First observe that any schedule which is feasible with some initial temperature is also feasible for any cooler initial temperature. This implies the first inequality.

For the second inequality, let S be a schedule with throughput OPT_T. If S is also feasible when the initial temperature is 1, then we have $\mathrm{OPT}_T = \mathrm{OPT}_1$. Otherwise there is a time t, where S schedules an h-job, which cannot be scheduled with initial temperature 1. Therefore $h > 1$. Let t be minimal and let S' be a schedule that is identical at all time points with S, except that it is idle at t. We claim that S' is feasible when the initial temperature is 1, which implies $\mathrm{OPT}_T \leq \mathrm{OPT}_1 + 1$.

By choice of t, S' is feasible up to time t. Now by $h > 1$, S has a temperature greater than 0.5 at $t + 1$. Since S' is idle at time t, it has a temperature not more than 0.5 at $t + 1$. The first observation from this proof applies again, and implies that S' is feasible from time $t + 1$ on as well. □

3 A First Analysis

In this section we propose a rough lower bound on the approximation factor of the Algorithm CoolestFirst. This is done by performing a rounding procedure, based on a partition of all possible jobs' heat contribution values into intervals that are geometrically decreasing as the heat contribution approaches the hottest job of the instance.

For every $i \in \mathbb{N}$ we define the number $h^i := 2 - 2^{1-i}$ and let $\mathcal{H} := \{h^i : i \in \mathbb{N}\}$. Then, the hot jobs can be divided into classes, where the *i-th job class*, $i \geq 1$ consists of the interval $(h^i, h^{i+1}]$ (see Figure 2). Extending our definition, we call the $[0, 1]$ interval as the 0-th class, consisting of all cool jobs.

Fig. 2. A partition of scheduled jobs into classes

The next lemma describes an optimal schedule for instances with heat contributions from $\mathcal{H}$.

Lemma 1. *Let I be a set of jobs of heat contributions from $\mathcal{H}$. Then, the following steps produce an optimal schedule.*

1. *Run* CoolestFirst *on all jobs from $\mathcal{H}\backslash\{h^0\}$.*
2. *Schedule greedily the 0-jobs in the idle time slots left by the previous step.*

Proof. We prove this by an exchange argument. Fix some optimal schedule. Let $t, t+1$ be two time slots such that the schedule is either idle or executes a 0-job at t and executes an h-job, with $h > 0$, at time $t+1$. Suppose that the temperature at time slot t is cool enough to execute h and then exchange the two time slots. This can lower the temperature at $t+1$ by $h/4$ and will preserve feasibility of the schedule. In such a schedule all h-jobs, having $h > 0$, are scheduled earliest possible, and in an arbitrary order. In particular any h^i-job, for $i \geq 1$, is preceded by exactly $i-1$ time slots, being idle or scheduling a 0-job, and the temperature right after their execution is exactly 1.

Therefore, every scheduled h^i-jobs form a block of i consecutive time slots, and these blocks can be reordered freely, while preserving feasibility of the schedule. This completes the proof. □

The previous lemma permits an output sensitive analysis of the approximation factor of CoolestFirst.

Theorem 1. *Let k be the largest integer such that* CoolestFirst *schedules some hot job from the k-th class on some instance. Then, the approximation factor of* CoolestFirst *on this instance is at least $k/(k+1)$.*

Proof. Let S be the schedule produced by CoolestFirst on the instance. Let n_i be the number of jobs in S from the i-th class with $0 \leq i \leq k$. By this notation the throughput obtained by the algorithm is at most $\sum_{i=0}^{k} n_i$. Note that by definition of the algorithm all jobs not scheduled by CoolestFirst have heat contribution more than h^k.

In order to provide an upper bound for the optimal schedule, we round the jobs down: For $i = 0, \ldots, k$ every job from the i-th class is rounded down to h^i and all remaining jobs are rounded down to h^k. Since by replacing jobs by cooler jobs in a schedule, it preserves its feasibility, this does not decrease the optimal throughput.

What is the optimal throughput of the rounded instance? Since all jobs now belong to $\mathcal{H}$, we can apply Lemma 1. Therefore, the optimal schedule can be produced by first applying CoolestFirst on all hot jobs, resulting in a schedule S' in which later the n_0 0-jobs are filled. Then S' consists of two parts. The first part contains all the jobs scheduled by S and ranges over some interval $[0, v]$, while the second part consists of h^k-jobs, scheduled in the remaining interval $[v + 1, D]$. In order to upper bound the jobs of the second part, we need to bound $D - v$.

The schedule S is partitioned into intervals of the form $[t_i, u_i]$ for every i: it is defined as the interval with minimal t_i and minimal u_i such that it contains exactly all the i-th class jobs scheduled in S and no other job. The last interval of this form might not end at time D, but then it is followed only by idle time slots.

Clearly, $u_i - t_i \leq n_i(i + 1)$. The rounded version of these n_i jobs use at least $n_i i$ time slots in S' including the leading idle time slots. From this we deduce that

$$D - v \leq n_0 + \sum_{i=1}^{k-1} [n_i(i + 1) - n_i i] + n_k(k + 1)$$

Therefore, $D - v \leq \sum_{i=0}^{k} n_i + k n_k$. In the interval $[v + 1, D]$ at most $(D - v)/k$ h^k-jobs are scheduled in S'. Hence, the total number of jobs scheduled in S' is at most

$$\sum_{i=0}^{k} n_i + \frac{1}{k} \sum_{i=0}^{k} n_i = \frac{k + 1}{k} \sum_{i=0}^{k} n_i.$$

This concludes the $\frac{k}{k+1}$-approximation factor of CoolestFirst. □

The above analysis is not tight. Consider the instance consisting of two 0-jobs and two 1.5-jobs with common deadline 4. The optimal schedule contains all four jobs alternating between their heat contributions, while CoolestFirst ends with an idle slot, and therefore has factor 3/4. Theorem 1 gives approximation factor 1/2, creating the need for a refined analysis.

4 A Finer Analysis

To refine the analysis of the previous section, we want to partition the heat contribution scale at heat contributions that are not necessarily from $\mathcal{H}$.

First, note that the output of CoolestFirst for a rounded instance (as the one in Theorem 1) results in a schedule that can be expressed as the concatenation of prefixes of schedules $G(h)$, one for each different h value. In fact, it consists of blocks of jobs with the same heat contribution where every block has some density ρ_h. Recall that a schedule $G(h)$ is formulated as a binary sequence $(w_0, w_1, \ldots, w_{t-1})$, $w_t \in \{0,1\}^\star$, where a job is executed at time $t-1$ if $w_t = 1$, otherwise t is an idle time slot.

As mentioned in the introduction, an important measure of a schedule $G(h)$ is its density, representing the proportion of 1's in the infinite word $G(h)$. The following theorem provides a relation between the density of $G(h)$ and the heat contribution h.

Theorem 2. *For every $\rho \in \mathbb{Q} \cap [0,1]$ there is a heat contribution $h_\rho \in [0,2]$ such that the following property holds: For every integer ℓ, the ℓ-length prefix $\{w_0, w_1, \ldots, w_\ell\}$ of $G(h_\rho)$ satisfies*

$$\lfloor \ell \cdot \rho \rfloor \leq \sum_{t=1}^{\ell} w_t \leq \lceil \ell \cdot \rho \rceil.$$

Moreover for $\rho < \rho'$ we have $h_\rho > h_{\rho'}$.

Before actually proving this theorem, which is done in Section 5, let's see how it can help to improve the analysis of the previous section.

Let $\mathcal{R} = \{\rho_0, \rho_1, \ldots, \rho_l\}$, where $\rho_i \in \mathbb{Q} \cap [0,1]$, $i = 0, 1, \ldots l$, be a set of a constant number of densities with $1 = \rho_0 > \rho_1 \ldots > \rho_l > 0$. These densities partition the interval $[0,1]$. By Theorem 2 the set $\mathcal{R}$ defines a sequence of heat contributions $1 = h_{\rho_0} < \ldots < h_{\rho_l}$. They partition the hot jobs further into the intervals $(h_{\rho_0}, h_{\rho_1}], \ldots, (h_{\rho_{l-1}}, h_{\rho_l}], (h_{\rho_l}, 2]$. Again we want to analyze the approximation factor of CoolestFirst in the case that the algorithm schedules at least some job of heat contribution at least $h_{\rho_{l-1}}$. For an arbitrary instance, let x_i, $0 \leq i \leq l$, be the number of jobs with heat contribution from the interval $(h_{\rho_{i-1}}, h_{\rho_i}]$.

We proceed in a similar manner as in the previous section, but we cannot simply round for every interval its jobs to its lower bound, because we don't know any good upper bound on the number of jobs in the optimal schedule. Instead for every ρ_j, $j = 1, 2, \ldots, l$ there is a rough upper bound, based in the following rounding. Every cool job is rounded to a 0-job, every hot job of heat contribution less or equal than $h_{\rho_{j-1}}$ is rounded to a 1-job, and all the remaining jobs are rounded to $h_{\rho_{j-1}}$-jobs. This permits us to apply the following lemma.

Lemma 2. *Consider an instance where all jobs have a heat contribution $0, 1$ or h and can all be completed before the deadline D. Then, there is an optimal schedule that is produced by the following steps.*

1. *Run* CoolestFirst *on the* 1- *and h-jobs.*
2. *Schedule greedily the* 0*-jobs in the time slots left idle by the previous step.*

The proof uses the same exchange argument used to show Lemma 1 and is omitted.

Now, by using Lemma 2 for the rounded instance, we have the inequality

$$\forall j = 1, 2, \ldots, l-1, l : \sum_{i=1}^{j-1} x_i + \sum_{i=j}^{l} \frac{x_i}{\rho_{j-1}} \leqslant D. \tag{1}$$

With the previous statements in mind we can analyse the performance of CoolestFirst based on a rounding scheme using densities rather than heat contributions.

Theorem 3. *Fix an arbitrary positive integer constant l. Suppose that on some instance, the last job executed by* CoolestFirst *has heat contribution at least h_ρ, for some density $\rho \geq (\sqrt{l}-1)/(l-1)$. Then, the approximation factor of* CoolestFirst *is at least*

$$\frac{l-1}{l} - \frac{l-2}{l}\rho + \frac{l-1}{l}\rho^2$$

up to an additive term of $2l\rho$.

Proof. Let I be an arbitrary instance. In order to lower bound the approximation factor of CoolestFirst we round the jobs to lower density jobs for the algorithm and to higher densitity jobs for the optimal schedule as described before. For this purpose we define the set of densities $\mathcal{R} = \{\rho_0, \rho_1, \ldots, \rho_l\}$ with

$$\rho_i := 1 - \frac{1-\rho}{l} i,$$

for $i = 0, \ldots, l$. We consider the following linear program.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=0}^{l-1} x_i + (D - v)\,\rho \\
\text{subject to} \quad & \sum_{i=0}^{l-1} x_i/\rho_i - v = 0 && \text{(a)} \\
& D - v \geqslant 0 && \text{(b)} \\
& D - \sum_{i=0}^{l} x_i \geqslant 0 && \text{(c)} \\
& D - \sum_{i=1}^{j-1} x_i - \sum_{i=j}^{l} \frac{x_i}{\rho_{j-1}} \geqslant 0 \qquad \forall j = 1, 2 \ldots, l && (y_j) \\
& \sum_{i=0}^{l} x_i = 1 && \text{(e)} \\
& x_0, \ldots, x_l, v, D \geq 0
\end{aligned}$$

The first part of the proof consists in showing that the optimum value of this linear program lower bounds the asymptotic approximation of CoolestFirst.

First we can assume w.l.o.g. that the optimal schedule contains all jobs and only jobs not hotter than h_ρ, and in addition has makespan exactly the deadline.

Let $\bar{D}$ be the deadline of instance I and $\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_l \in \mathbb{N}$ be the number of jobs in I belonging to each of the jobs' intervals defined by $\mathcal{R}$. Namely, $\bar{x}_0$ is the number of cool jobs, while $\bar{x}_i$ is the number of jobs belonging to $(h_{\rho_{i-1}}, h_{\rho_i}]$ for $i = 1, \ldots, l$. Now for the CoolestFirst we round the heat contribution of each job to the higher value of the interval it belongs to. We call $\bar{v}$ the last time the algorithm schedules some job from $[0, h_{\rho_{l-1}}]$ in this rounded instance. Then, by Theorem 2 we have the equality

$$\bar{v} = \bar{x}_0 + \left\lceil \sum_{i=1}^{l-1} \frac{\bar{x}_i}{\rho_i} \right\rceil .$$

and the number of jobs schedule by CoolestFirst are

$$\sum_{i=0}^{l-1} \bar{x}_i + \left\lfloor (\bar{D} - \bar{v})\rho \right\rfloor . \tag{2}$$

Clearly $\bar{v} \leq \bar{D}$. Also since we assumed that the optimum schedule contains all jobs we have

$$\sum_{i=0}^{l} \bar{x}_i \leq \bar{D}.$$

The next step in our proof is to round the jobs for the optimum schedule. For every $j = 1, \ldots, l$ we use the rounding mentioned earlier. So by Lemma 2 we have

$$\sum_{i=1}^{j-1} \bar{x}_i + \left\lceil \sum_{i=j}^{l} \frac{\bar{x}_i}{\rho_{j-1}} \right\rceil \leqslant \bar{D}.$$

Now the approximation factor of CoolestFirst is upper bounded by the factor between (2) and the sum $\sum \bar{x}_i$. Removing the integer roundings in the (in)equalities above could result in a decrease of at most $2l$ of the difference $D - v$. This means that the expression (2) would be decreased by at most $2l\rho$.

The last step in our proof consists in relaxing the integrality constraint of $\bar{x}_0, \ldots, \bar{x}_l, \bar{v}, \bar{D}$, and normalizing the sum $\sum_{i=0}^{l} x_i$ to 1. So let $x_0, \ldots, x_l, v, D$ be the result of dividing the above numbers respectively by $\sum_{i=0}^{l} \bar{x}_i$. Clearly, all the inequalities on the linear program are satisfied by these values, and the objective value lower bounds the approximation factor of CoolestFirst. This concludes the first part of our proof.

It remains to lower bound the objective value of this linear program. This will be done by providing a specific solution to the dual linear program. The dual of the previous linear program is

$$
\begin{aligned}
\text{maximize } & e \\
\text{subject to } & e - c + a \le 1 && (x_0) \\
& e - c + a/\rho_i - \sum_{j=1}^{i} y_j/\rho_{j-1} - \sum_{j=i+1}^{l} y_j \le 1 \quad \forall i = 1, \ldots, l-1 && (x_i) \\
& e - c - \sum_{j=1}^{l} y_j/\rho_{l-1} \le 0 && (x_l) \\
& b + c + \sum_{j=1}^{l} y_j \le \rho && (\mathrm{D}) \\
& b + a \ge \rho && (\mathrm{v}) \\
& y_0, \ldots, y_l, b, c \ge 0,\ e, a \in \mathbb{R}
\end{aligned}
$$

It is easy to verify that the following values provide a solution to the dual linear program, in particular the lower bound on ρ of the statement ensure that $c \ge 0$.

$$
\begin{aligned}
a &= \rho \\
b &= 0 \\
c &= e + \rho - 1 \\
y_j &= 0 && \forall j = 1, \ldots, l-1 \\
y_l &= 1 - e \\
e &= \frac{l-1}{l} - \frac{l-2}{l}\rho + \frac{l-1}{l}\rho^2.
\end{aligned}
$$

This completes the proof of the theorem. □

By using the first derivative of e in ρ, we can show that minimum is obtained at

$$\rho = \frac{l-2}{2l-2}$$

and has value

$$e_{\min} = \frac{3l-4}{4l-4}.$$

For example, for $l = 10$ this would show a lower bound on asymptotic approximation factor of CoolestFirst of $0.722\ldots$. Note that we cannot use the limit of $e_{\min}$ for l to be $+\infty$, in order to provide a bound on the asymptotic approximation factor, because the additive constant is increasing with l.

5 Discrete Lines

In this section we investigate the relation between the density and the heat contribution of a set of h-jobs, aiming to provide a detailed proof of Theorem 2.

The first of the following two procedures, called CoolestFirst(h) for the sake of uniformity, produces a binary sequence for a given value of h. In fact it produces only the part of the sequence that spans between two consecutive temperatures equal to one. For notational convenience, we force CoolestFirst(h) to return the digits of ω in reverse order i.e., $(w_{t-1}, w_{t-2}, \ldots, w_0)$. However, a sequence ω produced by the CoolestFirst(h), seems to be very similar with a sequence that corresponds to the discretization of a line with rational slope and zero offset (see Figure 3).

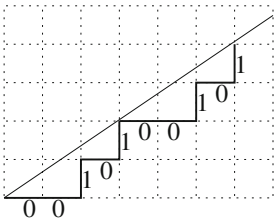


Fig. 3. A discrete line with (slope, offset)= $(\frac{2}{3}, 0)$

The procedure Staircase(p, q), shown below, produces the reverse of such a sequence for a slope equal to $\frac{p}{q-p}$, where p, q are considered to be co-prime integers.

Algorithm. Two procedures that produce a binary sequence

```
1: CoolestFirst(h)
2: T = 1;
3: t = 0;
4: repeat
5:     if (T + h)/2 > 1 then
6:         w_t = 0;
7:         T = T/2
8:     else
9:         w_t = 1;
10:        T = (T + h)/2
11:    t = t + 1
12: until T = 1
13: return ω = (w_{t-1}, w_{t-2}, ..., w_0)
```

```
1: Staircase(p, q)
2: y = 0;
3: t = 0;
4: repeat
5:     if y < p/(q − p) then
6:         a_t = 1;
7:         y = y + 1
8:     else
9:         a_t = 0;
10:        y = y − p/(q − p)
11:    t = t + 1
12: until y = 0
13: return α = (a_0, a_1, ..., a_{t-1})
```

Suppose now that q equals the length of ω, i.e., $T_{q \cdot k} = 1$, $k \in \mathbb{N}$. Then, the density of the schedule produced will be equal to $\rho_h = \sum_t w_t/q$. Let also $int(\omega) = \sum_{t=0}^{q-1} w_t 2^t$ be the decimal expansion of ω.

The following proposition establishes a very interesting (monotone) relation between the heat contribution h, the sequence ω and the density ρ_h of a schedule.

Proposition 2. *For a heat contribution h it holds that $h = \frac{2^q-1}{int(\omega)}$.*

Proof. During the execution of CoolestFirst(h), the digits of the sequence ω are produced in the order $w_0, w_1, \ldots, w_{q-1}$ and span between temperatures $T_0 = 1$ and $T_q = 1$. It holds that $T_q = \frac{T_0 + h\sum_{t=0}^{q-1} w_t 2^t}{2^q}$ and hence, $h = \frac{2^q-1}{int(\omega)}$. □

We shall prove that for a given density $\frac{p}{q}$ the procedure CollestFirst(h), for a heat contribution $h = \frac{2^q-1}{int(\alpha)}$, produces the same sequence with Staircase(p, q), i.e., $\omega = \alpha$. The following proposition summarizes the properties of the sequence α.

Proposition 3. *The sequence α produced by* Staircase(p, q) *starts with $a_0 = 1$, contains exactly p ones, has length equal to q, i.e., $\alpha = (a_0, a_1, \ldots, a_{q-1})$, finishes with $a_{q-1} = 0$ and it is non-periodic.*

Proof. Algorithm Staircase(p, q) starts with $y = 0$ and, hence, $a_0 = 1$. During its execution the value of the variable y is bounded by $0 \leq y < 1 + \frac{p}{q-p}$, that is $0 \leq \frac{y\cdot(q-p)}{q} < 1$. Moreover, each step $t = 0, 1, 2, ...$ of the procedure starts with $y = \ell - \frac{(t-\ell)\cdot p}{q-p} = \frac{\ell\cdot q - t\cdot p}{q-p}$, where ℓ is the number of ones produced so far. Hence, when the t starts we have $\ell = \frac{p\cdot t}{q} + \frac{y\cdot(q-p)}{q}$ and since ℓ is an integer and $\frac{y\cdot(q-p)}{q} < 1$, it holds that $\ell = \left\lceil \frac{p\cdot t}{q} \right\rceil + \left\lfloor \frac{y\cdot(q-p)}{q} \right\rfloor = \left\lceil \frac{p\cdot t}{q} \right\rceil$.

When step $(q-1)$ starts we have $\ell = \left\lceil \frac{p\cdot(q-1)}{q} \right\rceil = p$. Hence, this step starts with $y = \frac{p\cdot q - (q-1)\cdot p}{q-p} = \frac{p}{q-p}$ and the procedure sets $a_{q-1} = 0$, reduces y to 0 and stops after having executed q steps. Therefore, α contains exactly p ones, has length equal to q, i.e., $\alpha = (a_0, a_1, \ldots, a_{q-1})$, and finishes with $a_{q-1} = 0$.

As p, q are co-primes, there is no integer $k > 1$ such that q can be divided to k groups (periods), each one having $\frac{p}{k}$ ones and $\frac{q-p}{k}$ zeros. Therefore, α is non-periodic. □

We fix now $\alpha^k = (a_0^k, a_1^k \ldots, a_{q-1}^k)$, $0 \leq k \leq q-1$, to be the k-th left circular shift of α, with $\alpha^0 = \alpha$. The next proposition follows directly by the definition of the circular shifts and the non-periodicity of α.

Proposition 4. *For two circular shifts, $\alpha^k, \alpha^{k'}$, $k \neq k' \bmod q$ of α, it holds that*
(i) $a_t^k = a^{k'}_{(t+k-k') \bmod q}$, $t = 0, 1, \ldots, q-1$.
(ii) $\alpha^k \neq \alpha^{k'}$.

Let us denote the lexicographic relation between two binary sequences by $\succ$. Let also $1\alpha^k = (1, a_0^k, a_1^k \ldots, a_{q-1}^k)$ and $\alpha 1 = (a_0, a_1, \ldots, a_{q-1}, 1)$.

The following proposition gives two useful relations between the output α of the Algorithm Staircase(p, q) and its circular shifts.

Proposition 5. *For each* $k,\ 1 \leq k \leq q-1$, *it holds that*
(i) $\alpha \succ \alpha^k$.
(ii) If $a_0^k = 0$, *then* $1\alpha^k \succ \alpha 1$.

Proof. Let $y_k,\ 0 < k \leq q-1$, be the intermediate values of y at the beginning of the step k of Algorithm Staircase(p,q). First, we consider the procedure Staircase(p,q) initiated not by $y=0$, but by one of those intermediate values, say $y = y_k$. Then, if the procedure is allowed to iterate until y becomes again y_k, it will produce again a sequence of length q. In fact, this sequence will be the k-th circular shift of α, as the procedure in the first k steps, will produce the last $q-k-1$ digits of α and in the next $k+1$ steps the first $k+1$ digits of α. Next, we claim that if $y_k < y_{k'},\ k \neq k'$, then $\alpha^k \succ \alpha^{k'}$. To see this assume, by contradiction, that $y_k < y_{k'}$ and $\alpha^k \preceq \alpha^{k'}$ and let u be the minimum index, such that $a_u^k \neq a_u^{k'}$ and $a_u^{k'} = 1$. As $a_i^k = a_i^{k'}, 0 \leq i \leq u-1$ it follows that the difference of the y values, after those first $u-1$ steps, of the two runs of the procedure initiated with y_k and $y_{k'}$, is equal to $y_k - y_{k'}$. At step u, the procedure produces a $a_u^k = 0$ (for y_k) and $a_u^{k'} = 1$ (for $y_{k'}$). Hence, this step starts with $y \geq \frac{p}{q-p}$ and $y' < \frac{p}{q-p}$, respectively. However, $y - y' = y_k - y_{k'} < 0$, a contradiction.

For the point (i) of the proposition just observe that α and α^k are produced by two runs of the procedure initiated by $y_0 = 0$ and $y_k > 0$, respectively.

For the point (ii), observe first that $a_0 = 1$ (by Proposition 3) and $a_{q-1}^1 = 1$ (by Proposition 4). Therefore, $1\alpha^1 = \alpha 1$. Thus, it suffices to prove that if $a_{q-1}^k = 0$, then $\alpha^k \succ \alpha^1$ for each $k,\ 2 \leq k \leq q-1$. To produce α^1 and α^k the procedure starts with y_1 and y_k respectively. As $y_0 = 0$, we have that $y_1 = y_0 + 1 = 1$. As $a_{q-1}^k = 0$, it follows that $y_k = y_{k-1} - \frac{p}{q-p}$ and since $y_{k-1} < 1 + \frac{p}{q-p}$ (recall that this inequality holds for all values of y_k) we get $y_k < 1$. Therefore, by the claim above the relation in point (ii) holds. □

The following lemma together with Proposition 3 provides a proof for Theorem 2.

Lemma 3. *For a given density* $\frac{p}{q}$ *the binary sequences,* α *produced by* Staircase (p,q) *and* ω *produced by* CoolestFirst(h), *with* $h = \frac{2^q-1}{int(\alpha)}$, *are equal.*

Proof. Let $T_t,\ 0 \leq t \leq q-1$, be the temperature in the beginning of each execution step of CoolestFirst(h). Recall that $T_0 = 1$ and by CoolestFirst it follows that $T_t \in [1 - \frac{h}{2}, 1]$.

In order to produce a digit w_t, the procedure examines whether $T_t + h > 2$. By setting $h = \frac{2^q-1}{int(\alpha)}$, the latter inequality can be written as

$$T_t \cdot int(\alpha) + 2^q > 2int(\alpha) + 1. \tag{3}$$

As each T_t is calculated by a division by 2, the quantity $T_t \cdot int(\alpha)$ corresponds to the decimal expansion of the left circular shift $\alpha^{(q-t) \bmod q}$, $t = 0, \ldots, q-1$. Let $k = (q-t) \bmod q$. By converting (3) to its binary equivalent, we yield that

$$1\alpha^k \succ \alpha 1. \tag{4}$$

and by Lemma 5, if $a^k_{q-1} = 0$ then CoolestFirst(h) produces $w_t = 0$, otherwise it produces 1. Hence, at each step of the procedure we have that $w_t = a^k_{q-1}$. By applying Proposition 4, we yield that $a^k_{q-1} = a_{(q-1+q-t) \bmod q} = a_{q-t-1}$. □

6 Comments

We showed that the approximation factor of CoolestFirst is between 0.72 and 0.75. However, it seems to be a waste to start scheduling all cool jobs, since they could be used to fill idle times between hot jobs. This suggests an improved algorithm that gives higher priority to hot jobs over cool jobs. However we don't have the tools right now to analyze this new algorithm. Moreover, it remains an open question whether the throughput maximization problem accepts a PTAS or not.

References

1. Albers, S.: Energy-efficient algorithms. Communications of ACM 53(5), 86–96 (2010)
2. Albers, S.: Algorithms for dynamic speed scaling. In: Symposium on Theoretical Aspects of Computer Science (STACS), pp. 1–11 (2011)
3. Atkins, L., Aupy, G., Cole, D., Pruhs, K.: Speed scaling to manage temperature. In: Marchetti-Spaccamela, A., Segal, M. (eds.) TAPAS 2011. LNCS, vol. 6595, pp. 9–20. Springer, Heidelberg (2011)
4. Bampis, E., Letsios, D., Lucarelli, G., Markakis, E., Milis, I.: On multiprocessor temperature-aware scheduling problems. In: Snoeyink, J., Lu, P., Su, K., Wang, L. (eds.) AAIM-FAW 2012. LNCS, vol. 7285, pp. 149–160. Springer, Heidelberg (2012)
5. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. Journal of the ACM 54(1) (2007)
6. Birks, M., Cole, D., Fung, S.P.Y., Xue, H.: Online algorithms for maximizing weighted throughput of unit jobs with temperature constraints. In: Atallah, M., Li, X.-Y., Zhu, B. (eds.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 319–329. Springer, Heidelberg (2011)
7. Birks, M., Fung, S.P.Y.: Temperature aware online algorithms for scheduling equal length jobs. In: Atallah, M., Li, X.-Y., Zhu, B. (eds.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 330–342. Springer, Heidelberg (2011)
8. Birks, M., Fung, S.P.Y.: Temperature aware online scheduling with a low cooling factor. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 105–116. Springer, Heidelberg (2010)
9. Chrobak, M., Dürr, C., Hurand, M., Robert, J.: Algorithms for temperature-aware task scheduling in microprocessor systems. Sustainable Computing: Informatics and Systems 1(3), 241–247 (2011)
10. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News 36(2), 63–76 (2005)
11. Zhang, S., Chatha, K.S.: Approximation algorithm for the temperature-aware scheduling problem. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 281–288 (2007)

Algorithms for Cost-Aware Scheduling

Janardhan Kulkarni* and Kamesh Munagala**

Department of Computer Science, Duke University
{kulkarni,kamesh}@cs.duke.edu

1 Introduction

In this paper, we generalize classical machine scheduling problems by introducing a cost involved in processing jobs, which varies as a function of time. Before defining the problems formally and discussing the technical novelty, we present a few technological motivations for introducing this model.

Demand Response Models: Modern data centers are big consumers of electricity, and large providers see huge cost savings by even modest savings in electricity consumption. The conventional approach taken by system designers has been to build systems which are energy efficient, via technologies such as speed scalable processors, dynamic power-down and power-up mechanisms, new cooling technologies, and multi-core servers. These approaches have been investigated widely both in practice and theory; see [21,1,20] for more details.

Another relatively recent and less explored approach to reducing the energy costs is to exploit the *variable pricing of electricity*. The electricity markets in large parts of United States are moving towards variable pricing. As noted in [24], in those parts of the U.S. with wholesale electricity markets, prices vary on an hourly basis and are often not well correlated at different locations. Moreover, these variations are substantial, as much as a factor of ten from one hour to the next. Several suppliers offer "Time Of Use" plans [25], where they charge higher price for *peak* hours and considerably lower price during non-peak hours. Electricity markets are too complex to give a simple thumb rule on which this variation in cost depends but loosely speaking, the price depends on the resource used for generation: As the demand peaks, the cost goes up disproportionately as the suppliers have to rely on expensive and nonrenewable resources like coal to meet the demand [27]. In this sense, the cost of electricity is also an indicator of how "green" its generation is and its impact on environment.

This variation in prices of electricity offers opportunities for large scale system designers to cut down their electricity expenses by moving their workload both in space and time. Note that in contrast to energy efficient computing, the purpose of this line of work is not to reduce the amount of energy consumed per unit of work, but to reduce the cost for doing the work. In [24] the authors exploit the spatial nature of variation in electricity cost for scheduling, while

* Supported by NSF awards CCF-1008065 and IIS-0964560.
** Supported by an Alfred P. Sloan Research Fellowship, a gift from Cisco, and by NSF via grants CCF-0745761, CCF-1008065, and IIS-0964560.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 201–214, 2013.

in [10], the authors analyse a simple model to exploit the temporal nature of variation in electricity prices. The latter work is particularly relevant to our paper: They consider a system at a single location executing a workload that is delay tolerant, such as processing batch jobs. They further consider a pricing model where cost of electricity varies between two levels, a *base price* and a *peak price*. They propose simple schemes to defer the workload to less expensive base price periods, and show experimentally that it smoothly trades off costs for delay. The generalized scheduling problem we introduce models the effect temporal nature of variation in electricity prices on scheduling decisions. One of the highlights of our work is to analyze the model considered by authors in [10] from the theoretical perspective with worst case bounds (Section 2.2).

Spot Pricing in Data Centers: An entirely different technological motivation for cost-aware scheduling comes from the Amazon EC2 cloud computing system, which allows users to rent virtual machines on the cloud for computational needs. Amazon offers various pricing schemes to rent machines, one of which is *spot pricing*. Spot pricing enables the users to bid for unused capacity, and prices get set based on supply and demand. Again, as in the previous example, the cost of renting the machine on EC2 varies dynamically over time, offering opportunities for optimizing the cost and QoS of batch jobs on such a system.

1.1 Our Model and the Two-Cost Problem

The optimization problems arising in above applications can be captured by the following problem that we term Two-Cost, which generalizes the classical single-machine preemptive scheduling framework. There is a set J of n jobs, where each job J_j has processing time p_j, release time r_j, and weight w_j. Density of a job J_j is defined as ratio of $\frac{w_j}{p_j}$. For simplicity, we assume time is discrete, and the processing times and release times of the jobs are integers. There is a *processing cost* function $e(t)$: If a job J_j is scheduled at time t, it incurs processing cost $e(t)$. We assume that $e(t)$ is a piecewise constant function which takes exactly two distinct values, *high* and *low*, corresponding to the base and peak price of electricity markets. By scaling we assume that cost of processing at *high* time instants is β and 1 in low time instants. Given any schedule, the *processing cost* of J_j denoted by $E(j)$ is given by $\int_t e(t)x_j(t)dt$, where $x_j(t)$ indicates whether job J_j was scheduled at the instant t; since we assume time is discrete, $E(j) = \sum_t e(t)x_j(t)$. The completion time of job J_j denoted by C_j is the last time instant when this job is scheduled. The flow time of J_j is defined as $F_j = C_j - r_j$. Jobs arrive online and the cost function $e(t)$ changes in an online fashion as well. The objective is to minimize $\sum_{J_j \in J}(w_j F_j + E(j))$ – the sum of weighted flow time and processing cost.

1.2 Our Results and Techniques

In this paper we initiate the study of online scheduling problems with the objective of minimizing the processing costs plus some well-known QoS guarantees.

We show that these problems are significantly different from their counterparts without processing costs, hence requiring new algorithmic techniques. We assume the algorithm does not know the future job arrivals and the future cost function $e(t)$, and proceed via *speed augmentation* analysis, where we give the algorithm extra processing speed compared to OPT for the purpose of analysis. The holy grail of speed augmentation analysis is to design a so-called *scalable* algorithm: For any $\epsilon > 0$, the algorithm is $(1+\epsilon)$-speed, $O(\text{poly}(\frac{1}{\epsilon}))$-competitive.

In Section 2, we first show that no deterministic online algorithm for Two-Cost can be constant competitive even for unit length, unit weight jobs, when $e(t)$ is known in advance. We then present our *main result*. We show a scalable $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^3})$-competitive algorithm for Two-Cost.

Our algorithm for Two-Cost in Section 2 is the most natural one: Always schedule at low cost time instants. For high cost time instants, if the total flow time accumulated since the last scheduling decision is at least the cost of processing, then schedule using the highest density first priority rule, else wait. We outline the idea behind the analysis in Section 2.1. The analysis is complicated by the non-uniform nature of the problem - the behavior of the algorithm is different in the high and low cost instants, and these instants themselves arrive in an online fashion. The key decision that the algorithm has to make is whether to schedule a job at the current step in a high cost time instant, or wait for a low cost time instant that may arrive soon in the future. However, waiting poses a risk in that jobs could arrive in the future and create a huge backlog at the low cost time instants. To partially mitigate the backlogging effect, we resort to speed augmentation for the analysis, and show that this is necessary as well.

Technical Contributions. Speed augmentation and potential functions have proven to be useful techniques in the analysis of online algorithms for weighted flow time problems. See for example, scheduling policies on unrelated machines [9] and *speed scaling* problems [5,4,13]. These potential functions follow a similar template (the so-called *standard potential function* [15]), and are defined in terms of the future online cost (or cost-to-go) of the algorithm assuming no more jobs arrive in the system and how far the online algorithm is behind the optimal schedule in work processed. Due to the online nature of the cost function $e(t)$, one of the chief technical difficulties we face while analyzing this problem is that it is hard to give a closed form expression for the future cost of an online algorithm. Hence, it is not clear how to define any sort of potential function for our problem. We instead proceed via simplifying the input and certain revealing structural properties of OPT. We then observe a simple yet powerful *majorization* property (Theorem 4) of our schedule relative to the optimal schedule, as consequence of the fact that our algorithm uses HDF ordering of jobs. We use this characterization repeatedly in a non-trivial fashion to split the time horizon into suitably defined phases. Within these phases, we use a fairly simple charging argument to bound the cost of algorithm against the optimal. In effect we show that an algorithm that cannot estimate cost-to-go for its scheduling decision is competitive *even if* the costs in the future vary arbitrarily. We also believe Theorem 4 can be of independent interest in other scheduling problems.

1.3 Related Work and Comparision with Other Models

Use of speed augmentation in the analysis of online scheduling problems, particularly involving flow time objectives was first considered in [18]. Since then, several papers have used speed augmentation to show scalable algorithms for weighted flow time on various machine environments [8,6,9,16]. The introduction of the cost function $e(t)$ drastically changes the nature of the problem compared to the classical flow time problems. It is instructive to point out that our problem does not admit a competitive algorithm even for unit weight and unit length jobs, where as HDF is optimal for unit length jobs in this context for classical flow time. The fundamental difference in the complexity of this problem was noted earlier in the works on *robust machine scheduling*, which is a special case of our model [26,11]. In the offline case, Epstein *et al.* study the problem of minimizing weighted completion on a single machine when the machine can encounter unexpected failures [11] . They give a constant factor approximation algorithm on a single machine when jobs have no release dates and show polylogarithmic lowerbounds when jobs have release dates. Note that, in their problem only failure periods arrive online. These results sharply contrast with constant factor approximation algorithms known for minimizing weighted completion time on various machine environments. We outline the results we obtain for the offline version of our problem in Section 3.

Dynamic speed scaling and its variants have been studied extensively for power management. In this model pioneered by Yao *et al.* [28], the goal is to dynamically scale the speed of a processor to optimize power consumed (which is usually a convex function of speed) and some QoS metric like deadlines or flow time. This model has a rich literature in online algorithms and potential function design; see [5,4,23,7,2,3,21,12,22] for more details. Apart from the philosophical difference that we are concerned with minimizing the cost of power rather than efficient usage of it, we believe that our model is technically very different as well. Electricity costs vary with time in a non-monotone and adversarial fashion, whereas in speed scaling, the cost incurred by algorithms depends on the speed and is not a function of time. Therefore, the decision in speed scaling is to set the speed, while in our problem, the decision is about which time instants to process the jobs in. It is also interesting to note the technical similarities in these policies. In speed scaling algorithms, speed is set such that cost incurred on the speed is equal to the flow time of jobs at any time instant. Our algorithm also uses similar cost-balancing approach towards time slot selection policy. It would be interesting to study the effect of combining speed scaling with our model.

Another line of research which has technical similarities with our problem is power down mechanisms [17,19]. There are speed scalable processors, but transitioning from speed zero (sleep state) to non-zero speed (active state) incurs activation cost. These algorithms use algorithms for speed scaling as subroutine when the processor is active, but they also need to decide when to transition into and out of sleep states. These algorithms hence balance the cost of activation to the total flow time of jobs present at that time. Our algorithm also uses a similar cost balancing approach, but there are several subtle differences. For example,

the algorithms in [17,19] transition to sleep state only when there are no jobs in the system whereas our algorithm for Two-Cost may idle even when there are jobs present simply because processing cost is high. Further, the speed-scaling model is more sympathetic to lazy activation policies like ones used in [19] since any accumulated jobs can be cleared by varying the speed. But accumulating jobs in our problem poses a threat, since both jobs and high cost time instants arrive online. We emphasize that, even for unit length and unit weight jobs we cannot get a competitive algorithm where as almost all the problems considered in speed-scaling or power-down mechanisms admit competitive algorithms. In other words, the use of speed augmentation in speed scaling problems is for converting the schedule for fractional objective to the integral objective, whereas we need it even for a fractional schedule.

2 Online Algorithms for the Two-Cost Problem

In this section, we devise online algorithms for minimizing the sum of weighted flow time and processing cost on a single machine (with preemption). Recall that we denote the weights of the jobs by w_i, and the processing times by p_i. We assume without loss of generality that $e(t)$ takes either a value of 1 or β at all the time instants.

Before we present the scalable algorithm for Two-Cost, we first present lower bounds on the achievable competitive ratio. In this section, we first show that no online algorithm can have competitive ratio independent of the values taken by cost function $e(t)$, even when all jobs have the same weight and unit processing length, and when the cost function $e(t)$ is known in *advance*. We defer the proof of this lower bound to Appendix A.

Theorem 1. *No deterministic online algorithm for* Two-Cost *can have a competitive ratio independent of the values taken by* $e(t)$*, even when all jobs have unit length and equal weight and* $e(t)$ *is known in advance.*

To get around this negative result, our algorithms will use *speed augmentation* to be competitive - this means that to show a $O(1)$-competitive ratio, we pretend the algorithm runs on a *faster* machine than the optimal solution; the extra speed trades off with the competitive ratio. Let OPT denote both the optimal offline algorithm, as well as its value. Given any online algorithm and input sequence, there are two decisions the algorithm has to make every step:

Time Slot Selection: This policy decides which time slots to schedule jobs - we term these *active* time slots.
Job Selection Policy: Decides which job to schedule in each active time slot.

We briefly describe the analysis technique of speed augmentation, which is implicit in previous work [9,5,4].

Definition 1. *Given an algorithm A and speed s, we say that algorithm B is a s-speed simulation of A if:*

- *The active time slots of A and B are the same (or B simulates A).*
- *The job selection policy of B is same as A; however, B can process s units of jobs in every active time slot.*

Definition 2. *An online algorithm A is said to be s-speed, c-competitive if there is a s-speed simulation of A that is c competitive against OPT.*

We make this fine distinction for the reason that, schedule produced by A when run a machine with speed s can be completely different from schedule of B which takes the schedule of A on unit speed processor, but schedules s units of jobs whenever A processes 1 unit of job.We will show the following theorem in the sequel.

Theorem 2. Two-Cost *has a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^3})$-competitive algorithm.*

2.1 Proof Outline

We design our algorithm for the case when jobs have unit length with arbitrary weights and at each time step a single job is released. We later show how to convert this algorithm to handle jobs with arbitrary lengths using the ideas which have become standard now. For unit length jobs, the job selection policy of any algorithm is simple: Schedule that job J_i from the current queue with highest *density* or weight. This is the well-known *Highest Density First* (HDF) policy. Our overall online algorithm for unit length jobs is the most natural one: Always schedule using HDF in low cost instants. For high cost instants, we follow a ski-rental kind of policy. If the total flow time accumulated since the last scheduling decision is at least β, then schedule using HDF, else wait. We call this algorithm Balance. The hard part in defining a (standard) potential function is the non-uniformity in the processing cost. Instead, we first transform and simplify the input so that we only have to deal with unit length jobs, only one of which arrives per step.

The crux of our analysis is a *majorization property* of the HDF schedule, Theorem 4: If an online algorithm processing using HDF always lags another algorithm in terms of number of units processed, but eventually catches up, then if the initial weight of jobs in the queue of the first algorithm was smaller, the final weight will be smaller as well. We show that Balance always lags OPT in terms of number of jobs processed, hence the majorization result directly bounds the processing cost paid by Balance (Lemma 6).

To bound the flow time, we perform a speed augmentation analysis. Again, the analysis is complicated by the non-uniformity in processing costs between low and high cost time instants. We instead divide time into phases where the augmented Balance lags OPT. Using our majorization result, we show it is sufficient to analyze each phase separately. We now construct a simple charging scheme, and argue about the amortized cost, completing the proof (Lemma 7).

2.2 Simplifying the Input

By scaling the input, we can assume that $e(t)$ takes values either 1 or β. We also assume that processing times and release times of jobs take integer values. We

assume jobs are released at the beginning of a time slot. Our scheduling policies will be based on considering the weight of jobs in the queue *during* the time slot, and the processing happens at the end of the time slot.

We call a time instant t as *high cost* time instant if $e(t) = \beta$ in the interval $[t, t+1)$. Other wise, we call it as *low cost* time instant. We assume without loss of generality that $e(t)$ changes only at integral values of t. Thus every time instant is either a low cost time instant or a high cost time instant.

Step 1: Unit Length Jobs. The following lemma is an easy consequence of similar results in [9,4,8]. The proof follows by replacing each job J_i with p_i jobs of unit length and weight w_i/p_i.

Lemma 3. *If an online algorithm A is s-speed c-competitive for minimizing the objective $\sum_j w_j F_j$ for unit length jobs, then it is $(1+\epsilon)s$-speed $(1+\frac{1}{\epsilon})c$-competitive when jobs have arbitrary length.*

The above lemma allows us to focus on unit length jobs in designing the online algorithm. For unit length jobs, given the set of active slots, it is easy to characterize the job selection policy: If a slot is active, the algorithm will simply schedule that job J_i from its queue with highest *density* or weight per unit length, w_i. This is the well-known *Highest Density First* (HDF) policy. Note however that even for unit length jobs, Theorem 1 shows there is no 1-speed algorithm with competitive ratio independent of β. We therefore need to use a speed augmentation analysis even for this case. We redefine OPT to be the optimal offline algorithm for this new problem instance (with unit length jobs).

Step 2: Modifying Release Dates. We assume that only one job is released at each time step. This follows as consequence of Step 1 and we omit the details.

Step 3: Modifying the optimal schedule. Given any algorithm A, let $W^A(t)$ denote the total weight of jobs in A's queue during time t. The proof of the following claim follows easily from the observation that making OPT process a job has cost β, while the total weight of jobs in the queue contributes to the flow time. As a consequence, if $W^{OPT}(t) \geq \beta$, we can assume OPT schedules at time t.

Claim. With $O(1)$ loss in competitive ratio, we can assume that in any interval $I = [s, d]$ where OPT does not process jobs, $\sum_{t \in I} W^{OPT}(t) < \beta$.

2.3 Online Algorithm Balance

The online algorithm Balance is characterized by the following two rules. Here $W^{\mathcal{A}}(t)$ denotes the total weight of jobs in Balance's queue during time t.

Time slot selection Policy: If $e(t) = 1$, then mark t as active. If $e(t) = \beta$ then let t' be the last active time instant. Mark t as active if $\sum_{u \in (t', t]} W^{\mathcal{A}}(u) \geq \beta$.

Job selection Policy (HDF): If t is active, then among the set of jobs available at the time t, schedule the one with highest weight.

2.4 Analysis of BALANCE

We begin this section by showing some important structural properties of the schedule produced by BALANCE, which we use subsequently in our analysis.

Majorization Property of HDF. Before we analyze BALANCE, we observe a simple yet important property of scheduling jobs in HDF, which may be of independent interest. Let A and B denote two scheduling algorithms which process same number of unit length jobs in the interval $[s, d]$. Suppose A processes jobs in HDF and always lags B, i.e., total number of jobs processed by A at any time $t \in [s, d]$ is always less than B. Then, majorization property says that if A and B started off with equal weight at the beginning, and jobs arrive in the interval $[s, d]$, then the weight of jobs in A will have at most that in B in the end of this interval. But more importantly, for every job J_j in A's queue, total weight of jobs in B's queue with weight at least w_j will be at most that of A's. We make the statement formal below.

For a scheduling algorithm A processing unit length jobs in the interval $[t_1, t_2]$, let $Q^A(t)$ denote the set of jobs A has at the time t. Let $Q^A_{\geq w}(t)$ denote the subset of those jobs with weight at least w. Let $N^A(t_1, t_2)$ denote the number of jobs A has scheduled in the interval $[t_1, t_2]$. Then we have the following theorem about scheduling jobs in HDF.

Theorem 4. *(Proved in Appendix A) Let A be a scheduling algorithm which processes unit length jobs using the HDF job selection policy, and B be any other scheduling algorithm on the same input. Suppose $\forall J_i \in Q^B(t_1), |Q^A_{\geq w_i}(t_1)| \leq |Q^B_{\geq w_i}(t_1)|$. If $N^A(t_1, t_2) = N^B(t_1, t_2)$ and $\forall t \in [t_1, t_2], N^A(t_1, t) \leq N^B(t_1, t)$, then $\forall J_i \in Q^B(t_2), |Q^A_{\geq w_i}(t_2)| \leq |Q^B_{\geq w_i}(t_2)|$. Further, $W^A(t_2) \leq W^B(t_2)$.*

Bounding the Processing Cost. From this point on, we will use $\mathcal{A}$ to denote the schedule produced BALANCE. The following lemma is the crucial property of BALANCE: Compared to OPT, $\mathcal{A}$ always lags in total processing done.

Lemma 5. *(Proved in Appendix A) In the schedule $\mathcal{A}$ produced by* BALANCE, *$\forall t \in [0, t], N^{\mathcal{A}}(0, t) \leq N^{OPT}(0, t)$.*

Let $E^{\mathcal{A}}, E^{OPT}$ denote the total processing cost of BALANCE and OPT respectively. The processing cost $E^{\mathcal{A}}$ can now be bounded using the above lemmas.

Lemma 6. *(Proved in Appendix A) Total processing cost of $\mathcal{A}$, $E^{\mathcal{A}} \leq E^{OPT}$.*

Flow Time via Speed Augmentation. We now analyze the schedule produced by BALANCE using speed augmentation. In particular, we consider a class of algorithms, SIMULATE-BALANCE(s) that uses the same active time-slots as BALANCE (in that sense, it simulates BALANCE). However, on each active time slot of BALANCE, the new algorithm schedules at most s units of jobs from its own queue using the HDF policy.

It follows directly from Lemma 6 that the processing cost of SIMULATE-BALANCE(s) is at most $s \cdot E^{OPT}$. In the sequel, we bound the flow time of

SIMULATE-BALANCE$(1+\epsilon)$ against the flow time of OPT, which we denote F^{OPT}. We will use SIMULATE-BALANCE to mean SIMULATE-BALANCE$(1+\epsilon)$, where the factor $(1+\epsilon)$ will be implicit.

In order to bound the weighted flow time, we first split the contribution of the weighted flow time to individual time steps (and hence time intervals). We treat time as a discrete quantity here with each time instant t denoting the time interval $[t, t+1)$. For an algorithm B, let F^B denote the total weighted flow time, and let $W^B(t)$ denote the weight of jobs in queue of B at time instant t (this excludes the job getting processed at the time step t). Then, we have: $F^B = \sum_{t\geq 0} W^B(t) + \sum_j w_j$.

Let $F_\epsilon^{\mathcal{A}}$ denote the weighted flow time of SIMULATE-BALANCE. Let $W_\epsilon^{\mathcal{A}}(t)$ denote the total weight of the jobs in the queue of SIMULATE-BALANCE at time t. Recall that F^{OPT} and E^{OPT} are the weighted flow time and processing cost of OPT, respectively. We will prove the following lemma in the sequel, which will complete the proof of Theorem 2.

Lemma 7. $F_\epsilon^{\mathcal{A}} \leq O\left(\frac{1}{\epsilon^2}\right)(F^{OPT} + E^{OPT})$.

Let $\mathcal{W}_\epsilon^{\mathcal{A}}(t)$ denote the fractional weight of jobs in the queue of SIMULATE-BALANCE; $\mathcal{W}_\epsilon^{\mathcal{A}}(t) = \sum_{J_i \in Q_\epsilon^{\mathcal{A}}} w_i x_i(t)$, where $x_i(t) \in [0,1]$ denotes the remaining processing time of J_i at the time t. Then, weighted fractional flow time of SIMULATE-BALANCE is defined as: $f_\epsilon^{\mathcal{A}} = \sum_{J_j \in J} \frac{w_j}{2} + \sum_{t\geq 0} \mathcal{W}_\epsilon^{\mathcal{A}}(t)$.

We start by bounding the weighted fractional flow time $f_\epsilon^{\mathcal{A}}$ of SIMULATE-BALANCE. For an interval $[t_1, t_2]$, let $P_\epsilon^{\mathcal{A}}(t_1, t_2)$, $P^{OPT}(t_1, t_2)$ denote the total units of processing done by SIMULATE-BALANCE and OPT in the interval $[t_1, t_2]$.

Definition 3.
- *An interval $[s, d)$ is a* lag-interval *if $P_\epsilon^{\mathcal{A}}(s, d) \geq P^{OPT}(s, d)$, but for all $t \in [s, d)$, $P_\epsilon^{\mathcal{A}}(s, t) < P^{OPT}(s, t)$.*
- *An interval $[s, d)$ is a* lead-interval *if for all $t \in [s, d)$,* SIMULATE-BALANCE *processes more units that time instant than OPT, but at time d, it processes less units than OPT.*
- *The entire time horizon partitions into a sequence of alternating lag and lead intervals of the form $[0, d_1), [d_1, d_2), \ldots$, where $0 < d_1 < d_2 \cdots$. We call the interval $[d_i, d_{i+1})$ as the i^{th} phase.*

The following lemma, proved in Appendix A follows by a repeated application of Theorem 4.

Lemma 8. *For any phase $[d_i, d_{i+1})$, $\mathcal{W}_\epsilon^{\mathcal{A}}(d_i) \leq W^{OPT}(d_i)$. Furthermore, for any t in a lead phase $[d_i, d_{i+1})$, we have $\mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq W^{OPT}(t)$.*

Proof of Lemma 7: We will bound the fractional flow time of SIMULATE-BALANCE against that of OPT within each phase. For a *lead-phase*, the bound is simple since at every time instant t within the phase, we have $\mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq W^{OPT}(t)$. We therefore focus only on lag-phases. Since SIMULATE-BALANCE always lags OPT in terms of number of units processed within a lag phase, we have the following easy consequences for any time instant in a lag phase:

- If Simulate-Balance processes, it processes at least ϵ more units of jobs than OPT.
- If Simulate-Balance does not process, then it has to be a high cost instant.

We will now use a charging argument to show that the fractional flow time of Simulate-Balance within the lag-phase is at most $O(1/\epsilon)$ times the total cost spent by OPT within the phase. Fix a lag-phase i. We use $P^{OPT}(t)$ and $P_\epsilon^{\mathcal{A}}(t)$ to abbreviate $P^{OPT}(d_i, t)$ and $P_\epsilon^{\mathcal{A}}(d_i, t)$ respectively. Instead of analysing elaborate charging rules, for each time instant $t \in [d_i, d_{i+1})$ we define the following simple potential function, which keeps track of how far Balance is behind compared to OPT.

$$\Phi(t) = \frac{2\beta}{\epsilon}\left(P^{OPT}(t) - P_\epsilon^{\mathcal{A}}(t)\right) \tag{1}$$

The amortized cost paid by Simulate-Balance is defined as :

$$\theta(t) = \mathcal{W}_\epsilon^{\mathcal{A}}(t) + \Phi(t) - \Phi(t-1) \tag{2}$$

Define $[t, t')$ as an *idle period* if neither OPT nor Simulate-Balance schedule jobs in that interval. The following lemma follows from an (almost) straightfoward analysis of the potential function in Equation (1). Only corner cases are the periods when the change in potential is zero (idle periods) and one has to bound cost of algorithm during such periods. We push the details to Appendix A.

Lemma 9. *For any lag-phase $[d_i, d_{i+1})$ and any idle period $X = [t, t')$ so that $t, t' \in [d_i, d_{i+1})$, we have:*

$$\sum_{t \in X} \theta(t) \leq O(\frac{1}{\epsilon}) \left(\sum_{t \in X} \left(W^{OPT}(t) + e(t) \cdot \mathcal{I}^{OPT}(t) \right) \right) \tag{3}$$

where $\mathcal{I}^{OPT}(t)$ is the indicator variable denoting whether OPT schedules at t.

Equation (3) is true for lead-phases directly from Lemma 8. Therefore, summing over all phases, we conclude that the weighted fractional flow time of Simulate-Balance is at most $O(1/\epsilon)$ times the total cost of OPT. We then convert the weighted fractional flow time into weighted flow time by augmenting Simulate-Balance with another $(1+\epsilon)$-speed using ideas similar to that in Lemma 3. Therefore, we conclude that on a machine with $(1+\epsilon)$-speed, $F_\epsilon^{\mathcal{A}} \leq O(\frac{1}{\epsilon^2})(F^{OPT} + E^{OPT})$. This concludes the proof of Lemma 7.

Proof of Theorem 2: We first bound the competitive ratio of Simulate-Balance for the objective minimizing $\sum_j (w_j F_j + E(j))$ when jobs are of unit length. From Lemma 6, it follows that total processing cost of Simulate-Balance is at most $(1+\epsilon)E^{OPT}$. Lemma 7 shows that on a machine with $(1+\epsilon)$-speed, $F_\epsilon^{\mathcal{A}} \leq \frac{1}{\epsilon^2}(F^{OPT} + E^{OPT})$. Putting all the pieces together, we conclude that Balance is $(1+\epsilon)$-speed $O\left(\frac{1}{\epsilon^2}\right)$-competitive for unit length jobs with arbitrary weights. Using Lemma 3 we finally conclude that Balance is $(1+\epsilon)$-speed $O\left(\frac{1}{\epsilon^3}\right)$-competitive for arbitrary length jobs.

3 Conclusion

In this paper, we presented a scalable online algorithm for the Two-Cost problem. In the full version of the paper, we show that with $K > 2$ levels of electricity cost, we need a speed augmentation of at least $K - 1$ to achieve bounded competitive ratio, even when electricity costs are known in advance. An interesting question that we seek to explore is whether such lower bounds can be circumvented using the framework of *speed scaling* [5], where the processor can be made to run faster by paying cost which is a convex function of the processing speed.

In the full version of the paper, we also study offline version of this problem with completion time objective, since approximating flow-time even without the cost function is one of the most important open problems in scheduling theory. We show that for the *offline* setting the LP formulation of Hall *et al.* [14] can be extended to yield a $O(\frac{1}{\epsilon})$ approximation to $1|r_j, pmtn| \sum_j (w_j F_j + E(j))$ with $(1+\epsilon)$-speed augmentation,for arbitrary $e(t)$. We also establish interesting connections of this problem to *universal scheduling* and scheduling with limited machine availability [11,26], which yield pseudo-polynomial time constant factor approximation algorithms for $1|r_j, pmtn| \sum_j (w_j F_j + E(j))$. See Appendix B.

Acknowledgment. We thank Sudipto Guha, Bruce Maggs, and Jeff Chase for several helpful discussions.

References

1. Albers, S.: Algorithms for energy saving. In: Albers, S., Alt, H., Näher, S. (eds.) Festschrift Mehlhorn. LNCS, vol. 5760, pp. 173–186. Springer, Heidelberg (2009)
2. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms 3(4) (2007)
3. Bansal, N., Chan, H.-L., Pruhs, K.: Speed scaling with an arbitrary power function. In: SODA (2009)
4. Bansal, N., Chan, H.-L., Pruhs, K.: Competitive algorithms for due date scheduling. Algorithmica 59(4) (2011)
5. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM 54(1) (2007)
6. Bansal, N., Pruhs, K.: Server scheduling in the l_p norm: A rising tide lifts all boat. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, STOC (2003)
7. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: SODA (2007)
8. Becchetti, L., Leonardi, S., Marchetti-Spaccamela, A., Pruhs, K.R.: Online weighted flow time and deadline scheduling. In: Goemans, M.X., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX-RANDOM 2001. LNCS, vol. 2129, pp. 36–47. Springer, Heidelberg (2001)
9. Chadha, J.S., Garg, N., Kumar, A., Muralidhara, V.N.: A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In: STOC (2009)
10. Chase, J.: Demand response for computing centers, `http://www.cs.duke.edu/~chase/dr.pdf`

11. Epstein, L., Levin, A., Marchetti-Spaccamela, A., Megow, N., Mestre, J., Skutella, M., Stougie, L.: Universal sequencing on a single machine. In: Eisenbrand, F., Shepherd, F.B. (eds.) IPCO 2010. LNCS, vol. 6080, pp. 230–243. Springer, Heidelberg (2010)
12. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling heterogeneous processors isn't as easy as you think. In: SODA (2012)
13. Gupta, A., Krishnaswamy, R., Pruhs, K.: Scalably scheduling power-heterogeneous processors. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 312–323. Springer, Heidelberg (2010)
14. Hall, L.A., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: Off-line and on-line algorithms. In: SODA (1996)
15. Im, S., Moseley, B., Pruhs, K.: A tutorial on amortized local competitiveness in online scheduling. SIGACT News 42(2) (2011)
16. Im, S., Moseley, B., Pruhs, K.: Online scheduling with general cost functions. In: SODA (2012)
17. Irani, S., Shukla, S., Gupta, R.: Algorithms for power savings. ACM Trans. Algorithms 3(4) (November 2007)
18. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM 47 (July 2000)
19. Lam, T.-W., Lee, L.-K., Ting, H.-F., To, I.K.K., Wong, P.W.H.: Sleep with guilt and work faster to minimize flow plus energy. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 665–676. Springer, Heidelberg (2009)
20. Murugesan, S.: Harnessing green it: Principles and practices. IT Professional 10(1) (2008)
21. Pruhs, K.: Green computing algorithmics. In: FOCS, pp. 3–4 (2011)
22. Pruhs, K., Stein, C.: How to schedule when you have to buy your energy. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX and RANDOM 2010. LNCS, vol. 6302, pp. 352–365. Springer, Heidelberg (2010)
23. Pruhs, K., Uthaisombut, P., Woeginger, G.: Getting the best response for your erg. ACM Trans. Algorithms 4, 38:1–38:17 (2008)
24. Qureshi, A., Weber, R., Balakrishnan, H., Guttag, J., Maggs, B.: Cutting the electric bill for internet-scale systems. In: SIGCOMM (2009)
25. Electricity rates, `http://www.pge.com/tariffs/electric.shtml`
26. Schmidt, G.: Scheduling with limited machine availability. European Journal of Operational Research 121, 1–15 (1998)
27. Official Statistics. United States Department of Energy, `http://www.eia.doe.gov`
28. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced cpu energy. In: FOCS (1995)

A Missing Proofs

Proof of Theorem 1. Let $e(t) = \beta$ in the interval $[1, \ldots \sqrt{\beta}]$ and $e(t) = 1$ elsewhere. The adversary releases a unit length, unit weight job at each time instant $t \in [1, \ldots \sqrt{\beta}]$. Let A be any online algorithm. Consider the number of jobs in the queue of A at time $t = \sqrt{\beta}$. If A has more than $\beta^{\frac{1}{4}}$ jobs then the adversary releases one job at each time instant $t > \sqrt{\beta}$. For this input, the

optimal offline algorithm will process each job released in the interval $[1, \ldots \sqrt{\beta}]$ by paying a processing cost of β, hence number of jobs it has at any time is at most one. However, A accumulates $\beta^{\frac{1}{4}}$ jobs by the time $t = \sqrt{\beta}$ which it cannot clear subsequently. Hence there are at least $\beta^{\frac{1}{4}}$ jobs in its queue at every time instant $t > \sqrt{\beta}$, incurring a cost of $\beta^{\frac{1}{4}}$ towards flowtime at each time step. Therefore, the competitive ratio of A is at least $\beta^{\frac{1}{4}}$. Next, consider the case when A has less than $\beta^{\frac{1}{4}}$ jobs at time $t = \sqrt{\beta}$. In this case, the adversary will not release any more jobs. For this instance, the optimal offline algorithm will not process any jobs in the interval $[1, \ldots \sqrt{\beta}]$ and processes all jobs in the low cost time instants following $t > \sqrt{\beta}$ incurring a total cost of $O(\beta)$. The competitive ratio of A is at least $\beta^{\frac{1}{4}}$, since the algorithm pays at least $\beta^{\frac{5}{4}}$ towards the processing cost.

Proof of Theorem 4. We prove this by contradiction. Suppose at time t_2, there is a job $J_i \in Q^B(t_2)$ such that $|Q^A_{\geq w_i}(t_2)| > |Q^B_{\geq w_i}(t_2)|$. Consider the set of jobs processed by A in the interval $[t_1, t_2]$. If the weight of all these jobs is at least w_i, then since both algorithms process equal number of jobs in $[t_1, t_2]$ and B had more jobs initially of weight at least w_i, it must have more jobs with at least weight w_i at t_2. This is an immediate contradiction. Next, consider the case where A processes a job of weight less than w_i in the interval $[t_1, t_2]$. Let $t' \in [t_1, t_2]$ be the last time instant when A scheduled a job with weight less than w_i. Since A schedules jobs using HDF, it must be the case that $|Q^A_{\geq w_i}(t')| = 0$. Now observe that A processes at least as many jobs as B in the interval $[t', t_2]$. If $J_{\geq w_i}(t', t_2)$ denotes the set of jobs with weight greater than w_i released in the interval $[t', t_2]$ then $|Q^A_{\geq w_i}(t_2)| = |J_{\geq w_i}(t', t_2)| - N^A(t', t_2)$. Since, $N^A(t', t_2) \geq N^B(t', t_2)$, we have $|Q^A_{\geq w_i}(t_2)| \leq |Q^B_{\geq w_i}(t_2)|$. Hence we get a contradiction.

Proof of Lemma 5. For the sake of contradiction, let t_1 be the first time instant when $N^{\mathcal{A}}(0, t_1) > N^{OPT}(0, t_1)$ and $t_2 < t_1$ be the last time instant when $\mathcal{A}$ scheduled a job. By the definition of t_1 and t_2, we note that both OPT and $\mathcal{A}$ do not process any jobs in the interval $(t_2, t_1]$ and $N^{\mathcal{A}}(0, t_2) = N^{OPT}(0, t_2)$. Moreover, in the interval $[0, t_2]$ $\mathcal{A}$ lags OPT; that is, $\forall t \in [0, t_2]$, $N^{\mathcal{A}}(0, t) \leq N^{OPT}(0, t)$. Since BALANCE processes jobs in HDF, we apply Lemma 4 over the interval $[0, t_2]$ to claim that $W^{\mathcal{A}}(t_2) \leq W^{OPT}(t_2)$. If t_1 is a low cost time instant, we conclude that OPT will also process a job since $W^{OPT}(t_1) > 0$. Next, consider the case when t_1 is a high cost time instant. Since $\mathcal{A}$ is processing a job at t_1, from the description of BALANCE, we have $\sum_{t=t_2+1}^{t_1} W^{\mathcal{A}}(t) \geq \beta$. However, $W^{OPT}(t_2) \geq W^{\mathcal{A}}(t_2)$ and OPT does not process jobs in the interval $(t_2, t_1]$, then it must be the case that $\sum_{t=t_2+1}^{t} W^{OPT}(t) \geq \beta$. Therefore, by Claim 2.2 OPT also processes at t_1. This completes the proof.

Proof of Lemma 6. Proof follows from Lemma 5. For the sake of contradiction, let t be the first time instant when total processing cost of $\mathcal{A}$ is more than OPT. We note that in the interval $[0, t]$, when there are jobs to process, $\mathcal{A}$ has no idle time slots during the low cost time instants. Therefore, $\mathcal{A}$ schedules at least as many jobs as OPT in the low cost time instants of the interval $[0, t]$. Since the total processing cost of $\mathcal{A}$ is more than OPT at time t, $\mathcal{A}$ must have scheduled

more jobs in high cost time instants compared to OPT. This implies that total number of jobs scheduled by BALANCE in the interval $[0, t]$ is greater than that of OPT, which contradicts Lemma 5.

Proof of Lemma 8. We prove this by induction on i. If the phase is a lead phase, the induction is trivial since at each step, SIMULATE-BALANCE processes more units than OPT, hence for any $t \in [d_i, d_{i+1})$, we have $\mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq W^{OPT}(t)$. For a lag phase $[d_i, d_{i+1})$, SIMULATE-BALANCE processes jobs in HDF, always lags OPT in terms of number of units processed within the phase, but catches up with OPT at time d_{i+1}. We simply invoke the Theorem 4 on the jobs processed within the phase to argue that if $\mathcal{W}_\epsilon^{\mathcal{A}}(d_i) \leq W^{OPT}(d_i)$, then $\mathcal{W}_\epsilon^{\mathcal{A}}(d_{i+1}) \leq W^{OPT}(d_{i+1})$. The details are straightforward and omitted.

Proof of Lemma 9. By the description of BALANCE and since at most one job arrives each time step, we have: $\sum_{t \in X} \mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq 2\beta$. At time t', there are two cases:

SIMULATE-BALANCE **schedules:** In this case, it schedule ϵ more units than OPT, so the potential drops by at least 2β. The sum of flow time over the idle period is at most 2β and hence, the amortized cost is at most 0.

SIMULATE-BALANCE **does not schedule:** In this case, this time instant is a high cost time instant. OPT pays at least β in processing cost. The potential increases by at most $\frac{2\beta}{\epsilon}$, while SIMULATE-BALANCE pays at most 2β in flow time. Therefore, the amortized cost of SIMULATE-BALANCE is at most $O(\beta/\epsilon)$.

In either case, the amortized cost paid by SIMULATE-BALANCE is at most $O(1/\epsilon)$ times OPT's flow time plus processing cost. Next, note that $\Phi(d_i) - \Phi(d_{i+1} - 1)$ is non-negative in the entire time interval $[d_i, d_{i+1})$ since SIMULATE-BALANCE lags OPT. From Lemma 8 we know that $\mathcal{W}_\epsilon^{\mathcal{A}}(d_i) \leq W^{OPT}(d_i)$. Hence we conclude that over the interval $[d_i, d_{i+1})$, total weighted fractional flow time of SIMULATE-BALANCE is upper bounded by:

$$\sum_{t=d_i}^{d_{i+1}-1} \mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq O(\frac{1}{\epsilon}) \sum_{t=d_i}^{d_{i+1}-1} \left(W^{OPT}(t) + e(t) \cdot \mathcal{I}^{OPT}(t)\right)$$

B Discussion on Offline Case

See the link http://www.cs.le.ac.uk/events/WAOA2012/AppendixKM.pdf

A Unifying Tool for Bounding the Quality of Non-cooperative Solutions in Weighted Congestion Games*

Vittorio Bilò

Department of Mathematics and Physics "Ennio De Giorgi", University of Salento, Provinciale Lecce-Arnesano, P.O. Box 193, 73100 Lecce, Italy
vittorio.bilo@unisalento.it

Abstract. We present a general technique, based on a primal-dual formulation, for analyzing the quality of self-emerging solutions in weighted congestion games. With respect to traditional combinatorial approaches, the primal-dual schema has at least three advantages: first, it provides an analytic tool which can always be used to prove tight upper bounds for all the cases in which we are able to characterize exactly the polyhedron of the solutions under analysis; secondly, in each such a case the complementary slackness conditions give us an hint on how to construct matching lower bounding instances; thirdly, proofs become simpler and easy to check. For the sake of exposition, we first apply our technique to the problems of bounding the prices of anarchy and stability of exact and approximate pure Nash equilibria, as well as the approximation ratio of the solutions achieved after a one-round walk starting from the empty strategy profile, in the case of affine latency functions and we show how all the known upper bounds for these measures (and some of their generalizations) can be easily reobtained under a unified approach. Then, we use the technique to attack the more challenging setting of polynomial latency functions. In particular, we obtain the first known upper bounds on the price of stability of pure Nash equilibria and on the approximation ratio of the solutions achieved after a one-round walk starting from the empty strategy profile for unweighted players in the cases of quadratic and cubic latency functions.

1 Introduction

Characterizing the quality of self-emerging solutions in non-cooperative systems is one of the leading research direction in Algorithmic Game Theory. Given a game $\mathcal{G}$, a social function $\mathcal{F}$ measuring the quality of any solution which can be realized in $\mathcal{G}$, and the definition of a set $\mathcal{E}$ of certain self-emerging solutions,

* This work was partially supported by the PRIN 2008 research project COGENT "Computational and game-theoretic aspects of uncoordinated networks" funded by the Italian Ministry of University and Research and by the "Progetto 5 per mille per la ricerca": "Collisioni fra vortici puntiformi e fra filamenti di vorticità: singolarità, trasporto e caos" at the University of Salento.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 215–228, 2013.

we are asked to bound the ratio $\mathcal{Q}(\mathcal{G},\mathcal{E},\mathcal{F}) := \mathcal{F}(K)/\mathcal{F}(O)$, where K is some solution in $\mathcal{E}(\mathcal{G})$ (usually either the worst or the best one with respect to $\mathcal{F}$) and O is the solution optimizing $\mathcal{F}$.

In the last ten years, there has been a flourishing of contribution in this topic and, after a first flood of unrelated results, coming as a direct consequence of the fresh intellectual excitement caused by the affirmation of this new research direction, a novel approach, aimed at developing a more mature understanding of which is the big picture standing behind these problems and their solutions, is now arising.

In such a setting, Roughgarden [18] proposes the so-called "smoothness argument" as a unifying technique for proving tight upper bounds on $\mathcal{Q}(\mathcal{G},\mathcal{E},\mathcal{F})$ for several notions of self-emerging solutions $\mathcal{E}$, when $\mathcal{G}$ satisfies some general properties, K is the worst solution in $\mathcal{E}(\mathcal{G})$ and $\mathcal{F}$ is defined as the sum of the players' payoffs. He also gives a more refined interpretation of this argument and stresses also its intrinsic limitations, in a subsequent work with Nadav [16], by means of a primal-dual characterization which shares lot of similarities with the primal-dual framework we provide in this paper. Anyway, there is a subtle, yet substantial, difference between the two approaches and we believe that the one we propose is more general and powerful. Both techniques formulate the problem of bounding $\mathcal{Q}(\mathcal{G},\mathcal{E},\mathcal{F})$ via a (primal) linear program and, then, an upper bound is achieved by providing a feasible solution for the related dual program. But, while in [16] the variables defining the primal formulation are yielded by the strategic choices of the players in both K and O (as one would expect), in our technique the variables are the parameters defining the players' payoffs in $\mathcal{G}$, while K and O play the role of fixed constants. As it will be clarified later, such an approach, although preserving the same degree of generality, applies to a broader class of games and allows for a simple analysis facilitating the proof of tight results. In fact, as already pointed out in [16], the Strong Duality Theorem assures that each primal-dual framework can always be used to derive the exact value of $\mathcal{Q}(\mathcal{G},\mathcal{E},\mathcal{F})$ provided that, for any solution S which can be realized in $\mathcal{G}$, $\mathcal{F}(S)$ can be expressed though linear programming and (i) the polyhedron defining $\mathcal{E}(\mathcal{G})$ can be expressed though linear programming, when K is the worst solution in $\mathcal{E}(\mathcal{G})$ with respect to $\mathcal{F}$, (ii) the polyhedron defining K can be expressed though linear programming, when K is the best solution in $\mathcal{E}(\mathcal{G})$ with respect to $\mathcal{F}$. Moreover, in all such cases, by applying the "complementary slackness conditions", we can figure out which pairs of solutions (K,O) yield the exact value of $\mathcal{Q}(\mathcal{G},\mathcal{E},\mathcal{F})$, thus being able to construct quite systematically matching lower bounding instances.

We consider three sets of solutions $\mathcal{E}$, namely, (i) ϵ-approximate pure Nash equilibria (ϵ-PNE), that is, outcomes in which no player can improve her situation of more than an additive factor ϵ by unilaterally changing the adopted strategy (in this case, $\mathcal{Q}(\mathcal{G},\mathcal{E},\mathcal{F})$ is called the approximate price of anarchy of $\mathcal{G}$ (ϵ-PoA($\mathcal{G}$)) when K is the worst solution in $\mathcal{E}(\mathcal{G})$, while it is called the approximate price of stability of $\mathcal{G}$ (ϵ-PoS($\mathcal{G}$)) when K is the best solution in $\mathcal{E}(\mathcal{G})$); (ii) pure Nash equilibria (PNE), that is, the set of outcomes in which no player

can improve her situation by unilaterally changing the adopted strategy (by definition, each 0-PNE is a PNE and the terms price of anarchy ($\mathrm{PoA}(\mathcal{G})$) and price of stability ($\mathrm{PoS}(\mathcal{G})$) are used in this case); (*iii*) solutions achieved after a one-round walk starting from the empty strategy profile [15], that is, the set of outcomes which arise when, starting from an initial configuration in which no player has done any strategic choice yet, each player is asked to select, sequentially and according to a given ordering, her best possible current strategy (in this case, K is always defined as the worst solution in $\mathcal{E}(\mathcal{G})$ and $\mathcal{Q}(\mathcal{G}, \mathcal{E}, \mathcal{F})$ is denoted by $\mathrm{Apx}^1_{\emptyset}(\mathcal{G})$).

Our Contribution. Our method reveals to be particularly powerful when applied to the class of weighted congestion games. In these games there are n players competing for a set of resources. These games have a particular appeal since, from one hand, they are general enough to model a variety of situations arising in real life applications and, from the other one, they are structured enough to allow a systematic theoretical study. For example, for the case in which all players have the same weight (unweighted players), Rosenthal [19] proved through a potential function argument that PNE are always guaranteed to exist, while general weighted congestion games are guaranteed to possess PNE if and only if the latency functions are either affine or exponential [11–13, 17].

In order to illustrate the versatility and usefulness of our technique, we first consider the well-known and studied case in which the latency functions associated with the resources are affine and $\mathcal{F}$ is the sum of the players' payoffs and show how all the known results (as well as some of their generalizations) can be easily reobtained under a unifying approach. For ϵ-PoA and ϵ-PoS in the unweighted case and for $\mathrm{Apx}^1_{\emptyset}$, we reobtain known upper bounds with significatively shorter and simpler proofs (where, by simple, we mean that only basic notions of calculus are needed in the arguments), while for the generalizations of the ϵ-PoA and the ϵ-PoS in the weighted case, we give the first upper bounds known in the literature.

After having introduced the technique, we show how it can be used to attack the more challenging case of polynomial latency functions. In such a case, the PoA and ϵ-PoA was already studied and characterized in [1] and [9], respectively, and both papers pose the achievement of upper bounds on the PoS and ϵ-PoS as a major open problem in the area. For unweighted players, we show that $\mathrm{PoS} \leq 2.362$ and $\mathrm{Apx}^1_{\emptyset} \leq 37.5888$ for quadratic latency functions and that $\mathrm{PoS} \leq 3.322$ and $\mathrm{Apx}^1_{\emptyset} \leq 527.323$ for cubic latency functions. Extensions to ϵ-PoS and weighted players are left to future work.

What we would like to stress here is that, more than the novelty of the results achieved in this paper, what makes our method significative is its capability of being easily adapted to a variety of particular situations and we are more than sure of the fact that it will prove to be a powerful tool to be exploited in the analysis of the efficiency achieved by different classes of self-emerging solutions in other contexts as well. To this aim, in the full version of this paper, we show how the method applies also to other social functions, such as the maximum of

the players' payoffs, and to other subclasses of congestion games such as resource allocation games with fair cost sharing. Note that, in the latter case, as well as in the case of polynomial latency functions, the primal-dual technique proposed in [16] cannot be used, since the players' costs are not linear in the variables of the problem.

Related Works. The study of the quality of self-emerging solutions in non-cooperative systems initiated with the seminal papers of Koutsoupias and Papadimitriou [14] and Anshelevich et al. [2] which introduced, respectively, the notions of price of anarchy and price of stability.

Lot of results have been achieved since then and we recall here only the ones which are closely related to our scenario of application, that is, weighted congestion games with polynomial latency functions.

For affine latency functions and $\mathcal{F}$ defined as the sum of the players' payoffs, Christodoulou and Koutsoupias [7] show that the PoA is exactly $5/2$ for unweighted players, while Awerbuch et al. [3] show that it rises to exactly $(3+\sqrt{5})/2$ in the weighted case. These bounds keep holding also when considering the price of anarchy of generalizations of PNE such as mixed Nash equilibria and correlated equilibria, as shown by Christodoulou and Koutsoupias in [8]. Similarly, for polynomial latency functions with maximum degree equal to d, Aland et al. [1] prove that the price of anarchy of all these equilibria is exactly Φ_d^{d+1} in the weighted case and exactly $\frac{(k+1)^{2d+1}-k^{d+1}(k+2)^d}{(k+1)^{d+1}-(k+2)^d+(k+1)^d-k^{d+1}}$ in the unweighted case, where Φ_d is the unique non-negative real solution to $(x+1)^d = x^{d+1}$ and $k = \lfloor \Phi_d \rfloor$. These interdependencies have been analyzed by Roughgarden [18] who proves that unweighted congestion games with latency functions constrained in a given set belong to the class of games for which a so-called "smoothness argument" applies and that such a smoothness argument directly implies the fact that the prices of anarchy of PNE, mixed Nash equilibria, correlated equilibria and coarse correlated equilibria are always the same when $\mathcal{F}$ is the sum of the players' payoffs. Such a result has been extended also to the weighted case by Bhawalkar et al. in [4]. For the alternative model in which $\mathcal{F}$ is defined as the maximum of the players' payoffs, Christodoulou and Koutsoupias [7] show a PoA of $\Theta(\sqrt{n})$ in the case of affine latency functions.

For the PoS, only the case of unweighted players, affine latency functions and $\mathcal{F}$ defined as the sum of the players' payoffs, has been considered so far. The upper and lower bounds achieved by Caragiannis et al. [6] and by Christodoulou and Koutsoupias [8], respectively, set the PoS to exactly $1+1/\sqrt{3}$. Clearly, being the PoS a best-case measure and being the set of PNE properly contained in the set of all the other equilibrium concepts, again we have a unique bound for PNE and all of its generalizations.

As to ϵ-PNE, in the case of unweighted players, polynomial latency functions and $\mathcal{F}$ defined as the sum of the players' payoffs, Christodoulou et al. [9] show that the ϵ-PoA is exactly $\frac{(1+\epsilon)((z+1)^{2d+1}-z^{d+1}(z+2)^d)}{(z+1)^{d+1}-z^{d+1}-(1+\epsilon)((z+2)^d-(z+1)^d)}$, where z is the maximum integer satisfying $\frac{z^{d+1}}{(z+1)^d} < 1+\epsilon$, and that, for affine latency functions, the

ϵ-PoS is at least $\frac{2(3+\epsilon+\theta\epsilon^2+3\epsilon^3+2\epsilon^4+\theta+\theta\epsilon)}{6+2\epsilon+5\theta\epsilon+6\epsilon^3+4\epsilon^4-\theta\epsilon^3+2\theta\epsilon^2}$, where $\theta = \sqrt{3\epsilon^3+3+\epsilon+2\epsilon^4}$, and at most $(1+\sqrt{3})/(\epsilon+\sqrt{3})$.

Finally, for affine latency functions and $\mathcal{F}$ defined as the sum of the players' payoffs, $\text{Apx}^1_\emptyset$ has been shown to be exactly $2+\sqrt{5}$ in the unweighted case as a consequence of the upper and lower bounds provided, respectively, by Christodoulou et al. [10] and by Bilò et al. [5], while, for weighted players, Caragiannis et al. [6] give a lower bound of $3+2\sqrt{2}$ and Christodoulou et al. [10] give an upper bound of $4+2\sqrt{3}$. For $\mathcal{F}$ being the maximum of the players' payoffs, Bilò et al. [5] show that $\text{Apx}^1_\emptyset$ is $\Theta(\sqrt[4]{n^3})$ in the unweighted case and affine latency functions.

Paper Organization. In next section, we give all the necessary definitions and notation, while in Section 3 we briefly outline the primal-dual method. Then, in Section 4 we illustrate how it applies to affine latency functions and, finally, in Section 5 we use it to address the case of quadratic and cubic latency functions. Due to lack of space, some proofs are omitted and can be found in the full version of the paper.

2 Definitions

For a given integer $n > 0$, we denote as $[n]$ the set $\{1, \ldots, n\}$.

A *weighted congestion game* $\mathcal{G} = \big([n], E, (\Sigma_i)_{i\in[n]}, (\ell_e)_{e\in E}, (w_i)_{i\in[n]}\big)$ is a non-cooperative strategic game in which there is a set E of m *resources* to be shared among the players in $[n]$. Each player i has an associated weight $w_i \geq 1$ and the special case in which $w_i = 1$ for any $i \in [n]$ is called the *unweighted case*. The strategy set Σ_i, for any player $i \in [n]$, is a non-empty subset of resources, i.e., $\Sigma_i \subseteq 2^E \setminus \{\emptyset\}$. The set $\Sigma = \times_{i\in[n]}\Sigma_i$ is called the set of *strategy profiles* (or *solutions*) which can be realized in $\mathcal{G}$. Given a strategy profile $S = (s_1, s_2, \ldots, s_n) \in \Sigma$ and a resource $e \in E$, the sum of the weights of all the players using e in S, called the *congestion* of e in S, is denoted by $\mathcal{L}_e(S) = \sum_{i\in[n]:e\in s_i} w_i$. A *latency function* $\ell_e : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ associates each resource $e \in E$ with a *latency* depending on the congestion of e in S. The *cost* of player i in the strategy profile S is given by $c_i(S) = \sum_{e\in s_i} \ell_e(\mathcal{L}_e(S))$. This work is concerned only with *polynomial latency* functions of maximum degree d, i.e., the case in which $\ell_e(x) = \sum_{i=0}^{d} \alpha_{e,i}x^i$ with $\alpha_{e,i} \in \mathbb{R}_{\geq 0}$, for any $e \in E$ and $0 \leq i \leq d$.

Given a strategy profile $S \in \Sigma$ and a strategy $t \in \Sigma_i$ for player i, we denote with $(S_{-i} \diamond t) = (s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_n)$ the strategy profile obtained from S when player i changes unilaterally her strategy from s_i to t. We say that $S' = (S_{-i} \diamond t)$ is an *improving deviation* for player i in S if $c_i(S') < c_i(S)$. Given an $\epsilon \geq 0$, a strategy profile S is an *ϵ-approximate pure Nash equilibrium* (ϵ-PNE) if, for any $i \in [n]$ and for any $t \in \Sigma_i$, it holds $c_i(S) \leq (1+\epsilon)c_i(S_{-i} \diamond t)$. For $\epsilon = 0$, the set of ϵ-approximate pure Nash equilibria collapses to that of *pure Nash equilibria* (PNE), that is, the set of strategy profiles in which no player possesses an improving deviation.

Consider the social function $\textsc{Sum} : \Sigma \mapsto \mathbb{R}_{\geq 0}$ defined as the sum of the players' costs, that is, $\textsc{Sum}(S) = \sum_{i \in [n]} c_i(S)$ and let S^* be the strategy profile minimizing it. Given an $\epsilon \geq 0$ and a weighted congestion game $\mathcal{G}$, let $\mathcal{E}(\mathcal{G})$ be the set of ϵ-approximate Nash equilibria of $\mathcal{G}$. The ϵ-approximate price of anarchy of $\mathcal{G}$ is defined as $\epsilon\text{-PoA}(\mathcal{G}) = \max_{S \in \mathcal{E}(\mathcal{G})} \left\{ \frac{\textsc{Sum}(S)}{\textsc{Sum}(S^*)} \right\}$, while the ϵ-approximate price of stability of $\mathcal{G}$ is defined as $\epsilon\text{-PoS}(\mathcal{G}) = \min_{S \in \mathcal{E}(\mathcal{G})} \left\{ \frac{\textsc{Sum}(S)}{\textsc{Sum}(S^*)} \right\}$.

Given a strategy profile S and a player $i \in [n]$, a strategy profile $t^* \in \Sigma_i$ is a *best-response* for player i in S if it holds $c_i(S_{-i} \diamond t^*) \leq c_i(S_{-i} \diamond t)$ for any $t \in \Sigma_i$. Let $S^\emptyset$ be the *empty strategy profile*, i.e., the profile in which no player has performed any strategic choice yet. A one-round walk starting from the empty strategy profile is an $(n+1)$-tuple of strategy profiles $W = (S_0^W, S_1^W, \ldots, S_n^W)$ such that $S_0^W = S^\emptyset$ and, for any $i \in [n]$, $S_i^W = (S_{i-1}^W \diamond t^*)$, where t^* is a best-response for player i in S_{i-1}^W. The profile S_n^W is called the solution achieved after the one-round walk W. Clearly, depending on how the players are ordered from 1 to n and on which best-response is selected at step i when more than one best-response is available to player i in S_{i-1}^W, different one-round walks can be generated. Let $\mathcal{W}(\mathcal{G})$ denote the set of all possible one-round walks which can be generated in game $\mathcal{G}$. The approximation ratio of the solutions achieved after a one-round walk starting from the empty strategy profile in $\mathcal{G}$ is defined as $\text{Apx}_\emptyset^1(\mathcal{G}) = \max_{W \in \mathcal{W}(\mathcal{G})} \left\{ \frac{\textsc{Sum}(S_n^W)}{\textsc{Sum}(S^*)} \right\}$.

3 The Primal-Dual Technique

Fix a weighted congestion game $\mathcal{G}$, a social function $\mathcal{F}$ and a class of self-emerging solutions $\mathcal{E}$. Let $O = (s_1^O, \ldots, s_n^O)$ be the strategy profile optimizing $\mathcal{F}$ and $K = (s_1^K, \ldots, s_n^K) \in \mathcal{E}(\mathcal{G})$ be the worst-case solution in $\mathcal{E}(\mathcal{G})$ with respect to $\mathcal{F}$. For any $e \in E$, we set, for the sake of brevity, $O_e = \mathcal{L}_e(O)$ and $K_e = \mathcal{L}_e(K)$.

Since O and K are fixed, we can maximize the inefficiency yielded by the pair (K, O) by suitably choosing the coefficients $\alpha_{e,i}$, for each $e \in E$ and $0 \leq i \leq d$, so that $\mathcal{F}(K)$ is maximized, $\mathcal{F}(O)$ is normalized to one and K meets all the constraints defining the set $\mathcal{E}(\mathcal{G})$. For the sets $\mathcal{E}$ and social functions $\mathcal{F}$ considered in this paper, this task can be easily achieved by creating a suitable linear program $LP(K, O)$.

By providing a feasible solution for the dual program $DLP(K, O)$, we can obtain an upper bound on the optimal solution of $LP(K, O)$. Our task is to uncover, among all possibilities, the pair (K^*, O^*) yielding the highest possible optimal solution for $LP(K, O)$. To this aim, the study of the dual formulation plays a crucial role: if we are able to detect the nature of the "worst-case" dual constraints, then we can easily figure out the form of the pair (K^*, O^*) maximizing the inefficiency of the class of solutions $\mathcal{E}$. Clearly, by the complementary slackness conditions, if we find the optimal dual solution, then we can quite systematically construct the matching primal instance by choosing a suitable set of players and resources so as to implement all the tight dual constraints. This task is much more complicated to be achieved in the weighted case, because,

once established the values of the congestions K_e^* and O_e^* for any $e \in E$, there are still infinite many ways to split them among the players using resource e in both K and O.

4 Application to Affine Latency Functions

In order to easily illustrate our primal-dual technique, in this section we consider the well-known and studied case of affine latency functions and social function SUM and show how the results for ϵ-PoA, ϵ-PoS and $\text{Apx}_{\emptyset}^1$ already known in the literature can be reobtained in a unified manner for both weighted and unweighted players.

We say that the game $\mathcal{G}' = ([n], E', \Sigma', \ell', w)$ is equivalent to the game $\mathcal{G} = ([n], E, \Sigma, \ell, w)$ if there exists a one-to-one mapping $\varphi_i : \Sigma_i \mapsto \Sigma_i'$ for any $i \in [n]$ such that for any strategy profile $S = (s_1, \ldots, s_n) \in \Sigma$, it holds $c_i(S) = c_i(\varphi_1(s_1), \ldots, \varphi_n(s_n))$ for any $i \in [n]$.

Lemma 1. *For each weighted congestion game with affine latency functions $\mathcal{G} = ([n], E, \Sigma, \ell, w)$ there always exists an equivalent weighted congestion game with affine latency functions $\mathcal{G}' = ([n], E', \Sigma', \ell', w)$ such that $\ell'_e(x) = \alpha_{e,1}x := \alpha_e x$ for any $e \in E'$.*

Proof. Consider the weighted congestion game $\mathcal{G} = ([n], E, \Sigma, \ell, w)$ with latency functions $\ell_e(x) = \alpha_e x + \beta_e$ for any $e \in E$. For each $\widetilde{e} \in E$ such that $\beta_{\widetilde{e}} > 0$, let $N_{\widetilde{e}}$ be the set of players who can choose $\widetilde{e}$, that is, $N_{\widetilde{e}} = \{i \in [n] : \exists s \in \Sigma_i : \widetilde{e} \in s\}$. The set of resources E' is obtained by replicating all the resources in E and adding a new resource $e^i_{\widetilde{e}}$ for any $\widetilde{e} \in E$ and any $i \in N_{\widetilde{e}}$, that is, $E' = E \cup \bigcup_{\widetilde{e} \in E, i \in N_{\widetilde{e}}} \{e^i_{\widetilde{e}}\}$. The latency functions are defined as $\ell'_e(x) = \alpha_e x$ for any $e \in E' \cap E$ and $\ell'_{e^i_{\widetilde{e}}}(x) = \frac{\beta_{\widetilde{e}}}{w_i} x$ for any $\widetilde{e} \in E$ and any $i \in N_{\widetilde{e}}$. Finally, for any $i \in [n]$, the mapping φ_i is defined as follows: $\varphi_i(s) = s \cup \bigcup_{\widetilde{e} \in s} \{e^i_{\widetilde{e}}\}$. It is not difficult to see that for any $S = (s_1, \ldots, s_n) \in \Sigma$ and for any $i \in [n]$, it holds $c_i(S) = c_i(\varphi_1(s_1), \ldots, \varphi_n(s_n))$. □

As a consequence of Lemma 1, throughout this section, we restrict to latency functions of the form $\ell_e(x) = \alpha_e x$, for any $e \in E$. In such a setting, we can rewrite the social value of a strategy profile as $\text{SUM}(S) = \sum_{e \in E} (\alpha_e \mathcal{L}_e(S)^2)$.

4.1 Bounding the Approximate Price of Anarchy

By definition, we have that if K is an ϵ-PNE then, for any $i \in [n]$, it holds

$$c_i(K) = \sum_{e \in s_i^K} (\alpha_e K_e) \leq (1+\epsilon) c_i(K_{-i} \diamond s_i^O) \leq (1+\epsilon) \sum_{e \in s_i^O} (\alpha_e (K_e + w_i)).$$

Thus, the primal formulation $LP(K,O)$ assumes the following form.

$$\begin{aligned} & \textit{maximize} \sum_{e\in E}\left(\alpha_e K_e^2\right) \\ & \textit{subject to} \\ \sum_{e\in s_i^K}(\alpha_e K_e) - (1+\epsilon)\sum_{e\in s_i^O}(\alpha_e(K_e+w_i)) \le 0, & \quad \forall i\in[n] \\ \sum_{e\in E}\left(\alpha_e O_e^2\right) = 1, & \\ \alpha_e \ge 0, & \quad \forall e\in E \end{aligned}$$

The dual program $DLP(K,O)$ is

$$\begin{aligned} & \textit{minimize } \gamma \\ & \textit{subject to} \\ \sum_{i:e\in s_i^K}(y_i K_e) - (1+\epsilon)\sum_{i:e\in s_i^O}(y_i(K_e+w_i)) + \gamma O_e^2 \ge K_e^2, & \quad \forall e\in E \\ y_i \ge 0, & \quad \forall i\in[n] \end{aligned}$$

Let $\psi = \frac{1+\epsilon+\sqrt{\epsilon^2+6\epsilon+5}}{2}$ and $z = \lfloor\psi\rfloor$. For unweighted players we reobtain the upper bound proved in [9] with a much simpler and shorter proof, while for the weighted case we give the first known upper bound.

Theorem 1. *For any $\epsilon \ge 0$, it holds ϵ-PoA$(\mathcal{G}) \le \frac{(1+\epsilon)(z^2+3z+1)}{2z-\epsilon}$ when $\mathcal{G}$ has unweighted players, while it holds ϵ-PoA$(\mathcal{G}) \le \psi^2$ when $\mathcal{G}$ has weighted players.*

Proof. For the unweighted case, since $w_i = 1$ for each $i \in [n]$, by choosing $y_i = \frac{2z+1}{2z-\epsilon}$ for any $i \in [n]$ and $\gamma = \frac{(1+\epsilon)(z^2+3z+1)}{2z-\epsilon}$, the dual inequalities become of the form

$$\frac{2z+1}{2z-\epsilon}\left(K_e^2 - (1+\epsilon)(K_e+1)O_e\right) + \frac{(1+\epsilon)(z^2+3z+1)}{2z-\epsilon}O_e^2 \ge K_e^2$$

which is equivalent to

$$K_e^2 - (2z+1)(K_e O_e + O_e) + (z^2+3z+1)O_e^2 \ge 0.$$

Easy calculations (it suffices solving the inequality for K_e) show that this is always verified for any pair of non-negative *integers* (K_e, O_e). Note that the definition of z guarantees that $2z-\epsilon \ge 0$, so that the proposed dual solution is a feasible one.

For the weighted case, by choosing $y_i = \left(1+\frac{\sqrt{1+\epsilon}}{\sqrt{5+\epsilon}}\right)w_i$ for any $i \in [n]$ and $\gamma = \psi^2$, each dual inequality is verified when it holds

$$\left(1+\frac{\sqrt{1+\epsilon}}{\sqrt{5+\epsilon}}\right)\left(K_e^2 - (1+\epsilon)\left(K_e O_e + O_e^2\right)\right) + \psi^2 O_e^2 \ge K_e^2$$

which is equivalent to

$$\frac{\sqrt{1+\epsilon}}{\sqrt{5+\epsilon}}K_e^2 - \left(1+\frac{\sqrt{1+\epsilon}}{\sqrt{5+\epsilon}}\right)(1+\epsilon)(K_eO_e+O_e^2)+\psi^2O_e^2 \geq 0.$$

Easy calculations show that this is always verified for any pair of non-negative *reals* (K_e, O_e) such that $K_e, O_e \in \{0\} \cup [1,\infty)$ for any $e \in E$ (it suffices solving the inequality for K_e and noting that it has no solutions for $O_e \in \{0\} \cup [1,\infty)$ when $\epsilon \geq 0$). □

When $\epsilon = 0$, we reobtain the well-known prices of anarchy of $5/2$ and $(3+\sqrt{5})/2$ which hold for PNE in the unweighted and weighted case, respectively.

4.2 Bounding the Approximate Price of Stability

Recall that, since the ϵ-PoS is a best-case measure, the primal-dual approach guarantees a tight analysis only if we are able to exactly characterize the polyhedron defining the set of the best ϵ-PNE. It is not known how to do this at the moment, thus all the approaches used so far in the literature approximate the best ϵ-PNE with an ϵ-PNE minimizing a certain potential function. In [9], it is shown that, for unweighted players, any strategy profile S which is a local minimum of the function

$$\Phi_\epsilon(S) = \sum_{e\in E}\left(\alpha_e\left(\mathcal{L}_e(S)^2 + \frac{1-\epsilon}{1+\epsilon}\mathcal{L}_e(S)\right)\right),$$

called ϵ-approximate potential, is an ϵ-PNE. Thus, it is possible to get an upper bound on the ϵ-PoS by measuring the ϵ-PoA of the global minimum of Φ_ϵ.

We now illustrate our approach which yields the same $\frac{1+\sqrt{3}}{\epsilon+\sqrt{3}}$ upper bound achieved in [9]. Assume that K is the global minimum of Φ_ϵ. We can use the inequality $\Phi_\epsilon(K) \leq \Phi_\epsilon(O)$ which results in the constraint

$$\sum_{e\in E}\left(\alpha_e\left(K_e^2 + \frac{1-\epsilon}{1+\epsilon}K_e - O_e^2 - \frac{1-\epsilon}{1+\epsilon}O_e\right)\right) \leq 0.$$

Then, we also have $\sum_{i\in[n]}\left(\Phi_\epsilon(K) - \Phi_\epsilon(K_{-i}\diamond s_i^O)\right) \leq 0$ which results in the constraint

$$\sum_{e\in E}\left(\alpha_e\left(K_e^2 - \frac{\epsilon}{1+\epsilon}K_e - K_eO_e - \frac{1}{1+\epsilon}O_e\right)\right) \leq 0.$$

Thanks to these two constraints, the dual formulation becomes the following one.

$$\begin{aligned} &\textit{minimize}\ \gamma \\ &\textit{subject to} \\ K_e^2(y+z) + \tfrac{K_e}{1+\epsilon}(y(1-\epsilon) - z\epsilon) \\ -\left(yO_e^2 + zK_eO_e + \tfrac{O_e}{1+\epsilon}(y(1-\epsilon)+z)\right) + \gamma O_e^2 &\geq K_e^2, \forall e \in E \\ y, z &\geq 0 \end{aligned}$$

Thus, for unweighted players, we obtain the following result for any $\epsilon \in [0,1)$ (this is the only interesting case, since [9] shows that, for any $\epsilon \geq 1$, ϵ-PoS$(\mathcal{G}) = 1$ for any $\mathcal{G}$).

Theorem 2. *For any $\epsilon \in [0,1)$ and $\mathcal{G}$ with unweighted players, it holds ϵ-PoS$(\mathcal{G}) \leq \frac{1+\sqrt{3}}{\epsilon+\sqrt{3}}$.*

Proof. By choosing $y = \frac{2\epsilon+\sqrt{3}(1+\epsilon)}{2(\epsilon+\sqrt{3})}$, $z = \frac{1-\epsilon}{\epsilon+\sqrt{3}}$ and $\gamma = \frac{1+\sqrt{3}}{\epsilon+\sqrt{3}}$, the dual inequalities become

$$(\epsilon-1)((\sqrt{3}-2)K_e^2 + (2O_e - \sqrt{3})K_e + (2+\sqrt{3})(O_e - O_e^2)) \geq 0.$$

Easy calculations (it suffices solving the inequality for K_e) show that this is always verified for any pair of non-negative *integers* (K_e, O_e). □

4.3 Bounding the Approximation Ratio of One-Round Walks

For a one-round walk W, we set $K = S_n^W$. Define $K_e(i)$ as the sum of the weights of the players using resource e in K before player i performs her choice. $LP(K,O)$ in this case has the following form, where the first constraint comes from the fact that when player i enters the game and solution S_{i-1}^K is already constructed, this player picks s_i^K instead of s_i^O.

$$\begin{aligned} & \textit{maximize} \sum_{e\in E} \left(\alpha_e K_e^2\right) \\ & \textit{subject to} \\ \sum_{e\in s_i^K} (\alpha_e(K_e(i)+w_i)) - \sum_{e\in s_i^O} (\alpha_e(K_e(i)+w_i)) \leq 0, & \quad \forall i \in [n] \\ & \sum_{e\in E} \left(\alpha_e O_e^2\right) = 1 \\ & \alpha_e \geq 0, \quad \forall e \in E \end{aligned}$$

$DLP(K,O)$ is as follows.

$$\begin{aligned} & \textit{minimize } \gamma \\ & \textit{subject to} \\ \sum_{i:e\in s_i^K} (y_i(K_e(i)+w_i)) - \sum_{i:e\in s_i^O} (y_i(K_e(i)+w_i)) + \gamma O_e^2 \geq K_e^2, & \quad \forall e \in E \\ & y_i \geq 0, \quad \forall i \in [n] \end{aligned}$$

For both unweighted and weighted players we easily reobtain the upper bounds on $\text{Apx}_{\emptyset}^1$ given in [10].

Theorem 3. *For any $\mathcal{G}$ with unweighted players, it holds $Apx_{\emptyset}^1(\mathcal{G}) \leq 2+\sqrt{5}$, while for any $\mathcal{G}$ with weighted players, it holds $Apx_{\emptyset}^1(\mathcal{G}) \leq 4+2\sqrt{3}$.*

Proof. In the unweighted case, by choosing $y_i = 1 + \sqrt{5}$ for any $i \in [n]$ and $\gamma = 2 + \sqrt{5}$, since for any i such that $e \in s_i^O$ it holds $K_e(i) \leq K_e$, each dual inequality is verified when it holds

$$\left(1+\sqrt{5}\right)\left(\frac{K_e(K_e+1)}{2} - (K_e+1)O_e\right) + \left(2+\sqrt{5}\right)O_e^2 \geq K_e^2$$

which is equivalent to

$$\left(\frac{\sqrt{5}-1}{2}\right)K_e^2 + \left(\frac{1+\sqrt{5}}{2}\right)K_e - (1+\sqrt{5})K_eO_e - (1+\sqrt{5})O_e + (2+\sqrt{5})O_e^2 \geq 0.$$

Easy calculations (it suffices solving the inequality for K_e) show that this is always verified for any pair of non-negative *integers* (K_e, O_e).

In the weighted case, by choosing $y_i = \left(2 + \frac{2}{\sqrt{3}}\right) w_i$ for any $i \in [n]$ and $\gamma = 4 + 2\sqrt{3}$, since for any i such that $e \in s_i^O$ it holds $K_e(i) \leq K_e$, each dual inequality is verified when it holds

$$\left(2+\frac{2}{\sqrt{3}}\right)\left(\sum_{i\leq j : e\in s_i^K \cap s_j^K} (w_iw_j) - \sum_{i:e\in s_i^O} (w_i(K_e+w_i))\right) + \left(4+2\sqrt{3}\right)O_e^2 \geq K_e^2$$

which is true if it holds

$$\frac{1}{\sqrt{3}}K_e^2 - \left(2+\frac{2}{\sqrt{3}}\right)K_eO_e + \left(2+\frac{4}{\sqrt{3}}\right)O_e^2 \geq 0.$$

Easy calculations (it suffices solving the inequality for K_e) show that this is always verified for any pair of non-negative *reals* (K_e, O_e). □

5 Quadratic and Cubic Latency Functions

In this section, we show how to use the primal-dual method to bound PoS and $\mathrm{Apx}_{\emptyset}^1$ in the case of polynomial latency functions of maximum degree d and unweighted players. We only consider the case $d \leq 3$, that is, quadratic and cubic latency functions. It is not difficult to extend the approach to any particular value of d, but it is quite hard to obtain a general result as a function of d because we do not have simple closed formulas expressing some of the summations we need in our analysis for any value of d. We also leave the extension to ϵ-PNE and weighted players for future works. We restrict to the cases in which the latency functions are of the form $\ell_e(x) = \alpha_e x^d$, since it is possible to show that this can be supposed without loss of generality. In such a setting, [19] shows that, for unweighted players, any strategy profile S which is a local minimum of the potential function

$$\Phi(S) = \sum_{e\in E}\sum_{i=1}^{\mathcal{L}_e(S)} (\alpha_e x^d)$$

is a PNE.

5.1 Bounding the Price of Stability

For the case $d = 2$, it holds

$$\Phi(S) = \frac{1}{6} \sum_{e \in E} \left(\alpha_e \mathcal{L}_e(S)(\mathcal{L}_e(S) + 1)(2\mathcal{L}_e(S) + 1)\right).$$

Thus, the constraint $\Phi(K) \leq \Phi(O)$ becomes

$$\sum_{e \in E} \left(\alpha_e \left(K_e(K_e + 1)(2K_e + 1) - O_e(O_e + 1)(2O_e + 1)\right)\right) \leq 0$$

and the constraint $\sum_{i \in [n]} \left(\Phi(K) - \Phi(K_{-i} \diamond s_i^O)\right) \leq 0$ becomes

$$\sum_{e \in E} \left(\alpha_e \left(K_e^3 - O_e(K_e + 1)^2\right)\right) \leq 0.$$

Hence, $DLP(K, O)$ is the following.

$$\begin{aligned} & \textit{minimize } \gamma \\ & \textit{subject to} \\ & \left(y(K_e(K_e + 1)(2K_e + 1) - O_e(O_e + 1)(2O_e + 1))\right) \\ & \quad + \left(z(K_e^3 - O_e(K_e + 1)^2)\right) + \gamma O_e^3 \geq K_e^3, \forall e \in E \\ & \quad y, z \geq 0 \end{aligned}$$

Theorem 4. *For any $\mathcal{G}$ with quadratic latency functions and unweighted players, it holds $PoS(\mathcal{G}) \leq 2.362$.*

Proof. The claim follows by setting $y = 0.318$, $z = 0.453$ and $\gamma = 2.362$. □

For the case $d = 3$, it holds

$$\Phi(S) = \frac{1}{4} \sum_{e \in E} \left(\alpha_e \left(\mathcal{L}_e(S)(\mathcal{L}_e(S) + 1)\right)^2\right).$$

Thus, the constraint $\Phi(K) \leq \Phi(O)$ becomes

$$\sum_{e \in E} \left(\alpha_e \left((K_e(K_e + 1))^2 - (O_e(O_e + 1))^2\right)\right) \leq 0$$

and the constraint $\sum_{i \in [n]} \left(\Phi(K) - \Phi(K_{-i} \diamond s_i^O)\right) \leq 0$ becomes

$$\sum_{e \in E} \left(\alpha_e \left(K_e^4 - O_e(K_e + 1)^3\right)\right) \leq 0.$$

Hence, $DLP(K, O)$ is defined as follows.

$$\begin{aligned} & \textit{minimize } \gamma \\ & \textit{subject to} \\ & \left(y(K_e^2(K_e + 1)^2 - O_e^2(O_e + 1)^2)\right) \\ & + \left(z(K_e^4 - O_e(K_e + 1)^3)\right) + \gamma O_e^4 \geq K_e^4, \forall e \in E \\ & \quad y, z \geq 0 \end{aligned}$$

Theorem 5. *For any $\mathcal{G}$ with cubic latency functions and unweighted players, it holds $PoS(\mathcal{G}) \leq 3.322$.*

Proof. The claim follows by setting $y = 0.747$, $z = 0.331$ and $\gamma = 3.322$. □

By extending the instance given in [9] for lower bounding the PoS in the linear case, the following lower bounds can be easily achieved.

Theorem 6. *For any $\delta > 0$, there exist an unweighted congestion game with quadratic latency functions $\mathcal{G}_1$ and an unweighted congestion game with cubic latency functions $\mathcal{G}_2$ such that $PoS(\mathcal{G}_1) \geq 2.1859 - \delta$ and $PoS(\mathcal{G}_2) \geq 2.7558 - \delta$.*

5.2 Bounding the Approximation Ratio of One-Round Walks

For the case $d = 2$, $DLP(K, O)$ is defined as follows.

$$\begin{aligned} &\textit{minimize } \gamma \\ &\textit{subject to} \\ &\sum_{i:e\in s_i^K} \left(y_i(K_e(i)+1)^2\right) - \sum_{i:e\in s_i^O} \left(y_i(K_e(i)+1)^2\right) + \gamma O_e^3 \geq K_e^3, \ \forall e \in E \\ &y_i \geq 0, \quad \forall i \in [n] \end{aligned}$$

Theorem 7. *For any $\mathcal{G}$ with quadratic latency functions and unweighted players, it holds $Apx_{\emptyset}^1(\mathcal{G}) \leq 37.5888$.*

For the case $d = 3$, $DLP(K, O)$ is defined as follows.

$$\begin{aligned} &\textit{minimize } \gamma \\ &\textit{subject to} \\ &\sum_{i:e\in s_i^K} \left(y_i(K_e(i)+1)^3\right) - \sum_{i:e\in s_i^O} \left(y_i(K_e(i)+1)^3\right) + \gamma O_e^4 \geq K_e^4, \ \forall e \in E \\ &y_i \geq 0, \quad \forall i \in [n] \end{aligned}$$

Theorem 8. *For any $\mathcal{G}$ with cubic latency functions and unweighted players, it holds $Apx_{\emptyset}^1(\mathcal{G}) \leq \frac{17929}{34} \approx 527.323$.*

References

1. Aland, S., Dumrauf, D., Gairing, M., Monien, B., Schoppmann, F.: Exact Price of Anarchy for Polynomial Congestion Games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 218–229. Springer, Heidelberg (2006)
2. Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, E., Wexler, T., Roughgarden, T.: The Price of Stability for Network Design with Fair Cost Allocation. SIAM Journal on Computing 38(4), 1602–1623 (2008)
3. Awerbuch, B., Azar, Y., Epstein, A.: The Price of Routing Unsplittable Flow. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 57–66. ACM Press (2005)

4. Bhawalkar, K., Gairing, M., Roughgarden, T.: Weighted Congestion Games: Price of Anarchy, Universal Worst-Case Examples, and Tightness. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part II. LNCS, vol. 6347, pp. 17–28. Springer, Heidelberg (2010)
5. Bilò, V., Fanelli, A., Flammini, M., Moscardelli, L.: Performances of One-Round Walks in Linear Congestion Games. In: Mavronicolas, M., Papadopoulou, V.G. (eds.) SAGT 2009. LNCS, vol. 5814, pp. 311–322. Springer, Heidelberg (2009)
6. Caragiannis, I., Flammini, M., Kaklamanis, C., Kanellopoulos, P., Moscardelli, L.: Tight Bounds for Selfish and Greedy Load Balancing. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 311–322. Springer, Heidelberg (2006)
7. Christodoulou, G., Koutsoupias, E.: The Price of Anarchy of Finite Congestion Games. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 67–73. ACM Press (2005)
8. Christodoulou, G., Koutsoupias, E.: On the Price of Anarchy and Stability of Correlated Equilibria of Linear Congestion Games. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 59–70. Springer, Heidelberg (2005)
9. Christodoulou, G., Koutsoupias, E., Spirakis, P.G.: On the Performance of Approximate Equilibria in Congestion Games. Algorithmica (to appear)
10. Christodoulou, G., Mirrokni, V.S., Sidiropoulos, A.: Convergence and Approximation in Potential Games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 349–360. Springer, Heidelberg (2006)
11. Fotakis, D., Kontogiannis, S.C., Spirakis, P.G.: Symmetry in Network Congestion Games: Pure Equilibria and Anarchy Cost. In: Erlebach, T., Persinao, G. (eds.) WAOA 2005. LNCS, vol. 3879, pp. 161–175. Springer, Heidelberg (2006)
12. Harks, T., Klimm, M.: On the Existence of Pure Nash Equilibria in Weighted Congestion Games. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 79–89. Springer, Heidelberg (2010)
13. Harks, T., Klimm, M., Möhring, R.H.: Characterizing the Existence of Potential Functions in Weighted Congestion Games. In: Mavronicolas, M., Papadopoulou, V.G. (eds.) SAGT 2009. LNCS, vol. 5814, pp. 97–108. Springer, Heidelberg (2009)
14. Koutsoupias, E., Papadimitriou, C.: Worst-Case Equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
15. Mirrokni, V.S., Vetta, A.: Convergence Issues in Competitive Games. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) APPROX and RANDOM 2004. LNCS, vol. 3122, pp. 183–194. Springer, Heidelberg (2004)
16. Nadav, U., Roughgarden, T.: The Limits of Smoothness: A Primal-Dual Framework for Price of Anarchy Bounds. In: Saberi, A. (ed.) WINE 2010. LNCS, vol. 6484, pp. 319–326. Springer, Heidelberg (2010)
17. Panagopoulou, P.N., Spirakis, P.G.: Algorithms for Pure Nash Equilibria in Weighted Congestion Games. Journal of Experimental Algorithmics 11(2.7) (2006)
18. Roughgarden, T.: Intrinsic Robustness of the Price of Anarchy. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 513–522. ACM Press (2009)
19. Rosenthal, R.W.: A Class of Games Possessing Pure-Strategy Nash Equilibria. International Journal of Game Theory 2, 65–67 (1973)

Some Anomalies of Farsighted Strategic Behavior*

Vittorio Bilò[1], Michele Flammini[2], Gianpiero Monaco[2], and Luca Moscardelli[3]

[1] Dipartimento di Matematica e Fisica "Ennio De Giorgi", Università del Salento
Provinciale Lecce-Arnesano, P.O. Box 193, 73100 Lecce - Italy
vittorio.bilo@unisalento.it
[2] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica,
Università di L'Aquila
Via Vetoio, Loc. Coppito, 67100 L'Aquila - Italy
{michele.flammini,gianpiero.monaco}@univaq.it
[3] Dipartimento di Economia, Università di Chieti-Pescara
Viale Pindaro 42, 65127 Pescara - Italy
luca.moscardelli@unich.it

Abstract. We investigate Subgame Perfect Equilibria, that better capture the rationality of the players in sequential games with respect to other more myopic dynamics like the classical Nash one. We prove that the sequential price of anarchy, that is the worst case ratio between the social performance at a Subgame Perfect Equilibrium and the best possible one, is exactly 3 in cut and consensus games. Moreover, we improve the known $\Omega(n)$ lower bound for unrelated scheduling to $2^{\Omega(\sqrt{n})}$ and refine the corresponding upper bound to 2^n, where n is the number of players. Finally, we determine essentially tight bounds for fair cost sharing games by proving that the sequential price of anarchy is between $n+1-H_n$ and n. A surprising lower bound of $(n+1)/2$ is also determined for the singleton case.

Our results are quite interesting and counterintuitive, as they show that a farsighted behavior generally may lead to a worse performance with respect to myopic one; in fact, Nash equilibria and simple Nash rounds, consisting of a single (myopic) move per player starting from the empty state achieve a price of anarchy which result to be lower or equivalent to the sequential price of anarchy in almost all the considered cases.

1 Introduction

Modern global communication and service infrastructures (e.g., Internet, P2P, wireless ad-hoc, social networks, etc.) are increasingly introducing decentralization, autonomy, and general lack of coordination among the heterogeneous

* This research was partially supported by the PRIN 2008 research project COGENT (COmputational and GamE-theoretic aspects of uncoordinated NeTworks), funded by the Italian Ministry of University and Research.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 229–241, 2013.

network entities. The arising general mismatch between the system optimization goals and the competing users' private interests must be necessarily faced in emerging services and applications, and calls for a pressing solution of the resulting scientific and technological challenges. On this respect, useful tools and insights for modeling and analyzing the consequences of the autonomous users behavior on the system efficiency come from the integration of algorithmic ideas with techniques borrowed from Mathematical Economics and Game Theory. The fundamental approach adopted in the literature has been that of resorting on different concepts of equilibrium to characterize stable solutions consistent with the presence of rational and selfish users that have limited or no capabilities of cooperating, Nash equilibrium being among the most investigated one [26,27].

The central idea of quantifying the loss of efficiency deriving from non cooperation is that of bounding the ratio between the worst possible Nash equilibrium and the social optimal outcome. It was introduced by Koutsoupias and Papadimitriou [22], and it is commonly referred to as the Price of Anarchy (PoA) in the Computer Science literature [28]. Considerable research effort has been then devoted to bound the PoA in several non-cooperative games, including selfish routing [30], load balancing [22,17,11], linear congestion [15], fair cost sharing [4] and consensus games [6].

Often Nash equilibria may not exist or it may be hard to compute them or the time for convergence to Nash equilibria may be extremely long, even if the players always choose a best response move, i.e. a move providing them the smallest possible cost given the moves of the other players. Thus, recent research effort [13,19,20,25] focused on the evaluation of the performance after a limited number of selfish moves or a bounded number of rounds, with a round consisting of a sequence of best response moves, with each user moving exactly once.

As far as the specific games considered in this paper are concerned, in cut games Nash equilibria correspond to local optima of the classical local search algorithm. Moreover, a single round starting from the empty state in which every player has not selected any strategy coincides with a possible execution of the basic greedy algorithm. As a consequence, in both cases the price of anarchy is upper bounded by 2 and simple counterexamples show that this result is strict. Finally, the PoA after a single round from a generic state is $\Omega(n)$ [16]. In unrelated scheduling, it is known that the PoA of Nash (and thus of single rounds from a generic state) is unbounded, while it can be trivially verified that single rounds from the empty state have a PoA equal to the number of players n. Finally, the PoA of Nash equilibria in fair allocation games, that is using the basic Shapley cost sharing method [31] in which the cost of each resource is equally split among the allocated players, again is exactly n, while in case of single rounds it is $O(\log^2 n)$ from the empty state [12] and $n(n+1)/2$ from a generic state [8].

One drawback of Nash equilibria and of the corresponding dynamics is that they often have disappointing performances: this has stimulated considerable research attempts in studying other reasonable solution concepts, like approximate [14] and strong Nash equilibria [1], able to achieve better performances.

Examples about strong equilibria (approximate equilibria, respectively) can be found in [21] ([7], respectively) for cut games, in [5] for unrelated machine scheduling games and in [2,18] ([3], respectively) for cost sharing games.

Equilibrium solutions alternative to Nash can be defined according to suitable extensions of the agents rationality. In particular, considerable research effort focused on sequential games, modeling the strategic behavior of agents who anticipate future strategic opportunities. More precisely, in the majority of equilibria notions it is assumed that players simultaneously select their strategic choices. Even when speaking about speed of convergence and best response moves, the corresponding dynamics is actually the result of a myopic interaction among the players, in which each player merely selects the strategy being at the moment a good choice, without caring about the future of the game. In other words, the dynamics is not governed by farsighted strategic choices of the players. As a consequence, in the sequential setting the Subgame Perfect Equilibrium (SPE) [29], better capturing the sequential rationality of the players, is the basic used equilibrium notion. Very recently, such equilibria have been investigated in the context of auctions [24], cut, consensus, unrelated scheduling and fair cost sharing allocation games [23]. A desired and expected effect of SPEs remarked in [23] is that the corresponding farsighted choices may reduce the induced price of anarchy. Other non myopic extensions induced by more farsighted agents that take into account the long term effects of their adopted strategies were considered in [10].

In [23] the authors considered sequential games and measure their Sequential Price of Anarchy (SPoA), that is, the price of anarchy of the corresponding Subgame Perfect Equilibria. In particular, they proved that the SPoA in cut games is between 2 and 4, while in unrelated machine scheduling it is at least $\Omega(n)$ and at most $m \cdot 2^n$, where n is the number of players and m the number of machines. Finally, for fair cost sharing allocation games the authors just considered the singleton case in which each player can select a single resource and proved that under some restricted assumption the SPoA is exactly $H_n = O(\log n)$.

Motivated by the above reasons, and by following the way marked out in [23], we consider some fundamental games in algorithmic game theory. More precisely, we prove that the sequential price of anarchy is exactly 3 in cut and consensus games. Moreover, we improve the previously known lower bound for unrelated scheduling to $2^{\Omega(\sqrt{n})}$ and refine the corresponding upper bound to 2^n. Finally, we determine essentially tight bounds for fair cost sharing games by proving that the price of anarchy is between $n + 1 - H_n$ and n. A surprising lower bound of $(n + 1)/2$ is also determined for the singleton case.

Our results are quite interesting, counterintuitive and in some sense disappointing, as they put the expected performances of SPEs back in their right perspective. In fact, they show that farsighted choices may lead to a worse performance with respect to those yielded by a myopic behavior, as the price of anarchy of Nash equilibria and simple Nash rounds from the empty state happens to be lower than the sequential price of anarchy in almost all of the considered games.

The paper is organized as follows. In the next section we give the basic definitions and notation. In Sections 3, 4, 5 we show our results concerning SPE for cut and consensus games, unrelated machine scheduling and fair cost sharing games, respectively. Finally in Section 6 we give some conclusive remarks and discuss some interesting open questions.

2 Definitions and Notation

For an integer $n > 0$, denote as $[n]$ the set $\{1, 2, \ldots, n\}$.

A **sequential game** is a triple $\mathsf{G} = (N, A_{i \in N}, U_{i \in N})$, where $N = \{1, \ldots, n\}$ is a set of n *players* and, for any $i \in N$, A_i is the *set of actions* for player i and $U_i : \times_{j \in N} A_j \mapsto \mathbb{R}_{\geq 0}$ is her *utility function*. The game is played in n steps. At the ith step, player i observes the actions chosen by the first $i-1$ players and decides which action to adopt. The function $s_i : \times_{1 \leq j < i} A_j \mapsto A_i$ is the *strategy* of player i. A *strategy profile* $\boldsymbol{s} = (s_1, \ldots, s_n)$ is an n-tuple of strategies, one for each player. Each strategy profile $\boldsymbol{s}$ induces a unique *outcome* $\boldsymbol{a}(\boldsymbol{s}) = (a_1(\boldsymbol{s}), \ldots, a_n(\boldsymbol{s}))$, defined as $a_1(\boldsymbol{s}) = s_1(\emptyset)$ and $a_i(\boldsymbol{s}) = s_i(a_1(\boldsymbol{s}), \ldots, a_{i-1}(\boldsymbol{s}))$ for any $2 \leq i \leq n$.

For a strategy profile $\boldsymbol{s}$ and $i \in [n-1]$, let $\boldsymbol{s}_{>i}$ be the restriction of $\boldsymbol{s}$ to the players $i+1, \ldots, n$ and, for any $i \leq j \leq n$, let $\boldsymbol{s}_{\geq i}^{<j}$ be the restriction of $\boldsymbol{s}$ to the players $i, \ldots, j-1$, with $\boldsymbol{s}_{\geq i}^{<j} = \emptyset$ for $i = j$. Denote by $H_i(\mathsf{G}) = \times_{1 \leq j < i} A_j$ and $H(\mathsf{G}) = \bigcup_{i=1}^{n} H_i$ the sets of *histories* of G up to player $i-1$ and the set of all histories of G, respectively. Any history $h \in H(\mathsf{G})$ naturally induces a subgame $\mathsf{G}_i(h)$ in which players $i, \ldots, n$ have to sequentially move, where i is such that $h \in H_i(\mathsf{G})$. The outcome $\boldsymbol{a}(\boldsymbol{s}_{>i}) = (a_i(\boldsymbol{s}_{>i}), \ldots, a_n(\boldsymbol{s}_{>i}))$ induced by $\boldsymbol{s}_{>i}$ in $\mathsf{G}_i(h)$ is then defined as $a_i(\boldsymbol{s}_{>i}) = s_i(h)$ and $a_j(\boldsymbol{s}_{>i}) = s_j(a_i(\boldsymbol{s}_{>i}), \ldots, a_{j-1}(\boldsymbol{s}_{>i}))$ for any $i < j \leq n$, while the outcome $\boldsymbol{a}(\boldsymbol{s}_{\geq i}^{<j}) = (a_i(\boldsymbol{s}_{\geq i}^{<j}), \ldots, a_{j-1}(\boldsymbol{s}_{\geq i}^{<j}))$ induced by $\boldsymbol{s}_{\geq i}^{<j}$ in $\mathsf{G}_i(h)$ is then defined as $a_i(\boldsymbol{s}_{\geq i}^{<j}) = s_i(h)$ and $a_k(\boldsymbol{s}_{\geq i}^{<j}) = s_k(a_i(\boldsymbol{s}_{\geq i}^{<j}), \ldots, a_{k-1}(\boldsymbol{s}_{\geq i}^{<j}))$ for any $i < k < j-1$.

Given any history $h \in H(\mathsf{G})$ such that it also holds that $h \in H_i(\mathsf{G})$, a strategy profile $\boldsymbol{s}$ is a **Nash equilibrium** (NE) for the subgame $\mathsf{G}_i(h)$ if, for any $i \leq j \leq n$, it holds $U_j(h, \boldsymbol{a}(\boldsymbol{s}_{\geq i}^{<j}), s_j(h, \boldsymbol{a}(\boldsymbol{s}_{\geq i}^{<j})), \boldsymbol{a}(\boldsymbol{s}_{>j})) \geq U_j(h, \boldsymbol{a}(\boldsymbol{s}_{\geq i}^{<j}), t, \boldsymbol{a}(\boldsymbol{s}_{>j}))$ for any $t \in A_j$. A strategy profile $\boldsymbol{s}$ is a **subgame perfect equilibrium** (SPE) for G if it is a Nash equilibrium for any $h \in H(\mathsf{G})$.

These definitions are intended for utility games, that is, games in which players want to maximize their utility. Definitions for cost games can be obtained by reversing the inequality relationships.

Sequential games are special cases of Extensive Form games and can be graphically represented as a decision tree in which each level i models the ith step of the game. Leaves correspond to action profiles, while each node u at level i corresponds to the subgame $\mathsf{G}_i(h_u)$, where $h_u \in H_i$ is the $(i-1)$-tuple of actions defined by the path from the root to node u. Hence, a strategy profile $\boldsymbol{s}$ is a SPE if it is a NE for all the subgames induced by all the nodes in the tree. Each sequential game G admits at least one subgame perfect equilibrium and its set of subgame perfect equilibria $\mathcal{SPE}(\mathsf{G})$ can be easily found by backward induction (see Chapter 6 of [29] for further details).

Given a *social function* $\mathsf{SF} : \times_{i \in N} A_i \mapsto \mathbb{R}_{>0}$, let $\boldsymbol{a}^* = (a_1^*, \ldots, a_n^*) \in \times_{i \in N} A_i$ be the outcome optimizing SF. The **sequential price of anarchy** (SPoA) of G is defined as $\mathrm{SPoA}(\mathsf{G}) = \max_{\boldsymbol{s} \in \mathcal{SPE}(\mathsf{G})} \frac{\mathsf{SF}(\boldsymbol{a}^*)}{\mathsf{SF}(\boldsymbol{a}(\boldsymbol{s}))}$ when SF has to be maximized and $\mathrm{SPoA}(\mathsf{G}) = \max_{\boldsymbol{s} \in \mathcal{SPE}(\mathsf{G})} \frac{\mathsf{SF}(\boldsymbol{a}(\boldsymbol{s}))}{\mathsf{SF}(\boldsymbol{a}^*)}$ when SF has to be minimized. Thus, the sequential price of anarchy always falls in the interval $[1; \infty)$.

Given an action profile $\boldsymbol{a} = (a_1, \ldots, a_n)$, define $\boldsymbol{a}_{<i} = (a_1, \ldots, a_{i-1})$, with $\boldsymbol{a}_{<i} = \emptyset$. In the proofs of our lower bounds for the sequential price of anarchy, we will make extensive use of the following characterization. An action profile $\boldsymbol{a}$ is an outcome of some SPE $\boldsymbol{s}$ if, for any $i \in [n-1]$ and for any $t \in A_i$, there exists a SPE $\boldsymbol{s}(\boldsymbol{a}_{<i}, t)$ for the subgame $\mathsf{G}_i(\boldsymbol{a}_{<i}, t)$ such that $U_i(\boldsymbol{a}) \geq U_i(\boldsymbol{a}_{<i}, t, \boldsymbol{a}(\boldsymbol{a}_{<i}, t))$.

3 Cut and Consensus Games

In cut and consensus games there are n players corresponding to the vertices of an edge weighted graph $G = (V, E, w)$ with $w : E \mapsto \mathbb{R}_{>0}$. Players can choose between two sides: *left* and *right*. In cut games, each player i gets a utility equal to the sum of the weights of all edges incident to i and to some other player belonging to the opposite side. We represent an history up to player i, for any $i \in [n]$, as a pair (L_i, R_i), where $L_i \subseteq \{1, \ldots, i\}$ is the subset of the first i players choosing the left side and $R_i \equiv \{1, \ldots, i\} \setminus L_i$ is the subset of the first i players choosing the right side. Thus, a pair (L_n, R_n) denotes an action profile. For an action profile (L_n, R_n), the social function is defined as the sum of the players' utilities that corresponds to twice the sum of the weights of the edges in the cut defined by (L_n, R_n).

It has been shown in [23] that the sequential price of anarchy of cut games is a value between 2 and 4 and it is also conjectured that it is exactly 2, i.e., the given lower bound is tight. We improve both upper and lower bounds, thus disproving this conjecture, by showing that the sequential price of anarchy of cut games is 3. This result holds for games played on both weighted and unweighted graphs and is achieved by proving an upper bound of 3 for games played on weighted graphs and a lower bound which tends to 3 when the number of players goes to infinity for games played on unweighted graphs.

Theorem 1. *For any cut game* CG*, it holds* $SPoA(\mathsf{CG}) \leq 3$.

Proof. Fix a cut game CG played on and a weighted graph $G = (V, E, w)$ and an outcome $\mathcal{C} = (L := L_n, R := R_n)$ such that $\mathcal{C} = \boldsymbol{a}(\boldsymbol{s})$ for some SPE $\boldsymbol{s}$ of CG. Let $\mathsf{OPT}(\mathsf{CG})$ be the social optimum for CG and denote as $E_L = \{\{i, j\} \in E : i, j \in L\}$ and $E_R = \{\{i, j\} \in E : i, j \in R\}$ the set of all edges connecting pairs of nodes belonging to L and R, respectively. For the sake of simplicity in the notation, in the sequel of this proof we set $w_{ij} = 0$ for any $\{i, j\} \notin E$, so that we can assume that G is complete. It is easy to see that this assumption does not affect the analysis of the sequential price of anarchy. At any possible step $i \in [n]$, if a player $i \in L$ deviates to the right side, since all players $j < i$ have already moved and cannot change their strategy, she is guaranteed to get a

utility of at least $\sum_{j \in L: j<i} w_{ij}$ in any possible outcome generated by this choice. Thus, since $\boldsymbol{s}$ is a SPE, for any player $i \in L$, it holds

$$\sum_{j \in R} w_{ij} \geq \sum_{j \in L: j<i} w_{ij}. \tag{1}$$

Similarly, for any player $i \in R$, it holds

$$\sum_{j \in L} w_{ij} \geq \sum_{j \in R: j<i} w_{ij}. \tag{2}$$

By summing up (1) for any $i \in L$ and (2) for any $i \in R$, we get

$$\sum_{\{i,j\} \in \mathcal{C}} w_{ij} \geq \sum_{\{i,j\} \in E_L} w_{ij} \tag{3}$$

and

$$\sum_{\{i,j\} \in \mathcal{C}} w_{ij} \geq \sum_{\{i,j\} \in E_R} w_{ij}. \tag{4}$$

By summing (3) and (4), we get

$$2 \sum_{\{i,j\} \in \mathcal{C}} w_{ij} \geq \sum_{\{i,j\} \in E_L} w_{ij} + \sum_{\{i,j\} \in E_R} w_{ij} = \sum_{\{i,j\} \in E} w_{ij} - \sum_{\{i,j\} \in \mathcal{C}} w_{ij}$$

which yields

$$3 \sum_{\{i,j\} \in \mathcal{C}} w_{ij} \geq \sum_{\{i,j\} \in E} w_{ij} \geq \mathsf{OPT}(\mathsf{CG}).$$

□

Theorem 2. *For any $\epsilon > 0$, there exists a cut game CG played on an unweighted graph such that $SPoA(\mathsf{CG}) \geq 3 - \epsilon$.*

Proof. Consider the cut game CG_k played on the unweighted graph $G_k = (V, E)$ depicted in Figure 1 where players move according to the lexicographic order of their labels, that is, the order $a, b_1, \ldots, b_k, c_1, \ldots, c_k, d, e_1, e_2$. Since G_k is bipartite, there exists an action profile, namely $(\{a, d\}, \{b_1, \ldots, b_k, c_1, \ldots, c_k, e_1, e_2\})$, of social value $|E| = 3k + 4$. We now show that there exists a SPE $\boldsymbol{s}$ for CG_k such that $\boldsymbol{a}(\boldsymbol{s}) = \mathcal{C}$, where $\mathcal{C} = (\{a, b_1, \ldots, b_k, e_1, e_2\}, \{c_1, \ldots, c_k, d\})$ for a social value of $k + 2$.

Consider the subgame yielded by the history $(\{a, (b_i)_{i \in [k]}\}, \{(c_i)_{i \in [k]}, d\})$. If player e_i, with $i \in [2]$, chooses the right side, she gets utility 1, thus, since it holds $U_{e_i}(\mathcal{C}) = 1$ for any $i \in [2]$, none of them has an incentive to deviate from the left side.

Consider the subgame yielded by the history $(\{a, (b_i)_{i \in [k]}\}, \{(c_i)_{i \in [k]}\})$. If player d chooses the left side, she gets utility of at most $k + 2$, thus, since it holds $U_d(\mathcal{C}) = k + 2$, d has no incentive to deviate from the right side.

For any index $j \in [k]$, consider the subgame CG'_k yielded by the history $(\{a, (b_i)_{i\in[k]}, c_j\}, \{(c_i)_{i\in[k]\setminus\{j\}}\})$. We claim that there exists a SPE $\boldsymbol{s}'$ for CG'_k such that $\boldsymbol{a}(\boldsymbol{s}') = \mathcal{C}_j$, where $\mathcal{C}_j = (\{a, (b_i)_{i\in[k]}, c_j, d\}, \{(c_i)_{i\in[k]\setminus\{j\}}, e_1, e_2\})$. For each $i \in [k] \setminus \{j\}$, it holds $U_{c_i}(\mathcal{C}_j) = 1$ which is the maximum utility that any player c_i, with $i \in [k] \setminus \{j\}$, can achieve in any possible action profile, thus none of them has an incentive to deviate to the left side. If player d chooses the right side, she gets utility of at most $k + 1$, thus, since it holds $U_d(\mathcal{C}_j) = k + 1$, d has no incentive to deviate from the left side. Finally, both players e_1 and e_2 prefers to be on the right side when both a and d are on the left one, and this shows that $\boldsymbol{s}'$ is a SPE for CG'_k. Now, since it holds $U_{c_j}(\mathcal{C}_j) = 0 = U_{c_j}(\mathcal{C})$, it follows that no player c_j, with $j \in [k]$, has an incentive to deviate from the right side.

For any set of indexes $J \subseteq [k]$, consider the subgame CG'_k yielded by the history $(\{a, (b_i)_{i\notin J}\}, \{(b_i)_{i\in J}\})$. We claim that there exists a SPE $\boldsymbol{s}'$ for CG'_k such that $\boldsymbol{a}(\boldsymbol{s}') = \mathcal{C}_J$, where $\mathcal{C}_J = (\{a, (b_i)_{i\notin J}, (c_i)_{i\in[k]}, e_1, e_2\}, \{(b_i)_{i\in J}, d\})$. For each $i \in [k]$, it holds $U_{c_i}(\mathcal{C}_J) = 1$ which is the maximum utility that any player c_i can achieve in any possible action profile, thus none of them has an incentive to deviate to the right side. If player d chooses the left side, she gets utility of at most $k + 2$, thus, since it holds $U_d(\mathcal{C}_J) = k + 2$, d has no incentive to deviate from the right side. Finally, both sides are equivalent for players e_1 and e_2, thus showing that $\boldsymbol{s}'$ is a SPE for CG'_k. Now, since for any $i \in J$ it holds $U_{b_i}(\mathcal{C}_J) = 1 = U_{b_i}(\mathcal{C})$, by the arbitrariness of J, it follows that no player b_i, with $i \in [k]$, has an incentive to deviate from the left side.

Finally, consider the subgame CG'_k yielded by the history $(\emptyset, \{a\})$. By a simple symmetric argument, it follows that there exists a SPE $\boldsymbol{s}'$ for CG'_k such that $\boldsymbol{a}(\boldsymbol{s}') = \mathcal{C}'$ where $\mathcal{C}' = (\{(c_i)_{i\in[k]}, d\}, \{a, (b_i)_{i\in[k]}, e_1, e_2\})$. Since $U_a(\mathcal{C}') = U_a(\mathcal{C})$, it follows that $\boldsymbol{s}$ is a SPE for CG_k.

The claim then follows by taking the limit for k going to infinity of the ratio $\frac{3k+4}{k+2}$. □

Consensus games were defined in [23] as cost games in which the cost of a player i is defined as the sum of the weights of all edges incident to i and to some other player belonging to the opposite side. The authors of [23] show that, when all the edges have non-zero weights and G is a complete graph, there is a unique SPE whose outcome coincides with the optimal solution in which all the players choose the same side. In the general case, however, things are quite different as witnessed by the simple consensus game played on the unweighted graph $G = (V = \{1, 2, 3\}, E = \{\{1, 3\}, \{2, 3\}\})$. Any strategy profile in which player 2 always chooses the opposite side of player 1 and player 3 always chooses the same side of player 2 is a SPE yielding a non-optimal outcome. Thus, in order to study the sequential price of anarchy of consensus games played on general graphs and circumvent the problem created by the existence of a social optimum of zero cost, we redefine consensus games as utility games in which the utility of a player i is defined as the sum of the weights of all edges incident to i and to some other player belonging to the same side.

It is not difficult to mimic the proof of Theorem 1 to show that, for any consensus game CG, it holds $\mathrm{SPoA}(\mathsf{CG}) \leq 3$. Similarly, the instance graph CG_k

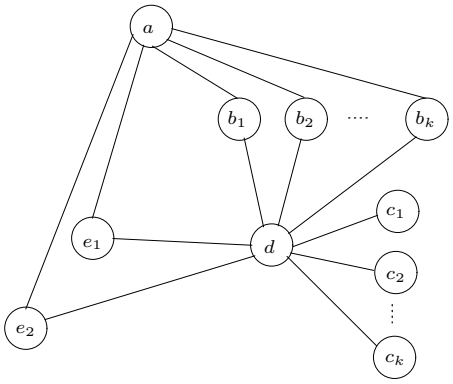


Fig. 1. The unweighted graph G_k yielding a sequential price of anarchy for cut games approaching 3 as k goes to infinity

depicted on Figure 1 can be used to prove that, for any $\epsilon > 0$, there exists a consensus game CG played on an unweighted graph such that $\text{SPoA}(\mathsf{CG}) \geq 3-\epsilon$. In particular, such a last result can be achieved by showing that there exists a SPE $\boldsymbol{s}$ for CG_k such that $\boldsymbol{a}(\boldsymbol{s}) = (\{a, (c_i)_{i\in[k]}\}, \{(b)_{i\in[k]}, (d)_{i\in[k]}, e_1, e_2\})$.

4 Unrelated Machine Scheduling

In this section we address the problem (also considered in [23]) of scheduling n jobs on m unrelated machines belonging to set M. Each job is owned by a selfish player, and $t_{x,y}$ is the processing time of job x on machine y. A job assignment is a function $\phi : N \to M$ such that $\phi(x) = y$ when job x is assigned to machine y. Given a machine y and a job assignment ϕ, the completion time of y is given by $\sum_{x:\phi(x)=y} t_{x,y}$. Each player aims at minimizing the completion time of her selected machine, and the social function to be minimized is defined as the makespan of a job assignment, that is the maximum completion time over all machines ($\max_{y\in M} \sum_{x:\phi(x)=y} t_{x,y}$).

In [23], the authors proved that the sequential price of anarchy of such a game is $m \cdot 2^n$, and also exhibited a family of unrelated machine scheduling games having sequential price of anarchy $\Omega(n)$. On the one hand, we would like to notice that by refining their proof it is possible to show that a slightly better upper bound of 2^n holds. On the other hand, we significantly improve the lower bound to $2^{\Omega(\sqrt{n})}$.

Theorem 3. *For any unrelated machine scheduling game* UMSG*, it holds* $SPoA(\mathsf{UMSG}) \leq 2^n$.

Proof. The proof is a refinement of the one in [23] (proof of Theorem 4). Let $\overrightarrow{L_0} \in \mathbb{R}_+^M$ be a vector representing an initial load on each of the machines,

$\textsc{Spe}(\overrightarrow{L_0}, k)$ be the makespan of the subgame perfect equilibrium arising when players $k, k+1, \ldots, n$ play starting from load $\overrightarrow{L_0}$. Moreover, let $t_x^* = \min_{y \in M} t_{x,y}$.

We now prove by induction on k that

$$\forall \overrightarrow{L_0} \in \mathbb{R}_+^M, \textsc{Spe}(\overrightarrow{L_0}, k) \leq \|\overrightarrow{L_0}\|_\infty + \sum_{j=k}^{n} 2^{j-k} t_j^*.$$

In fact, by taking $\overrightarrow{L_0} = \overrightarrow{0}$ and $k = 1$, since the optimal makespan is at least $\max_{i \in \{1,\ldots,n\}} t_i^*$, we obtain $\textsc{Spe}(\overrightarrow{0}, 1) \leq \sum_{j=1}^{n} 2^{j-k} t_j^* \leq \sum_{j=1}^{n} 2^{j-k} \max_{i \in \{1,\ldots,n\}} t_i^* = (\max_{i \in \{1,\ldots,n\}} t_i^*) \sum_{j=1}^{n} 2^{j-k} \leq \textsc{Opt} \cdot 2^n$. It remains to show the induction. The basis of the induction for $k = n$ is clearly verified. For any $k = n-1, \ldots, 1$, suppose that the induction hypothesis holds for $k+1$. Player k has the option of playing the machine in which he has load t_k^*. Let $\overrightarrow{L_1}^*$ be the load on the machines after such a move. It holds that $\|\overrightarrow{L_1}^*\|_\infty \leq t_k^* + \|\overrightarrow{L_0}\|_\infty$. Moreover, by the induction hypothesis, the makespan (and therefore also player k's cost) in the end of the game is at most $\|\overrightarrow{L_1}^*\|_\infty + \sum_{j=k+1}^{n} 2^{j-k-1} \cdot t_j^*$. Let y be the machine chosen by k at equilibrium (given the actions of players $1, \ldots, k-1$), and L_1 the load vector after k chooses machine y. It holds that

$$\begin{aligned}
\|\overrightarrow{L_1}\|_\infty &\leq \max \left\{ \|\overrightarrow{L_0}\|_\infty, \|\overrightarrow{L_1}^*\|_\infty + \sum_{j=k+1}^{n} 2^{j-k-1} \cdot t_j^* \right\} \\
&\leq \max \left\{ \|\overrightarrow{L_0}\|_\infty, t_k^* + \|\overrightarrow{L_0}\|_\infty + \sum_{j=k+1}^{n} 2^{j-k-1} \cdot t_j^* \right\} \\
&\leq t_k^* + \|\overrightarrow{L_0}\|_\infty + \sum_{j=k+1}^{n} 2^{j-k-1} \cdot t_j^*.
\end{aligned}$$

By again applying the induction hypothesis, we finally obtain

$$\begin{aligned}
\textsc{Spe}(\overrightarrow{L_0}, k) &= \textsc{Spe}(\overrightarrow{L_1}, k+1) \\
&\leq \|\overrightarrow{L_1}\|_\infty + \sum_{j=k+1}^{n} 2^{j-k-1} \cdot t_j^* \\
&\leq t_k^* + \|\overrightarrow{L_0}\|_\infty + 2 \sum_{j=k+1}^{n} 2^{j-k-1} \cdot t_j^* \\
&= \|\overrightarrow{L_0}\|_\infty + \sum_{j=k}^{n} 2^{j-k} \cdot t_j^*.
\end{aligned}$$

□

Theorem 4. *For any $n \geq 2$, there exists an unrelated machine scheduling game UMSG_n with n players such that $SPoA(\mathsf{UMSG}_n) = 2^{\Omega(\sqrt{n})}$.*

5 Fair Cost Sharing Games

In this section we consider fair cost sharing games in which we are given a set N of n players and a ground set R of resources. Each resource $r \in R$ has a cost $c(r)$. For each player $i \in N$, the set of actions A_i can be any family of subsets of resources. We will refer to *singleton* fair cost sharing games when any action for each player is a single resource. For an action profile $\boldsymbol{a} = (a_1, \ldots, a_n)$, let $n_r = |\{j \in N : r \in a_j\}|$ be the number of players choosing a resource $r \in R$ in $\boldsymbol{a}$. The cost of a player $i \in N$ in $\boldsymbol{a}$ is then $U_i(\boldsymbol{a}) = \sum_{r \in a_i} \frac{c(r)}{n_r}$.

This class of game was introduced by Anshelevich et al. [4] who studied simultaneous move game and showed that the price of anarchy is n under the social cost function $\mathsf{SF}(\boldsymbol{a}) = \sum_{i \in N} U_i(\boldsymbol{a})$.

It is useful at this point to note that for any fair cost sharing game CSG_n with n players, it also holds that $SPoA(\mathsf{CSG}_n) \leq n$. In fact, for any $i \in N$ and for any $t \in A_i$, let $c_t = \sum_{r \in t} c(r)$ and $c_i^* = \min_{t \in A_i}\{c_t\}$. Denote $c_{max}^* = \max_{i \in N}\{c_i^*\}$. Clearly, for any SPE $\boldsymbol{s}$, it holds $U_i(\boldsymbol{a}(\boldsymbol{s})) \leq c_i^*$ for any $i \in N$, which yields $\sum_{i \in N} U_i(\boldsymbol{a}(\boldsymbol{s})) \leq n \cdot c_{max}^*$, while the optimal outcome $\boldsymbol{a}^*$ is such that $\sum_{i \in N} U_i(\boldsymbol{a}^*) \geq c_{max}^*$.

In the next theorem, we give an almost matching lower bound.

Theorem 5. *For any $n \geq 2$, there exists a fair cost sharing game CSG_n with n players such that $SPoA(\mathsf{CSG}_n) \geq n + 1 - H_n$.*

Recently it has been shown in [23] that, when considering the singleton version of the game, the sequential price of anarchy is bounded by $H_n = O(\log n)$ under the generic cost assumption, where resources have generic costs if $c(r)/k \neq c(r')/k'$ for any pair of resources $r, r' \in R$ and any $1 \leq k, k' \leq n$. Authors also claimed that when costs are not generic this bound still holds. We prove here that this claim is not correct and that the generic cost assumption is a necessary condition in order to get the H_n upper bound, by showing that the sequential price of anarchy of singleton fair cost sharing games is at least $\frac{n+1}{2}$.

Theorem 6. *For each $n \geq 2$, there exists a singleton fair cost sharing game CSG_n such that $SPoA(\mathsf{CSG}_n) = \frac{n+1}{2}$.*

Proof. For a fixed $n \geq 2$, we define CSG_n as follows (see Figure 2). There are n players and a set $R = \{X_1, \ldots, X_n, Y_1, \ldots, Y_n\}$ of $2n$ resources, where, for each $i \in [n]$, it holds $c(X_i) = c(Y_i) = n - i + 1$. For each player $i \in [n]$, the set of actions is $A_i = \{Y_i, \{X_j\}_{j \in [i]}\}$. Note that, for any $i \in [n]$, resource X_i can be shared by at most $n - i + 1$ players. It follows that, in any possible outcome of the game, none of the players can ever pay a cost smaller than 1. Hence, we can correctly assume that in any strategy profile $\boldsymbol{s}$ such that $U_i(\boldsymbol{a}(\boldsymbol{s})) = 1$, player i has no incentive in migrating to a different strategy.

We now show that the strategy profile $\boldsymbol{s}$ such that $\boldsymbol{a}(\boldsymbol{s}) = \bigcup_{i=1}^{n}\{Y_i\}$ of total cost $\sum_{i=1}^{n}(n - i + 1) = \frac{n(n+1)}{2}$ is a SPE for CSG_n. Since the outcome in which all players choose X_1 has total cost n, it will follow that $\mathrm{SPoA}(\mathsf{CSG}_n) \geq \frac{n+1}{2}$.

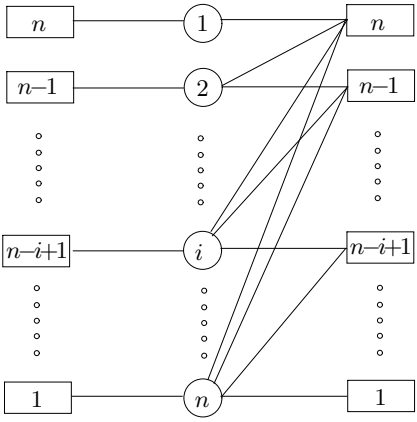


Fig. 2. The singleton fair cost sharing game CSG_n yielding a sequential price of anarchy $\frac{n+1}{2}$

The proof is by induction on i. For the base case of $i = 1$, we have that player 1 has two choices, namely, X_1 and Y_1. If she chooses X_1, consider the action profile in which all the other $n-1$ players choose X_2. Since $c(X_2) = n-1$, it follows that all these players get a final cost of 1. Hence, when player 1 chooses X_1, she ends up paying n which is the same that she pays by choosing Y_1 and this completes the proof of the base case.

Now for a player $i > 1$ assume, for the sake of induction, that all the previous players $j \in [i-1]$ have chosen resource Y_j. When $i = n$, player n can choose between X_n and Y_n. Since, both choices give her the same cost of 1, the claim follows. When $i < n$, player i can choose any resource in the set $\{X_1, \ldots, X_i\}$. Independently of her choice, consider the action profile in which all the other $n-i$ players choose X_{i+1}. Since $c(X_{i+1}) = n-i$, it follows that all these players get a final cost of 1. Hence, when player i chooses X_j, with $j \in [i]$, she ends up paying at least $n-i+1$ which is the same that she pays by choosing Y_i and this completes the induction. □

6 Conclusions

We have investigated Subgame Perfect Equilibria (SPE), that better capture the rationality of the players in sequential games, and proven optimal bounds on the sequential price of anarchy for cut and consensus games and asymptotically optimal bounds for fair cost sharing games. Moreover, we have improved the known bounds for unrelated machine scheduling.

Our results are quite surprising and put the expected performances of SPE back in their right perspective, as they show that a myopic behavior can lead to a better performance. This suggests the adoption and investigation of reasonable variants of SPE exhibiting better performances and removing some excessively

strong assumptions that to our opinion can limit their applicability, like the fact that every player has full information of the actions selected by every other players in each possible state of the game.

As another open question, it would be nice to reduce the gaps on the sequential price of anarchy for some of the considered games, like $2^{\Omega(\sqrt{n})} \div 2^n$ in unrelated scheduling and the $n/2 \div n$ in singleton fair allocation. Moreover, it would be worth to consider other classical games so far investigated only in more myopic settings. Furthermore, it would be interesting to consider the effect of a limited farsighted behavior, Nash equilibria and SPE being the two opposite extremes.

Finally, considering SPE in the framework of [9], in which graphical games (i.e., games in which each player is only aware of the existence of the players she is connected to in a given social knowledge graph) are analysed, constitutes an interesting research direction.

References

1. Aumann, R.J.: Acceptable points in general cooperative n-person games. In: Tucker, A.W., Luce, R.D. (eds.) Contributions to the Theory of Games. Ann. of Math. Stud., vol. 40, pp. 287–324. Princeton University Press, Princeton (1959)
2. Albers, S.: On the Value of Coordination in Network Design. SIAM Journal on Computing 38(6), 2273–2302 (2009)
3. Albers, S., Lenzner, P.: On Approximate Nash Equilibria in Network Design. In: Saberi, A. (ed.) WINE 2010. LNCS, vol. 6484, pp. 14–25. Springer, Heidelberg (2010)
4. Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, É., Wexler, T., Roughgarden, T.: The Price of Stability for Network Design with Fair Cost Allocation. SIAM Journal on Computing 38(4), 1602–1623 (2008)
5. Andelman, N., Feldman, M., Mansour, Y.: Strong price of anarchy. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 189–198 (2007)
6. Balcan, M.F.: Leading dynamics to good behavior. SIGecom Exchanges 10, 19–22 (2011)
7. Bhalgat, A., Chakraborty, T., Khanna, S.: Approximating pure Nash equilibrium in cut, party affiliation, and satisfiability games. In: Proceedings of the 11th ACM Conference on Electronic Commerce (EC), pp. 73–82 (2010)
8. Bilò, V., Fanelli, A., Flammini, M., Melideo, G., Moscardelli, L.: Designing Fast Converging Cost Sharing Methods for Multicast Transmissions. Theory of Computing Systems 47(2), 507–530 (2010)
9. Bilò, V., Fanelli, A., Flammini, M., Moscardelli, L.: When ignorance helps: Graphical multicast cost sharing games. Theoretical Computer Science 411(3), 660–671 (2010)
10. Bilò, V., Flammini, M.: Extending the Notion of Rationality of Selfish Agents: Second Order Nash Equilibria. Theoretical Computer Science 412(22), 2296–2311 (2011)
11. Caragiannis, I., Flammini, M., Kaklamanis, C., Kanellopoulos, P., Moscardelli, L.: Tight Bounds for Selfish and Greedy Load Balancing. Algorithmica 61(3), 606–637 (2011)
12. Charikar, M., Karloff, H.J., Mathieu, C., Naor, J., Saks, M.E.: Online multicast with egalitarian cost sharing. In: Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 70–76 (2008)

13. Chekuri, C., Chuzhoy, J., Lewin-Eytan, L., Naor, J., Orda, A.: Non-Cooperative Multicast and Facility Location Games. IEEE Journal on Selected Areas in Communications 25(6), 1193–1206 (2007)
14. Chien, S., Sinclair, A.: Convergence to approximate nash equilibria in congestion games. In: SODA, pp. 169–178. SIAM (2007)
15. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: Proceedings of ACM Symposium on Theory of Computing (STOC), pp. 67–73 (2005)
16. Christodoulou, G., Mirrokni, V.S., Sidiropoulos, A.: Convergence and Approximation in Potential Games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 349–360. Springer, Heidelberg (2006)
17. Czumaj, A., Krysta, P., Vöcking, B.: Selfish traffic allocation for server farms. In: Proceedings of ACM Symposium on Theory of Computing (STOC), pp. 287–296 (2002)
18. Epstein, A., Feldman, M., Mansour, Y.: Strong equilibrium in cost sharing connection games. In: ACM Conference on Electronic Commerce (EC), pp. 84–92 (2007)
19. Fanelli, A., Flammini, M., Moscardelli, L.: The Speed of Convergence in Congestion Games under Best-Response Dynamics. ACM Transactions on Algorithms 8(3), 25 (2012)
20. Fanelli, A., Moscardelli, L.: On best response dynamics in weighted congestion games with polynomial delays. Distributed Computing 24(5), 245–254 (2011)
21. Gourvès, L., Monnot, J.: On Strong Equilibria in the Max Cut Game. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 608–615. Springer, Heidelberg (2009)
22. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
23. Paes Leme, R., Syrgkanis, V., Tardos, É.: The curse of simultaneity. In: Proceedings of Innovations in Theoretical Computer Science (ITCS), pp. 60–67. ACM Press (2012)
24. Paes Leme, R., Syrgkanis, V., Tardos, É.: Sequential Auctions and Externalities. In: SODA, pp. 869–886. ACM (2012)
25. Mirrokni, V.S., Vetta, A.: Convergence issues in competitive games. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) APPROX and RANDOM 2004. LNCS, vol. 3122, pp. 183–194. Springer, Heidelberg (2004)
26. Nash, J.F.: Equilibrium Points in n-Person Games. Proceedings of the National Academy of Sciences of the United States of America 36, 48–49 (1950)
27. Nash, J.F.: Non-Cooperative Games. Annals of Mathematics 54(2), 286–295 (1951)
28. Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
29. Osborne, M.J., Rubinstein, A.: A course in game theory. MIT Press (1994)
30. Roughgarden, T., Tardos, É.: How bad is selfish routing? Journal of the ACM 49(2), 236–259 (2002)
31. Shapley, L.S.: The value of n-person games. In: Contributions to the Theory of Games, pp. 31–40. Princeton University Press (1953)

Scheduling with an Orthogonal Resource Constraint

Martin Niemeier[1] and Andreas Wiese[2]

[1] TU Berlin, Sekretariat MA 5-1, Straße des 17. Juni 136, 10623 Berlin, Germany
martin.niemeier@tu-berlin.de
[2] Università di Roma "La Sapienza", Via Ariosto 25, 00185 Roma, Italy
wiese@dis.uniroma1.it

Abstract. We address a scheduling problem that arises in highly parallelized environments like modern multi-core CPU/GPU computer architectures. Here simultaneously active jobs share a common limited resource, e.g., memory cache. The scheduler must ensure that the demand for the common resource never exceeds the available capacity. This introduces an *orthogonal constraint* to the classical *minimum makespan scheduling* problem. Such a constraint also arises in many other contexts where a common resource is shared across the machines.

We study the non-preemptive case of this problem and give a $(2+\epsilon)$-approximation algorithm which relies on the interplay of several classical and modern techniques in scheduling like grouping, job-classification, and the use of configuration-LPs. This improves upon previous bound of 3 that can be obtained by list scheduling approaches, and gets close to the $(3/2-\epsilon)$ inapproximability bound. If the number of machines or the number of different resource requirements are bounded by a constant we have a polynomial time approximation scheme.

1 Introduction

Highly parallelized processing in modern multi-core CPU/GPU computer architectures poses the following scheduling challenge: Simultaneously active jobs need to share a common limited resource, e.g. a memory cache. The scheduler must ensure that the demand for the common resource never exceeds the available capacities. This introduces an additional "orthogonal" constraint to scheduling problems.

In this paper we address the imposition of such an orthogonal constraint to the classical and well studied *minimum makespan problem*. We are given a set of jobs where each job j has a processing time $p(j)$ and resource requirement $r(j)$. The goal is to schedule the jobs to m identical machines minimizing the makespan, i.e. the largest finishing time of a job. The scheduler must ensure that the *resource constraint* is not violated: At no time t, the total resource consumption (i.e. the sum of resource requirements of jobs active at time t) exceeds 1. (Note that fixing the resource capacity to 1 is without loss of generality.) This problem is called the *resource constrained scheduling problem* (see e.g. [15]). Note that this

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 242–256, 2013.

model also captures many other settings where the jobs on the machines share a common resource like power supply, workers, etc.

This problem can be seen as a hybrid of two classical problems from two different domains. On the one hand, one obtains minimum makespan scheduling on identical machines if all resource requirements are zero. On the other hand, the well-known Bin-Packing problem is a special case of the resource constrained scheduling problem: Given a Bin-Packing instance, create a job of processing time 1 for each item and set its resource requirement to the item size. The hybrid nature of the problem is also reflected in our algorithms. To tackle it, we combine ideas and techniques from both domains.

1.1 Related Work

Without the resource constraint, one obtains the classic and well studied problem of scheduling on identical machines. Here Graham shows that a natural greedy list scheduling algorithm, where jobs are scheduled in the order of non-increasing processing times, yields an approximation ratio of $\frac{4}{3} - \frac{3}{m}$ [9,10]. After a series of improvements [4,6,18,20] Hochbaum and Shmoys present a a polynomial time approximation scheme (PTAS) [14].

The problem with orthogonal resource constraints, i.e. the resource constrained scheduling problem, was first studied by Garey and Graham [7]. They consider s independent orthogonal resources and show that every list scheduler is a $(s + 2 - \frac{2s+1}{m})$-approximation algorithm. In the setting of unrelated machines and one resource a 3.75-approximation algorithm follows from a more general result for scheduling with resource dependent processing times [11]. For the latter problem on identical machines also a $(3.5 + \epsilon)$-approximation algorithm is known [16]. If preemption is allowed then there is a PTAS if the number of resource requirements is bounded by a constant [15]. If additionally the number of machines is constant, the problem can be solved in polynomial time [2]. There is extensive work in classifying further variants of the problem into polynomial time solvable and NP-hard problems. We refer to [8] and [2] for good overviews. Also, the problem can be seen as a special case of the resource-constrained project scheduling problem (RCPSP), see [1,12] for overviews on this problem.

As already mentioned above, the problem of scheduling jobs to minimize the makespan is closely related to the intensively studied Bin-Packing problem, see [13] for a good survey. For this problem, an asymptotic PTAS is known [5] while it is NP-hard to approximate with a better ratio than $3/2$ [13].

Several related, yet clearly different problems are studied in the literature, including multi-dimensional packing problems [3] and geometrical packing problems like strip-packing [17]. For a survey on results for two-dimensional packing problems see [19].

1.2 Our Contribution

We study the problem to minimize the makespan for non-preemptive scheduling with an orthogonal resource constraint. Our main result is a $(2+\epsilon)$-approximation

algorithm which requires a novel and complex combination of modern and classical techniques from scheduling and Bin-Packing, including

- Configuration-LPs and geometric properties of extreme point solutions
- Enumeration by exploiting structure and search-space reduction
- Linear grouping techniques in the spirit of de la Vega and Lueker [5]
- Classification of jobs (large, small, fat, thin jobs).

In particular, our techniques for constructing the schedule go far beyond the list scheduling type methods used in previous work on the problem and its generalizations [7,11,16]. If either the number of machines or the number of different resource requirements is bounded by a constant, these techniques allow us even to obtain a PTAS.

As mentioned above, a simple list scheduler as in [7] achieves an approximation guarantee of 3. In contrast, our $(2+\epsilon)$-approximation algorithm is rather complex, but gets close to the $(\frac{3}{2}-\epsilon)$-inapproximability bound of the problem. The extra effort to achieve an improvement of $1-\epsilon$ in the approximation ratio might not appear to be justified for practical purposes. However, considering that the $(2+\epsilon)$-guarantee is close to the inapproximability bound, we believe that our contribution to the theoretical understanding of the problem is well worth it.

1.3 Formal Definition and Notation

We define the *resource constrained scheduling problem* formally. An instance consists of an integer m and a set of jobs $\mathcal{J}$ where each job $j \in \mathcal{J}$ is characterized by its processing time $p(j)$ and its resource requirement $r(j)$. For notational convenience, we usually denote an instance just by the job set $\mathcal{J}$, implicitly assuming that m, p and r are given as well. For a natural number $\ell \in \mathbb{N}$, we write $[\ell] := \{0, \ldots, \ell-1\}$. A *slot* is a time-interval $[t_1, t_2)$ with $t_2 \geq t_1$. A *machine slot* $(k, t_1, t_2) \in [m] \times \mathbb{Q}_{\geq 0} \times \mathbb{Q}_{\geq 0}$ is a slot assigned to a machine k. Let $\mathcal{M}$ denote the set of all machine slots. For notational convenience we often identify machine slots s with the interval they represent: We write $|s|$ instead of $|[t_1, t_2)|$, and $t \in s$ instead of $t \in [t_1, t_2)$. The same applies for unions and intersections of machine slots. A *schedule* is a map $\varphi : \mathcal{J} \to \mathcal{M}$ that assigns a machine slot to each job $j \in \mathcal{J}$. We call a schedule *feasible*, if the length of the assigned machine slot to each job is sufficient, assigned machine slots on the same machine are pairwise non-intersecting, and the resource constraint is satisfied. Formally, a schedule is feasible if $|\varphi(j)| \geq p(j)$ for all $j \in \mathcal{J}$, we have that $\varphi(j) \cap \varphi(j') = \emptyset$ for all $j, j' \in \mathcal{J}$ assigned to the same machine, and $\sum_{j \in \mathcal{J}:\ t \in \varphi(j)} r(j) \leq 1$ for all $t \in \mathbb{Q}_{\geq 0}$. If not noted otherwise, when talking about schedules we always mean feasible schedules. The *makespan* $T(\varphi)$ of a schedule φ is the largest endpoint of an assigned machine slot, i.e. $T(\varphi) := \sup \bigcup_{j \in \mathcal{J}} \varphi(j)$. With $OPT(\mathcal{J})$ we denote the optimal makespan of an instance $\mathcal{J}$. We write $c = O(1)$ to denote that c is some constant. If the constant depends on the parameter ϵ, we write $c = O_\epsilon(1)$ instead.

2 A List Scheduler

Before we describe the main result, we discuss the following simple list scheduling algorithm introduced by Garey and Graham [7]: On input $\mathcal{J}$, we iteratively compute a schedule by adding the jobs one by one as follows. While there are unassigned jobs, determine the smallest time t a not yet assigned job could be placed into the schedule without making it infeasible. Assign it by allocating a suitable machine slot $(k, t, t + p(j))$. If there are several candidates, choose an arbitrary one. Garey and Graham proved an approximation guarantee of $3 - \frac{3}{m}$, and give examples that show that the analysis is tight. Hence this list scheduler is insufficient for our purposes as we are aiming for a $(2 + \epsilon)$-approximation algorithm. Nevertheless, it will prove useful as a subroutine of our main algorithm later, however in a slight variation. Instead of starting with an empty schedule, we start with a partial schedule φ' that schedules a subset of jobs $\mathcal{J}' \subseteq \mathcal{J}$. The list scheduler is then used to complete the schedule by adding the remaining jobs $\mathcal{J} \setminus \mathcal{J}'$ one by one as described above. We will now derive a simple upper bound on the makespan of schedules generated by this algorithm. It depends on three parameters that we define now. Given an instance $\mathcal{J}$, we set

$$\bar{P}(\mathcal{J}) := \sum_{j \in \mathcal{J}} p(j)/m \text{ and } \bar{R}(\mathcal{J}) := \sum_{j \in \mathcal{J}} p(j)r(j).$$

Both values are lower bounds on the optimal makespan (which was also observed by Garey and Graham), a fact that will come in handy later.

Lemma 1. *$OPT(\mathcal{J}) \geq \max\{\bar{P}(\mathcal{J}), \bar{R}(\mathcal{J})\}$.*

Proof (sketch). The first bound follows from the fact that no schedule can do better than keeping all machines busy at all times. The second bound follows from the resource constraint limiting the total resource consumption to 1. □

While the parameters $\bar{P}(\mathcal{J})$ and $\bar{R}(\mathcal{J})$ only depend on the instance, the third parameter depends on the given partial schedule φ'. An *activation point* of φ' is a time-index t where some machine becomes busy that was idle before, or the resource consumption increases. Let $A(\varphi')$ denote the number of activation points of a schedule φ'. We now derive the following bound on the makespan of the schedule computed by our list scheduler when used to complete a partial schedule φ'.

Lemma 2. *Let $\mathcal{J}' \subseteq \mathcal{J}$ and let φ' be a schedule for $\mathcal{J}'$. Let p and r be such that $p(j) \leq p$ and $r(j) \leq r$ for all $j \in \mathcal{J} \setminus \mathcal{J}'$. In polynomial time we can compute a schedule φ for $\mathcal{J}$ with $T(\varphi) \leq \max\{T(\varphi'), \bar{P}(\mathcal{J}) + \frac{1}{1-r}\bar{R}(\mathcal{J}) + (A(\varphi') + 1) \cdot p\}$.*

Proof. If the makespan of φ is $T(\varphi')$, we are done. Hence assume that the makespan increased when adding the jobs of $\mathcal{J} \setminus \mathcal{J}'$. Let $j^* \in \mathcal{J} \setminus \mathcal{J}'$ be a job that finishes last, i.e. it determines the makespan.

Let $t \leq T(\varphi)$ be a time index. Observe that we are always in (at least) one of the following three cases: (a) All machines are busy at time t, (b) Job j^* is

active, (c) A machine is idle and job j^* is not active. Clearly the total time case (a) can apply is bounded by $\bar{P}(\mathcal{J})$. Trivially, the total time spent in case (b) is at most $p(j^*) \leq p$. Now assume that we are in case (c),and there is *no* activation point in the interval $[t, t+p)$. Then the resource consumption at time t is at least $1-r$ as otherwise the algorithm could and would have scheduled job j^* or some other job at time t on the idle machine. It follows that the total time spent in case (c) with no upcoming activation point in the interval $[t, t+p)$ is at most $\frac{1}{1-r}\bar{R}(\mathcal{J})$. The remaining time spent in case (c) not yet accounted for is then $A(\varphi') \cdot p$. Summing up the individual bounds for all cases, we get the desired running time bound. □

Observe that if both r and p are "very small" and $A(\varphi')$ is "not too large", the second term of the statement from the lemma gets arbitrarily close to $2 \cdot OPT(\mathcal{J})$. This is why the algorithm will be useful as a subprocedure of our $(2+\epsilon)$-approximation algorithm. We will use the algorithm also to schedule some very small sub-instances separately. To do so, we will rely on the following bound.

Corollary 1. *Given an instance $\mathcal{J}$, and let $p := \max\{p(j) : j \in \mathcal{J}\}$. In polynomial time we can compute a schedule φ for $\mathcal{J}$ with $T(\varphi) \leq \bar{P}(\mathcal{J}) + 2\bar{R}(\mathcal{J}) + 2p$.*

Proof. Let $\mathcal{J}' := \{j \in \mathcal{J} : r(j) > \frac{1}{2}\}$. Let φ' be a schedule for $\mathcal{J}'$ that schedules all jobs of that instance sequentially on the first machine, sorted non-increasingly by resource requirement. Observe that the schedule has makespan at most $2\bar{R}(\mathcal{J})$ as the resource usage is at least $\frac{1}{2}$ at all times. Moreover, the schedule has only one activation point. Hence applying Lemma 2 to this schedule proves the claim. □

3 The $(2+\varepsilon)$-Approximation Algorithm

Fix a constant $\epsilon > 0$. We present a polynomial time algorithm with the following property. For any instance $\mathcal{J}$ with $OPT(\mathcal{J}) \leq 1$, it computes a feasible schedule of makespan at most $2 + O(1) \cdot \epsilon$. Such an algorithm can easily be turned into a $(2+\epsilon)$-approximation algorithm using a binary search framework. We simplify the problem even further by considering only instances that are (γ, M)-restricted.

Definition 1. *Let $1 \geq \gamma > 0$, $M > 1$. An instance $\mathcal{J}$ is* (γ, M)-restricted *if for each job j it holds that either $p(j) \geq \gamma$ or $p(j) < \gamma/M$.*

In (γ, M)-restricted instances, we classify jobs j with $p(j) \geq \gamma$ as *large* and jobs j with $p(j) < \gamma/M$ as *small*. Every large job is at least M times as long as any small job. This fact will become handy later. For a (γ, M)-restricted instance $\mathcal{J}$, let $\mathcal{J}_{\text{large}}$ be the set of large and $\mathcal{J}_{\text{small}}$ be the set of small jobs respectively. The following lemma justifies that we can restrict ourselves to restricted instances.

Lemma 3. *For any constants $\epsilon > 0$, $M > 1$, there is a constant $\gamma^*_{\epsilon,M}$ such that for any instance $\mathcal{J}$, in polynomial time we can find a value $\gamma \geq \gamma^*_{\epsilon,M}$ and a*

partition of the instance $\mathcal{J} = \mathcal{J}_1 \dot\cup \mathcal{J}_2$ so that $\mathcal{J}_1$ is (γ, M)-restricted and for J_2 the list scheduler computes a schedule of makespan at most $O(1) \cdot \epsilon \cdot OPT(\mathcal{J})$.

Proof (sketch). Choose $O_\epsilon(1)$ many disjoint sub-intervals from $(0, O_\epsilon(1)]$ so that each right endpoint is M times larger than its left endpoint. By the pigeonhole-principle, the jobs of one of the intervals have negligible $\bar{P}$ and $\bar{R}$ values. Call them $\mathcal{J}_2$. Using the greedy list scheduler from Section 2, by Lemma 1 and Corollary 1 the list scheduler computes a schedule of makespan at most $O(1) \cdot \epsilon \cdot OPT(\mathcal{J})$ for $\mathcal{J}_2$. □

We set $M := \epsilon^{-3}$. For the remainder of this section, let $\mathcal{J}$ be a (γ, M)-restricted instance with $OPT(\mathcal{J}) \leq 1$. The algorithm to schedule restricted instances is in itself composed of several steps. The rough outline is as follows. We first consider only the large jobs. We compute a schedule of makespan $1 + \epsilon$ for all except a constant number of jobs. Next we compute a set of candidate schedules for the remaining jobs. Each of them has makespan $1+\epsilon$ as well. We can guarantee that for at least one of the schedules, all small jobs can be added without increasing the makespan (we do not know which one though, so we try them all). As the problem of adding the small jobs is NP-hard on its own, we again rely on approximations by allowing the makespan to increase "a bit". Adding small jobs takes place in two steps. First, for each candidate schedule we add the *fat* jobs, i.e. small jobs of "large" resource requirement. This is successful at least for one candidate. We then concatenate the successful schedule with the separate schedule for only large jobs that we computed in the first step. Finally we complete the schedule by adding the so far not yet scheduled *thin* jobs, i.e., small jobs with "small" resource requirement, using a list scheduler. In total this will result in a schedule of makespan $2 + O(1) \cdot \epsilon$. In summary, we have the following steps.

Step 1a: Compute schedule φ_1 for "almost" all large jobs.
Step 1b: Compute set of candidate schedules for remaining large jobs.
Step 2a: For each candidate, try to add small fat jobs.
Concatenate φ_1 with successful schedule from step 2a.
Step 2b: Add small thin jobs to concatenated schedule using the list scheduler.

Step 1a: Scheduling "almost" All Large Jobs. We discretize the scheduling decisions for large jobs by setting $\delta := \epsilon \cdot \gamma/2$ and requiring the assigned machine slots to start and end at integer multiples of δ. I.e., for each $j \in \mathcal{J}_{\text{large}}$, its machine slot should be of the form $(k, \ell_1 \cdot \delta, \ell_2 \cdot \delta)$ with $\ell_1, \ell_2 \in \mathbb{N}_0$. We call schedules whose large jobs have this property *δ-atomic*. The following lemma asserts that it is sufficient to consider only δ-atomic schedules.

Lemma 4. *There exists a δ-atomic schedule for $\mathcal{J}$ whose makespan is at most $(1 + \epsilon) \cdot OPT(\mathcal{J}) \leq 1 + \epsilon$.*

Proof. Take an optimal schedule φ. Define another schedule φ' by setting $\varphi'(j) := (k, (1 + \epsilon) \cdot a, (1 + \epsilon) \cdot b)$ for each $\varphi(j) = (k, a, b)$. Observe that φ' is feasible and

has a makespan of at most $(1+\epsilon)T(\varphi) = (1+\epsilon)OPT(\mathcal{J})$. For every large job $j \in \mathcal{J}_{\text{large}}$, its machine slot $\varphi'(j)$ has length $|\varphi'(j)| = (1+\epsilon)|\varphi(j)| \geq p(j) + 2\delta$. The last inequality is by the fact that j is large and hence $\epsilon \cdot p(j) \geq \epsilon\gamma = 2\delta$. Hence, we can round the starting times of large machine slots up and their ending times down to multiples of δ without decreasing their length below $p(j)$. This way we obtain a feasible δ-atomic schedule. □

We call a set of $|\mathcal{J}_{\text{large}}|$ many machine slots (without job assignment) a *template*. It is called *feasible* if it corresponds to a feasible schedule (i.e. there is a feasible schedule that uses the slots from the template). To compute a schedule for the large jobs, we will first find a feasible template, and later assign jobs to its slots. More precisely, we want to find the template corresponding to the schedule of makespan $1+\epsilon$ due to Lemma 4. To do so, we partition the timeline $[0, 1+\epsilon)$ into *frames* $F_\ell := [\delta \cdot (\ell - 1), \delta \cdot \ell)$ for $\ell \in \mathbb{N}$ and set $\mathcal{F} := \left\{F_\ell \ : \ 1 \leq \ell \leq \left\lceil \frac{1+\epsilon}{\delta} \right\rceil\right\}$. Observe that all large machine slots in our δ-atomic schedule are unions of frames from $\mathcal{F}$. Note that since $\gamma \geq \gamma^*_{\epsilon,M}$ (and $\gamma^*_{\epsilon,M}$ is a constant), we have that $|\mathcal{F}| = \left\lceil \frac{1+\epsilon}{\delta} \right\rceil = O_\epsilon(1)$ is constant. This allows us to find a feasible template by enumeration:

Lemma 5. *In polynomial time we can compute a polynomial number of candidate templates. At least one of them is feasible.*

Proof. In a feasible δ-atomic schedule there are at most $|\mathcal{F}|$ many large jobs on a single machine. For each job, start and end times are chosen from $|\mathcal{F}|+1$ many possibilities. Hence, there are at most $Q := \binom{|\mathcal{F}|+1}{2}^{|\mathcal{F}|}$ feasible combinations of machine slots for one machine. Up to permutation of machines, we can describe every template by specifying how many machines use each of the Q possibilities. This results in at most $m^Q = m^{O_\epsilon(1)}$ many candidate templates. □

To compute the schedule, we repeat the following procedure for each of the candidate templates. It will be successful for any feasible template. Let $\mathcal{T}$ be a template. We use a linear program to compute an assignment of large jobs to machine slots from $\mathcal{T}$. Let $\mathcal{I}$ denote the set of all slots from $\mathcal{T}$ (i.e. the time intervals without the machine assignment). For each slot $s \in \mathcal{I}$, let μ_I denote the number of machines slots from the template $\mathcal{T}$ that use it. We model the problem of assigning jobs to intervals with the following linear program.

$$\sum_{I \in \mathcal{I},\ |I| \geq p(j)} x_{j,I} = 1 \quad \forall j \in \mathcal{J}_{\text{large}} \tag{1}$$

$$\sum_{j \in \mathcal{J}_{\text{large}}} x_{j,I} \leq \mu_I \quad \forall I \in \mathcal{I}$$

$$\sum_{I \in \mathcal{I}, F \subseteq I} \sum_{j \in \mathcal{J}_{\text{large}}} r(j) x_{j,I} \leq 1 \quad \forall F \in \mathcal{F}$$

$$x_{j,I} \geq 0 \quad \forall j \in \mathcal{J}_{\text{large}}\ \forall I \in \mathcal{I}$$

The variables $x_{j,I}$ model the assignment of jobs to intervals, where $x_{j,I} = 1$ means that job i is assigned a machine slot with time interval I.

Lemma 6. *Given a feasible template, in polynomial time we can compute a δ-atomic schedule of makespan $(1+\epsilon)$ that schedules all except for $|\mathcal{I}|+|\mathcal{F}|$ many large jobs. The schedule has at most $O(1)\frac{1}{\epsilon\gamma}$ activation points.*

Proof. An extreme point solution of the above linear program has at most $|\mathcal{J}_{\text{large}}|+|\mathcal{I}|+|\mathcal{F}|$ many non-zero entries. Hence, due to Constraints (1) at most $|\mathcal{I}|+|\mathcal{F}|$ jobs are fractionally assigned. We output the schedule obtained for the integrally assigned jobs. Since it is δ-atomic, there are at most $|\mathcal{F}|$ activation points. □

Note that $|\mathcal{I}| = O_\epsilon(1)$ just like $|\mathcal{F}|$. Hence all except for a constant number of jobs are scheduled.

Step 1b: Scheduling the Remaining Large Jobs. If there are only constantly many large jobs to schedule, we can enumerate all possible δ-atomic schedules in polynomial time. Importantly, one of these schedules is *extendable.*

Definition 2. *Let $\mathcal{J}$ be a set of jobs and let φ be a schedule for a subset $\mathcal{J}' \subseteq \mathcal{J}$. The schedule φ is* extendable for $\mathcal{J}$ *if the jobs $\mathcal{J} \setminus \mathcal{J}'$ can be added to φ without increasing the makespan.*

Lemma 7. *Let $\mathcal{J}'_{\text{large}} \subseteq \mathcal{J}$ be a set of large jobs with $|\mathcal{J}'_{\text{large}}| = O_\epsilon(1)$. In polynomial time we can compute a set of δ-atomic schedules of makespan $1+\epsilon$. Each of them is feasible for the sub-instance $\mathcal{J}'_{\text{large}}$. At least one of them is extendable for $\mathcal{J}$. The number of activation points of each schedule is at most $O(1) \cdot \frac{1}{\epsilon\gamma}$.*

Proof (sketch). We can restrict to only using the first $|\mathcal{J}'_{\text{large}}| = O_\epsilon(1)$ machines. Hence there are only constantly many machine slots to consider. □

Step 2a: Adding Small Fat Jobs. We now describe how to add small jobs to a δ-atomic schedule for large jobs. As mentioned previously, there are two kinds of small jobs that need to be treated differently. Define $\beta := \epsilon$. We say that a small job is *fat* if $r(j) \geq \beta$. Otherwise it is *thin.*

We are now in the following situation: We consider a sub-instance $\mathcal{J}' \subseteq \mathcal{J}$ consisting of some large jobs $\mathcal{J}'_{\text{large}}$ and all tiny fat jobs $\mathcal{J}'_{\text{small}}$. Note that $\mathcal{J}'_{\text{small}}$ does *not* contain the small thin jobs. We have a schedule φ_2 for $\mathcal{J}'_{\text{large}}$ of makespan $1+\epsilon$, and we want to add the small fat jobs $\mathcal{J}'_{\text{small}}$. For simplicity we assume that φ_2 is extendable and show that in this case, the algorithm is successful. The roadmap is as follows: We first round the resource requirements so that only a constant number of different values remain. We then compute an "optimal invalid" schedule for a transformed instance. It is invalid because it allows preemption, migration and parallelization. However, in the end this invalid schedule allows us to compute a "good" feasible schedule for our instance.

The resource rounding is done with a linear grouping technique similar in spirit as the technique employed in [5] for Bin-Packing. We first sort the jobs from $\mathcal{J}'_{\text{small}}$ non-decreasingly by resource requirement. Let $\mathcal{J}'_{\text{small}} = \{j_1, \ldots, j_n\}$ be in this order. For $K := \lceil 1/\epsilon^2 \rceil$, we divide the jobs into K groups as follows:

Figuratively, we take a schedule where the jobs from $\mathcal{J}'_{\text{small}}$ are scheduled sequentially in the order from above, slice it into K intervals of equal length and define that $\mathcal{J}_i$ is the group of jobs that are completely contained in interval i. Group $\mathcal{J}_0$ is the set of jobs that are cut when defining the intervals. Formally, define $\mathcal{J}_i := \{j_k \ : \ (i-1) \cdot p(\mathcal{J}'_{\text{small}})/K \leq \sum_{\ell=1}^{k-1} p(j_\ell) \leq i \cdot p(\mathcal{J}'_{\text{small}})/K - p(j_k)\}$ for $i = 1, \ldots, K$, and $\mathcal{J}_0 := \mathcal{J}'_{\text{small}} \setminus \bigcup_{i=1}^{K} \mathcal{J}_i$. By construction, we obtain the following properties of the set system.

Lemma 8. *We have $|\mathcal{J}_0| \leq K$, $p(\mathcal{J}_i) \leq p(\mathcal{J}'_{\text{small}})/K$ for all $i = 1, \ldots, K$, and for any two jobs $j \in \mathcal{J}_i$ and $j' \in \mathcal{J}_{i'}$ with $1 \leq i < i'$ it holds that $r(j) \leq r(j')$.*

Based on this grouping, we define a set of jobs $\tilde{\mathcal{J}}'_{\text{small}} := \{g_1, ..., g_{K-1}\}$ by setting $p(g_i) := p(\mathcal{J}'_{\text{small}})/K$ and $r(g_i) := \max\{r(j) \,|\, j \in \mathcal{J}_i\}$ for each $i \in \{1, ..., K\}$. For the resulting instance $\tilde{\mathcal{J}}' := \mathcal{J}'_{\text{large}} \cup \tilde{\mathcal{J}}'_{\text{small}}$, we later compute an "invalid" schedule. The jobs g_i are going to act as placeholders to fill in the jobs from $\mathcal{J}_i$ in the final solution later. The groups $\mathcal{J}_0$ and $\mathcal{J}_K$ are treated differently. They can be scheduled separately:

Lemma 9. *The jobs in $\mathcal{J}_0 \cup \mathcal{J}_K$ can be scheduled with makespan $O(1) \cdot \epsilon$.*

Proof. Because the jobs in $\mathcal{J}_0$ are small we have $p(\mathcal{J}_0) \leq K \cdot \frac{\gamma}{M} \leq O(1) \cdot \epsilon$. For $\mathcal{J}_K$, recall the lower bound $\bar{R}$ from Section 2. We get

$$1 \geq OPT(\mathcal{J}') \geq \bar{R}(\mathcal{J}') \geq \bar{R}(\mathcal{J}'_{\text{small}}) \geq \beta \cdot p(\mathcal{J}'_{\text{small}}) \geq \beta \cdot K \cdot p(\mathcal{J}_K) \geq p(\mathcal{J}_K)/\epsilon.$$

We conclude that if we schedule the jobs $\mathcal{J}_0 \cup \mathcal{J}_K$ sequentially on one machine, the makespan is $O(1) \cdot \epsilon$. □

We now discuss how to compute the "invalid" helper schedule. We call it a *relaxed* schedule. In relaxed schedules, we allow jobs to be preempted, migrated, and executed in parallel. However the same rules for feasibility apply as for "real" schedules. For simplicity of presentation, we refrain from a formal definition of relaxed schedules. We first prove the existence of a relaxed schedule for $\tilde{\mathcal{J}}'$.

Lemma 10. *If φ_2 is extendable for $\mathcal{J}'$, then it is extendable (as a relaxed schedule) for $\tilde{\mathcal{J}}'$.*

Proof (sketch). Let $\bar{\varphi}_2$ be an extension of φ_2 for $\mathcal{J}'$. Recall the illustration of slicing a sequential schedule for $\mathcal{J}'_{\text{small}}$ into K equally sized intervals. To construct a relaxed schedule for $\tilde{\mathcal{J}}'$, for each i we use the jobs from interval $i+1$ as a template to fill in the job g_i. Note that this might include some fractions of jobs from that interval which are not included in $\mathcal{J}_i$ but in $\mathcal{J}_0$. □

We now show how to compute such a relaxed schedule for $\tilde{\mathcal{J}}'$. We again resort to linear programming. Note that in order to extend φ_2 we can use only resources "left over" by the large jobs $\mathcal{J}'_{\text{large}}$. Based on φ_2, for each frame $F \in \mathcal{F}$ let $r(F)$ denote the amount of resources available for non-large jobs, and let $m(F)$ denote the number of machines available. Note that these quantities are the same

throughout a frame as the schedule φ_2 is δ-atomic. We will use these remaining resources to schedule small jobs of the instance.

Every possibility of small jobs being simultaneously active during frame $F \in \mathcal{F}$ can be described by a characteristic vector $\chi \in [m]^{K-1}$ such that $\sum_{i=1}^{K-1} \chi_i r(g_i) \leq r(F)$ and $\sum_i \chi_i \leq m(F)$. Let $\mathcal{C}(F)$ be the set of all such vectors. For each job g_i, the entry χ_i specifies the number of machines used for executing job g_i in parallel. We call a vector $\chi \in \mathcal{C}(F)$ a *job configuration*. This allows us to formulate a "configuration-LP" that packs job configurations to frames and ensures that all small jobs are covered. Denote by CONF-LP the following linear program.

$$\sum_{F \in \mathcal{F}} \sum_{\chi \in \mathcal{C}(F)} \chi_i x_{F\chi} \geq p(g_i) \quad \forall i = 1, \ldots, K-1 \tag{2}$$

$$\sum_{\chi \in \mathcal{C}(F)} x_{F\chi} \leq \delta \quad \forall F \in \mathcal{F} \tag{3}$$

$$x_{F\chi} \geq 0 \quad \forall F \in \mathcal{F}\ \forall \chi \in \mathcal{C}(F)$$

The variable $x_{F\chi}$ models for how much time configuration χ should be used within frame F. Due to Lemma 10 we know that CONF-LP has a solution and an extreme point solution fulfills the properties of the following lemma.

Lemma 11. *There is a polynomial time algorithm which computes a relaxed schedule for $\tilde{\mathcal{J}}'$ which extends φ_2. In particular, in polynomial time we can compute a solution to CONF-LP with at most $K + |\mathcal{F}|$ non-zero variables.*

Based on the solution to CONF-LP we construct a non-relaxed schedule φ_2' for $\mathcal{J}'$. We partition each frame $F \in \mathcal{F}$ into subframes, each subframe corresponds to a positive variable $x_{F\chi}$ and has length $x_{F\chi}$. The packing constraints (3) ensure that we can do that. Now for each subframe and each $i \in [K]$, create χ_i many machine slots (assign them to free machines greedily) and reserve them for jobs of group $\mathcal{J}_i$. The machine slots created in this way act as placeholders and, to avoid confusion, we will refer to them as *placeholder slots*. Our construction ensures that if we pack jobs of group $\mathcal{J}_i$ arbitrarily to its reserved placeholder slots, we will not violate the resource requirement (as by Lemma 8, the resource requirement of all jobs from $\mathcal{J}_i$ is at most $r(g_i)$). As the covering constraints (2) are satisfied, the total amount of execution time reserved for each group $\mathcal{J}_i$ is at least $p(g_i)$ which by definition and Lemma 8 is at least $\sum_{j \in \mathcal{J}_i} p(j)$.

Now for each group $\mathcal{J}_i$, $i = 1 \ldots K-1$ and each job $j \in \mathcal{J}_i$, select an arbitrary placeholder slot reserved for group $\mathcal{J}_i$ which has a positive amount of space left and assign j to it. By the observations from above, it is clear that this algorithm manages to assign all jobs. However, it might produce an infeasible solution as some placeholder slots might be over-packed. We can repair this as follows: For each subframe (i.e. for each positive variable in the LP-solution), and for each placeholder slot belonging to this subframe, pick the job added last and remove it. Now the placeholder slots are not over-packed anymore, i.e. the resulting schedule is feasible. For the removed jobs, observe that those that are taken from the same subframe can be scheduled in parallel. Hence, because they are small, we can schedule them separately in a time-frame of γ/M timeunits. As there are at most $|\mathcal{F}| + K$ many nonzeros the LP solution due to Lemma 11, we

conclude that the increase of the makespan is at most $\gamma/M \cdot (|\mathcal{F}| + K) = O(1) \cdot \epsilon$. Hence, in summary we get the following result:

Lemma 12. *Given φ_2, in polynomial time we can compute a schedule φ_2' for $\mathcal{J}'$ with $T(\varphi_2') \leq 1 + O(1) \cdot \epsilon$. Moreover we have $A(\varphi_2') \leq O(1) \cdot \frac{1}{\epsilon^2 \gamma}$.*

Proof. The makespan increase due to the above procedure, as well as the length of the schedules for $\mathcal{J}_0$ and $\mathcal{J}_K$, is bounded by $O(1) \cdot \epsilon$. To see the bound on the activation points, observe that new activation points can only be introduced for each subframe from the construction above. □

Step 2b: Adding Small Thin Jobs. Let φ_1 and φ_2' be the schedules obtained due to Lemma 6 and Lemma 12, respectively. Observe that if we concatenate φ_1 and φ_2', we obtain a schedule of makespan $2 + O(1) \cdot \epsilon$ that schedules all jobs from $\mathcal{J}$ except for the small thin ones. The number of activation points of this schedule is $O(1) \cdot 1/(\epsilon^2 \gamma)$.

We use the list scheduler from Section 2 to complete the schedule. Applying Lemma 2 in this situation, we can set $p := \gamma/M$ and $r := \beta$. Hence we obtain a full schedule of makespan

$$\max \left\{ 2 + O(1) \cdot \epsilon, \bar{P}(\mathcal{J}) + \frac{1}{1-\beta} \bar{R}(\mathcal{J}) + O(1) \cdot \frac{1}{\epsilon^2 \gamma} \gamma/M \right\} = 2 + O(1) \cdot \epsilon.$$

With the discussion from the beginning of this section we conclude:

Theorem 1. *There is a $(2+\epsilon)$-approximation algorithm for the resource constraint scheduling problem.*

4 Polynomial Time Approximation Schemes for Special Cases

In this section we will explain the ideas on how to turn the $2 + \epsilon$ approximation algorithm for non-preemptive resource constrained scheduling into a PTAS for two special cases: The number of machines is upper bounded by a constant C, or the number of *different* resource requirements is bounded by C.

Essentially there are two steps of the algorithm that need to be improved. The first one is the way we deal with large jobs: Instead of creating two schedules of makespan $1 + \epsilon$ and concatenating them, we need to treat all large jobs at the same time. The second issue is the use of the greedy list scheduler to schedule the thin/small jobs: No matter how we tweak the parameters, this algorithms performance guarantee will not get better than $2+\epsilon$. Both issues will be addressed for both special cases, but in different ways.

As before, we assume the instances $\mathcal{J}$ we consider have $OPT(\mathcal{J}) \leq 1$.

4.1 Constant Number of Machines

We now assume that all instances for the resource constrained scheduling problem have the property that there is a universal constant $C > 0$ so that the

number of machines m is upper bounded by C. We also assume that $\frac{1}{\epsilon} \geq C$. Adapting the way on how we treat the large jobs is very easy using the following insight. If there is a schedule of makespan at most 1, how many large jobs can it have? As every large job has processing time at least γ, an optimal schedule can process at most $1/\gamma$ many jobs per machine. As there are at most C machines, there are at most C/γ, i.e. *constantly many*, large jobs. Hence to treat the large jobs, we simply skip Step 1a. Lemma 7 guarantees that in polynomial time we can compute a candidate set of feasible δ-atomic schedules for all large jobs $\mathcal{J}_{\text{large}}$ of the instance, where at least one of them is extendable to a full schedule for $\mathcal{J}$.

To deal with the small jobs, we need to recall why the distinguishment of *fat* and *thin* jobs was made in the first place. The *only* place in the proof where we needed that the jobs in step 2a are fat was the following. Recall that in the process of dealing with fat jobs, we removed two subsets of jobs and scheduled them separately (namely the sets $\mathcal{J}_0$ and $\mathcal{J}_K$). We argued that they can be scheduled separately using only a negligible, i.e. $O(1) \cdot \epsilon$ amount of time. Particularly, the argument for $\mathcal{J}_K$ was that as the jobs are fat, only $1/\beta$, i.e. constantly many, of them can run in parallel. Now, only constantly many of them can run in parallel anyway because we have only at most C many machines. Hence we do not need to distinguish between fat and thin jobs anymore and treat all small jobs as described in Step 2a. We need however a new proof for Lemma 9.

Lemma 13. *Consider the linear grouping process as described in Section 3, where the linear grouping is done with all small jobs. The jobs in $\mathcal{J}_0 \cup \mathcal{J}_K$ can be scheduled with makespan $O(1) \cdot \epsilon$.*

Proof. The proof for the jobs from $\mathcal{J}_0$ is as in Lemma 9: There are at most K many jobs, so their total processing time is a most $p(\mathcal{J}_0) \leq K \cdot \frac{\gamma}{M} \leq O(1) \cdot \epsilon$. For $\mathcal{J}_K$, recall the load bound $\bar{P}$ from Section 2. Using that at most C jobs run in parallel, we get that

$$1 \geq OPT(\mathcal{J}) \geq \bar{P}(\mathcal{J}) \geq \bar{P}(\mathcal{J}_{\text{small}}) \geq p(\mathcal{J}_{\text{small}})/C \geq K/C \cdot p(\mathcal{J}_K) \geq p(\mathcal{J}_K)/\varepsilon.$$

We conclude that if we schedule the jobs $\mathcal{J}_0 \cup \mathcal{J}_K$ sequentially on one machine, the makespan is $O(1) \cdot \epsilon$. □

We conclude that for the special case of constant number of machines, we have a PTAS, simply by using only a subset of components from the main algorithm.

Theorem 2. *For any constant C and $\epsilon > 0$, there is a $(1+\epsilon)$-approximation algorithm for all instances $\mathcal{J}$ with $m \leq C$.*

Proof. We follow the steps of the $(2+\epsilon)$-approximation algorithm presented in Section 3. In particular, using the same constants as before, we again restrict to the case of $(\gamma,\ M)$-restricted instances with $OPT(\mathcal{J}) \leq 1$. As observed above, the total number of large jobs in $\mathcal{J}$ is constant. Hence with Lemma 7 we can compute a candidate set of feasible schedules for $\mathcal{J}_{\text{large}}$. One of them is extendable to a full schedule for $\mathcal{J}$ of makespan $1+\epsilon$.

We then apply the algorithm from Step 2a for all small jobs. Lemma 13 and Lemma 12 guarantee that we will find a schedule for $\mathcal{J}$ of makespan $1+O(1)\cdot\epsilon$. □

4.2 Constant Number of Different Resource Requirements

We now assume that there is a universal constant $C > 0$ so that for each instance $\mathcal{J}$, there are at most C different resource requirements: For each instance $\mathcal{J}$ there are numbers $R_1, \ldots, R_C$ so that $r(j) = R_\ell$ for some $\ell = 1, \ldots, C$. To deal with large jobs, we proceed as follows. Recall the *templates* from Step 1a. We noted that given a feasible template, it is NP-hard to assign the jobs to the machine slots in the template so that we get a feasible schedule. In this special case we can circumvent this as follows. We define a *colorful* machine slot as a pair $(s, \ell) \in \mathcal{M} \times [C]$. I.e. a colorful machine slot is a "regular" machine slot s that is reserved for a job with resource requirement R_ℓ. Analogous, a *colorful template* is a set of $|\mathcal{J}|$ colorful machine slots without a job assignment. It is feasible if there is a feasible schedule that uses the colorful machine slots from the templates for jobs of corresponding resource requirement. Unlike the regular templates, it is now easy to determine whether a template is feasible and to compute a feasible schedule from a feasible template: If a template is feasible, the following greedy algorithm successfully computes a feasible schedule: While there is an unassigned job, choose the one of largest processing time and assign it to any machine slot of its resource requirement whose length is sufficient. We skip the straightforward proof.

Using the fact that there are only a constant number of "colors", essentially by the lines of the proof of Lemma 5 one can prove the following statement.

Lemma 14. *In polynomial time, we can compute a set of colorful templates. At least one of them is feasible and extendable.*

This shows that in polynomial time, we can compute a schedule φ_1 of makespan $1+\epsilon$ for *all* large jobs $\mathcal{J}_{\text{large}}$. Moreover, φ_1 is extendable, i.e. we can add all small jobs without increasing the makespan. It remains to deal with the small jobs. As in the case of constant number of machines, we will not distinguish between thin and fat jobs. Instead we will modify the grouping procedure. Recall that in Step 2a, we grouped the small fat jobs into $K+1$ groups $\mathcal{J}_0, \ldots, \mathcal{J}_K$ and rounded the resource requirement. Now we instead define C many groups $\mathcal{J}_1, \ldots, \mathcal{J}_C$ by resource requirement, i.e.

$$\mathcal{J}_\ell := \{j \in \mathcal{J}_{\text{small}} \ : \ r(j) = R_\ell\}.$$

Analogous to Step 2a, we now construct a helper instance $\tilde{\mathcal{J}}_{\text{small}} := \{g_1, \ldots, g_C\}$. We set $p(g_\ell) := p(\mathcal{J}_\ell)$ and $r(g_\ell) := R_C$. We observe that as φ_1 is extendable, φ_1 is also extendable as a relaxed schedule for $\tilde{\mathcal{J}} := \mathcal{J}_{\text{large}} \cup \tilde{\mathcal{J}}_{\text{small}}$. The proof is a simplified version of the proof for Lemma 10 and we omit the details. Now we can argue by the lines of the description of Step 2a that CONF-LP finds a good extremal point solution which we can turn into a full schedule for $\mathcal{J}$ of makespan $1 + O(1) \cdot \epsilon$. We conclude:

Theorem 3. *For any constant C and $\epsilon > 0$, there is a $(1+\epsilon)$-approximation algorithm for all instances $\mathcal{J}$ where the number of different resource requirements is bounded by C.*

Acknowledgements. We would like to thank Marco Di Summa, Friedrich Eisenbrand, Thomas Rothvoß, and José Verschae for helpful discussions.

References

1. Artigues, C., Demassey, S., Néron, E.: Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. In: ISTE (2010)
2. Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G.: Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics 5(1), 11–24 (1983)
3. Chekuri, C., Khanna, S.: On multi-dimensional packing problems. In: Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), pp. 185–194. SIAM (1999)
4. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: An application of bin-packing to multiprocessor scheduling. SIAM Journal on Computing 7, 1–17 (1978)
5. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1+\varepsilon$ in linear time. Combinatorica 1, 349–355 (1981)
6. Friesen, D.K.: Tighter bounds for the multifit processor scheduling algorithm. SIAM Journal on Computing 13, 170–181 (1984)
7. Garey, M.R., Grahams, R.L.: Bounds for multiprocessor scheduling with resource constraints. SIAM Journal on Computing 4, 187–200 (1975)
8. Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. SIAM Journal on Computing (1975)
9. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell System Technical Journal 45, 1563–1581 (1966)
10. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17, 263–269 (1969)
11. Grigoriev, A., Sviridenko, M., Uetz, M.: Machine scheduling with resource dependent processing times. Mathematical Programming 110, 209–228 (2007)
12. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of OR 207, 1–14 (2010)
13. Hochbaum, D.S. (ed.): Approximation Algorithms for NP-Hard Problems. Thomson (1996)
14. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: Theoretical and practical results. Journal of the ACM 34, 144–162 (1987)
15. Jansen, K., Porkolab, L.: On preemptive resource constrained scheduling: Polynomial-time approximation schemes. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 329–349. Springer, Heidelberg (2002)
16. Kellerer, H.: An approximation algorithm for identical parallel machine scheduling with resource dependent processing times. OR Letters 36, 157–159 (2008)

17. Kenyon, C., Rémila, E.: A near-optimal solution to a two-dimensional cutting stock problem. Mathematics of Operations Research 25, 645–656 (2000)
18. Langston, M.A.: Processor scheduling with improved heuristic algorithms. PhD thesis, Texas A&M University (1981)
19. Lodi, A., Martello, S., Monaci, M.: Two-dimensional packing problems: A survey. European Journal of Operational Research 141, 241–252 (2002)
20. Sahni, S.: Algorithms for scheduling independent tasks. Journal of the ACM 23, 116–127 (1976)

Improved Approximation Guarantees for Lower-Bounded Facility Location*

Sara Ahmadian** and Chaitanya Swamy**

Combinatorics and Optimization, Univ. Waterloo, Waterloo, ON N2L 3G1
{sahmadian,cswamy}@math.uwaterloo.ca

Abstract. We consider the *lower-bounded facility location* (LBFL) problem, which is a generalization of *uncapacitated facility location* (UFL), where each open facility is required to serve a certain *minimum* amount of demand. The current best approximation ratio for LBFL is 448 [17]. We substantially advance the state-of-the-art for LBFL by improving its approximation ratio from 448 [17] to 82.6.

Our improvement comes from a variety of ideas in algorithm design and analysis, which also yield new insights into LBFL. Our chief algorithmic novelty is to present an improved method for solving a more-structured LBFL instance obtained from $\mathcal{I}$ via a bicriteria approximation algorithm for LBFL, wherein all clients are aggregated at a subset $\mathcal{F}'$ of facilities, each having at least αM co-located clients (for some $\alpha \in [0, 1]$). The algorithm in [17] proceeds by reducing $\mathcal{I}_2(\alpha)$ to CFL. One of our key insights is that one can reduce the resulting LBFL instance, denoted $\mathcal{I}_2(\alpha)$, to a problem we introduce, called *capacity-discounted UFL* (CDUFL), which is a structured special case of capacitated facility location (CFL). We give a simple local-search algorithm for CDUFL based on add, delete, and swap moves that achieves a significantly-better approximation ratio than the current-best approximation ratio for CFL, which is one of the reasons behind our algorithm's improved approximation ratio.

1 Introduction

Facility location problems have been widely studied in the Operations Research community (see, e.g., [13]). In its simplest version, *uncapacitated facility location* (UFL), we are given a set of facilities with opening costs, and a set of clients, and we want to open some facilities and assign each client to an open facility so as to minimize the sum of the facility-opening and client-assignment costs. This problem has a wide range of applications. For example, a company might want to open its warehouses at some locations so that its total cost of opening warehouses and servicing customers is minimized.

We consider the *lower-bounded facility location* (LBFL) problem, which is a generalization of UFL where each open facility is required to serve a certain

* A full version [1] is available on the CS arXiv.

** Research supported in part by NSERC grant 327620-09 and the second author's Ontario Early Researcher Award.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 257–271, 2013.

minimum amount of demand. More formally, an LBFL instance $\mathcal{I}$ is specified by a set $\mathcal{F}$ of facilities, a set $\mathcal{D}$ of clients, and an integer M. Opening facility i incurs a *facility-opening cost* f_i, and assigning a client j to a facility i incurs a *connection cost* c_{ij}. A feasible solution specifies a subset $F \subseteq \mathcal{F}$ of facilities, and assigns each client j to an open facility $i(j) \in F$ so that *each open facility serves at least M clients*. The cost of such a solution is the sum of the facility-opening and connection costs, that is, $\sum_{i \in F} f_i + \sum_j c_{i(j)j}$, and the goal is to find a feasible solution of minimum cost. As is standard in the study of facility location problems, we assume throughout that c_{ij}s form a metric. We use the terms connection cost and assignment cost interchangeably in the sequel.

LBFL can be motivated from various perspectives. This problem was introduced independently by Karger and Minkoff [8], and Guha et al. [5] (who called the problem *load-balanced facility location*, both of whom arrived at LBFL as a means of solving their respective buy-at-bulk style network design problems. LBFL arises as a natural subroutine in solving buy-at-bulk problems because obtaining a near-optimal solution often entails aggregating a certain minimum demand at certain hub locations, and then connecting the hubs via links of lower per-unit-demand cost (and higher fixed cost). LBFL also finds direct applications in supply-chain logistics problems, where the lower-bound constraint can be used to model the fact that it is not profitable or feasible to use services unless they satisfy a certain minimum demand. For example (as noted in [17]), Lim et al. [11], use LBFL to abstract a transportation problem faced by a company that has to determine the allocation of cargo from customers to carriers, who then ship their cargo overseas. Here the lower bound arises because each carrier, if used, is required (by regulation) to deliver a minimum amount of cargo.

Clearly, LBFL with $M = 1$ is simply UFL, and hence, is *NP*-hard; consequently, we are interested in designing approximation algorithms for LBFL. The first constant-factor approximation algorithm for LBFL was devised by Svitkina [17], whose approximation ratio is 448. Prior to this, the only known approximation guarantees were *bicriteria guarantees*. [8] and [5] independently devised (ρ, α)-approximation algorithms via a reduction to UFL: these algorithms return a solution of cost at most ρ times the optimum where each open facility serves at least αM clients ($\alpha < 1$, ρ is a function of α).

Our Results and Techniques. We devise an approximation algorithm for LBFL that achieves a substantially-improved approximation guarantee of 82.6 (Theorem 1), thus significantly advancing the state-of-the-art for LBFL. Our improvement comes from a combination of ideas in algorithm design and analysis, and yields new insights about the approximability of LBFL. In order to describe the ideas underlying our improvement, we first briefly sketch Svitkina's algorithm.

Svitkina's algorithm begins by using the reduction in [8,5] to obtain a bicriteria solution for $\mathcal{I}$, which is then used to convert $\mathcal{I}$ into an LBFL instance $\mathcal{I}_2$ with facility-set $\mathcal{F}' \subseteq \mathcal{F}$ having the following structure: (i) all clients are aggregated at $\mathcal{F}'$ with each facility $i \in \mathcal{F}'$ having $n_i \geq \alpha M$ co-located clients; (ii) all facilities in $\mathcal{F}'$ have zero opening costs; and (iii) near-optimal solutions to $\mathcal{I}_2$ translate to near-optimal solutions to $\mathcal{I}$ (and vice versa). The goal now is to identify a

subset of $\mathcal{F}'$ to close, such that transferring the clients aggregated at these closed facilities to the remaining (open) facilities in $\mathcal{F}'$ ensures that each remaining facility serves at least M demand (and the cost incurred is "small"). [17] shows that one can achieve this by solving a suitable CFL instance. Essentially the idea is that a facility i that remains open corresponds to a *demand point* in the CFL instance that requires $M - n_i$ units of demand, and a facility i that is closed maps to a *supply point* in the CFL instance having n_i units that can be supplied to demand points (i.e., open facilities). Of course, one does not know beforehand which facilities will be closed and which will remain open; so to encode this correspondence in the CFL instance, we create at every location $i \in \mathcal{F}'$, a supply point with (suitable opening cost and) capacity M, and a demand point with demand $M - n_i$ if $n_i \leq M$ (so the supply point at i has n_i residual capacity after satisfying this demand). (Assume $n_i \leq M$ for simplicity; facilities with $n_i > M$ are treated differently.) [17] argues that a CFL-solution can be mapped to an $\mathcal{I}_2$-solution without increasing the cost incurred by much; since CFL admits an $O(1)$-approximation algorithm, one obtains an $O(1)$-approximate solution to $\mathcal{I}_2$, and hence to the original LBFL instance $\mathcal{I}$.

Our algorithm also proceeds by (a) obtaining an LBFL instance $\mathcal{I}_2$ satisfying properties (i)–(iii) mentioned above, (b) solving $\mathcal{I}_2$, and (c) mapping the $\mathcal{I}_2$-solution to a solution to $\mathcal{I}$, but our implementation of steps (a) and (b) differs from that in Svitkina's algorithm. These modified implementations, which are independent of each other and yield significant improvements in the overall approximation ratio even when considered in isolation, result in our much-improved approximation ratio. We detail how we perform step (a) later, and focus first on describing how we solve $\mathcal{I}_2$, which is our chief algorithmic contribution.

Our key insight is that one can solve the LBFL instance $\mathcal{I}_2$ by reducing it to a new problem we introduce that we call *capacity-discounted* UFL (CDUFL), which closely resembles UFL and admits an algorithm (that we devise) with a much better approximation ratio than CFL. A CDUFL-instance has the property that every facility is either uncapacitated (i.e., has infinite capacity), or has finite capacity and *zero* facility cost. The CDUFL instance we construct consists of the same supply and demand points as in the reduction of $\mathcal{I}_2$ to CFL in [17], except that all supply points with non-zero opening cost are now uncapacitated. (Interestingly, if all facilities in $\mathcal{I}_2$ have $n_i \leq M$, the CDUFL instance is in fact a UFL-instance!)

We prove two crucial algorithmic results. The "standard" integrality-gap example for the natural LP-relaxation of CFL can be cast as a CDUFL instance, thus showing that the natural LP-relaxation for CDUFL has a large integrality gap, and we are not aware of any LP-relaxation with constant integrality gap. Circumventing this difficulty, we devise a local-search algorithm for CDUFL based on add, swap, and delete moves that achieves the *same performance guarantees* as the corresponding local-search algorithm for UFL [2] (Section 4.2). The local-search algorithm yields significant dividends in the overall approximation ratio because not only is its approximation ratio for CDUFL better than the state-of-the-art for CFL, but also because it yields separate (asymmetric) guarantees on the facility-opening and assignment costs, which allows one to perform a tighter analysis. Second, we show that any near-optimal CDUFL-solution can be mapped

to a near-optimal solution to $\mathcal{I}_2$ (Section 4.1). As in [17], in the CDUFL-solution, an open supply point i (which corresponds to closing facility i) may send less than n_i supply to other demand points, so that closing down i entails transferring its residual clients to open facilities. But since some supply points are now uncapacitated, it could also be that i sends *more* than n_i supply to other demand points. We argue that this artifact can also be handled without increasing the solution cost by much, by opening the facilities in a carefully-chosen subset of $\{i\} \cup \{\text{demand points satisfied by } i\}$ and closing down the remaining facilities. For *every* α (recall that the LBFL instance $\mathcal{I}_2$ is specified in terms of a parameter α), the resulting approximation factor for $\mathcal{I}_2$ (Theorem 5) is better than the guarantee obtained for $\mathcal{I}_2$ in Svitkina's algorithm; this in turn translates (by choosing α suitably) to an improved solution to the original instance.

We now discuss how we implement step (a), that is, how we obtain instance $\mathcal{I}_2$. As in [17], we arrive at $\mathcal{I}_2$ by computing a bicriteria solution to LBFL, but we obtain this bicriteria solution in a different fashion (see Section 3). The reduction from LBFL to UFL in [8,5] proceeds by setting the opening cost of facility i to $f_i + \frac{2\alpha}{1-\alpha} \cdot \sum_{j \in \mathcal{D}(i)} c_{ij}$, where $\mathcal{D}(i)$ is the set of M clients closest to i, solving the resulting UFL instance, and postprocessing using (single-facility) delete moves if such a move improves the solution cost. We modify this reduction subtly by creating a UFL instance, where facility i's opening cost is instead set to $f_i + 2\alpha M R_i(\alpha)$, where $R_i(\alpha)$ is the distance between i and the αM-closest client to it. As in the case of the earlier reduction, we argue that each open facility i in the resulting solution (obtained by solving UFL and postprocessing) serves at least αM clients. The overall bound we obtain on the total cost now includes various $R_i(\alpha)$ terms. Instead of plugging in the (weak) bound $M R_i(\alpha) \leq \frac{\sum_{j \in \mathcal{D}(i)} c_{ij}}{1-\alpha}$ (which would yield the same guarantee as that obtained via the earlier reduction), we are able to perform a tighter analysis by choosing α from a suitable distribution and leveraging the fact that $M \int_0^1 R_i(\alpha) d\alpha = \sum_{j \in \mathcal{D}(i)} c_{ij}$. (This can easily be derandomized, since there are only M combinatorially distinct choices for α.) These simple modifications yield a surprising amount of improvement in the approximation factor, which is reminiscent of the mileage provided by (random) α-points for various scheduling problems and UFL [15,16]. Also, we observe that one can obtain further improvements by using the local-search algorithm of [3,2] to solve the above UFL instance: this is because the resulting solution is then already postprocessed, which allows us to exploit the asymmetric bounds on the facility-opening and assignment costs provided by the local-search algorithm via scaling, and improve the approximation ratio.

Finally, we remark that the study of CDUFL may provide useful and interesting insights about CFL. CDUFL is a special case of CFL that despite its special structure inherits the intractability of CFL with respect to LP-based approximation guarantees. If one seeks to develop LP-based techniques and algorithms for CFL (which has been a long-standing and intriguing open question), then one needs to understand how one can leverage LP-based techniques for CDUFL, and it is plausible that LP-based insights developed for CDUFL may yield similar insights for CFL (and potentially LP-based approximation guarantees for CFL).

Related Work. LBFL was independently introduced by [8] and [5], who used it as a subroutine to solve the *maybecast* and *access network design* problems respectively. Their ideas lead to bicriteria guarantees for LBFL and play a preprocessing role both in Svitkina's algorithm [17] and (slightly indirectly) in our algorithm.

There is a large body of literature that deals with approximation algorithms for (metric) UFL, CFL and its variants; see [14] for a survey on UFL. The first constant approximation guarantee for UFL was obtained by Shmoys et al. [15] via an LP-rounding algorithm, and the current state-of-the-art is a 1.488-approximation algorithm due to Li [10]. Local-search techniques have also been utilized to obtain $O(1)$-approximation guarantees for UFL [9,3,2]. We apply some of the ideas of [3,2] in our algorithm. Starting with the work of Korupolu et al. [9], various local-search algorithms with constant approximation ratios have been devised for CFL, with the current-best approximation ratio being $5.83 + \epsilon$ [18]. Local-search approaches are however not known to work for LBFL; in the full version [1], we show that local search based on add, delete, and swap moves yields poor approximation guarantees. A related problem is *universal facility location* (UniFL), a generalization of UFL where the facility cost depends on the number of clients served by the facility. UniFL with non-decreasing functions was introduced by [6,12], and [12] obtained a constant approximation algorithm. We are not aware of any work on UniFL with arbitrary non-increasing functions (which generalizes LBFL). [4] give a constant approximation for the case where the cost-functions do not decrease too steeply (the constant depends on the steepness); notice that LBFL does not fall into this class so their results do not apply here.

2 Problem Definition and Notation

Recall that an LBFL instance $\mathcal{I}$ consists of a set $\mathcal{F}$ of facilities with facility-opening costs $\{f_i\}$, a set $\mathcal{D}$ of clients, metric connection (or assignment) costs $\{c_{ij}\}$ specifying the cost of assigning client j to facility i, and a (integer) parameter M. Our objective is to open a subset F of facilities and assign each client j to an open facility $i(j) \in F$, so that at least M clients are assigned to each open facility, and the total cost incurred, $\sum_{i \in F} f_i + \sum_j c_{i(j)j}$, is minimized.

Let F^* and C^* denote respectively the facility-opening and assignment cost of an optimal solution to $\mathcal{I}$; we will often refer to this solution as "the optimal solution" in the sequel. We sometimes abuse notation and also use F^* to denote the set of open facilities in this optimal solution. Let $OPT = F^* + C^*$ denote the total optimal cost. For a facility $i \in \mathcal{F}$, let $\mathcal{D}(i)$ be the set of M clients closest to i, and $R_i(\alpha)$ denote the distance between i and the $\lceil \alpha M \rceil$-closest client to i; that is, if $\mathcal{D}(i) = \{j_1, \ldots, j_M\}$, where $c_{ij_1} \leq \ldots \leq c_{ij_M}$, then $R_i(\alpha) = c_{ij_{\lceil \alpha M \rceil}}$ (for $0 < \alpha \leq 1$). Let $R^*(\alpha) = \sum_{i \in F^*} R_i(\alpha)$. Observe that each $R_i(\alpha)$ is an increasing function of α, $M \int_0^1 R_i(\alpha) d\alpha = \sum_{j \in \mathcal{D}(i)} c_{ij}$, and $R_i(\alpha) \leq \left(\sum_{j \in \mathcal{D}(i)} c_{ij}\right)/(M - \lceil \alpha M \rceil + 1) \leq \frac{\sum_{j \in \mathcal{D}(i)} c_{ij}}{M(1-\alpha)}$. Hence, $R^*(\alpha)$ is an increasing function of α, $M \int_0^1 R^*(\alpha) d\alpha \leq C^*$, and $R^*(\alpha) \leq \frac{C^*}{M(1-\alpha)}$.

3 Our Algorithm and the Main Theorem

We now give a high-level description of our algorithm using certain building blocks that are supplied in the subsequent sections.

(1) **Obtaining a Bicriteria Solution.** Construct a UFL instance with the same set of facilities and clients, and the same assignment costs as $\mathcal{I}$, where the opening cost of facility i is set to $f_i + 2\alpha M R_i(\alpha)$. Use the local-search algorithm for UFL in [3] or [2] with scaling parameter $\gamma > 0$ to solve the resulting UFL instance. (We set α, γ suitably to get the desired approximation; see Theorem 1.) Let $\mathcal{F}' \subseteq \mathcal{F}$ be the set of facilities opened in the UFL-solution. Claim 2 and Lemma 3 show that each $i \in F'$ serves at least αM clients.

(2) **Transforming to a Structured LBFL Instance.** We use the bicriteria solution obtained above to transform $\mathcal{I}$ into another structured LBFL instance $\mathcal{I}_2$ as in [17]. In the instance $\mathcal{I}_2$, we set the opening cost of each $i \in \mathcal{F}'$ to zero, and we "move" to i all the $n_i \geq \alpha M$ clients assigned to it, that is, all these clients are now co-located at i. So $\mathcal{I}_2$ consists of only the points in $\mathcal{F}'$ (which forms both the facility-set and client-set). We sometimes denote this instance by $\mathcal{I}_2(\alpha)$ to indicate explicitly that its specification depends on α.

(3) Solve $\mathcal{I}_2$ using the method described in Section 4. Obtain a solution to $\mathcal{I}$ by opening the same facilities and making the same client assignments as in the solution to $\mathcal{I}_2$.

Analysis. Our main theorem is as follows.

Theorem 1. *For any $\alpha \in (0.5, 1]$ and $\gamma > 0$, the above algorithm returns a solution to $\mathcal{I}$ of cost at most*

$$F^*\big(1 + \gamma h(\alpha)\big) + C^*\Big(2h(\alpha) - 1 + \tfrac{2}{\gamma}\Big) + 2\gamma\alpha M R^*(\alpha) h(\alpha) + 2\alpha M R^*(\alpha)$$

where $h(\alpha) = 1 + \frac{4}{\alpha} + \frac{4\alpha}{2\alpha - 1} + 4\sqrt{\frac{6}{2\alpha - 1}}$. Thus, we can compute efficiently a solution to $\mathcal{I}$ of cost at most: (i) $92.84 \cdot OPT$, by setting $\alpha = 0.75, \gamma = 3/h(\alpha)$; (ii) $82.6 \cdot OPT$, by letting γ be a suitable function of α, and choosing α randomly from the interval $[0.67, 1]$ according to the density function $p(x) = \frac{1}{\ln(1/0.67)x}$.

The roadmap for proving Theorem 1 is as follows. We first bound the cost of the bicriteria solution obtained in step (1) in terms of OPT (Lemma 3). This will allow us to bound the cost of an optimal solution to $\mathcal{I}_2$, and argue that mapping an $\mathcal{I}_2$-solution to a solution to $\mathcal{I}$ does not increase the cost by much (Lemma 4). The only missing ingredient is a guarantee on the cost of the solution to $\mathcal{I}_2$ found in step (3), which we supply in Theorem 5, whose proof appears in Section 4.

The following claim follows from essentially the same arguments as in [8,5].

Claim 2. Let S' be a delete-optimal *solution to the above UFL instance; that is, the total UFL-cost does not decrease by deleting any open facility of S'. Then, each facility of S' serves at least αM clients.*

The local-search algorithms for UFL in [3,2] have the same performance guarantees and both include a delete-move as a local-search operation, so upon termination, we obtain a delete-optimal solution.[1] Opening the same facilities and making the same client assignments as in the optimal solution to $\mathcal{I}$ yields a solution S to the UFL instance constructed in step (1) of the algorithm with facility cost $F^S \leq F^* + 2\alpha MR^*(\alpha)$ and assignment cost $C^S \leq C^*$. Combined with the analysis in [3,2], this yields the following. (For simplicity, we assume that local search terminates with a local optimum; standard arguments show that dropping this assumption increases the approximation by at most a $(1+\epsilon)$ factor.)

Lemma 3. *For a given parameter $\gamma > 0$, executing the local-search algorithm in [3,2] on the above UFL instance returns a solution with facility cost F_b and assignment cost C_b satisfying $F_b \leq F^* + 2\alpha MR^*(\alpha) + 2C^*/\gamma$, $C_b \leq \gamma\big(F^* + 2\alpha MR^*(\alpha)\big) + C^*$, where each open facility serves at least αM clients.*

Lemma 4 ([17]). *(i) The cost $C^*_{\mathcal{I}_2}$ of an optimal solution to $\mathcal{I}_2$ is at most $2(C_b + C^*)$. (ii) Any solution to $\mathcal{I}_2$ of cost C yields a solution to $\mathcal{I}$ of cost at most $F_b + C_b + C$.*

Theorem 5. *For any $\alpha > 0.5$, there is a $g(\alpha)$-approximation algorithm for $\mathcal{I}_2(\alpha)$, where $g(\alpha) = \frac{2}{\alpha} + \frac{2\alpha}{2\alpha-1} + 2\sqrt{\frac{2}{\alpha^2} + \frac{4}{2\alpha-1}}$.*

Remark 6. Our $g(\alpha)$-approximation ratio for $\mathcal{I}_2(\alpha)$ improves upon the approximation obtained in [17] by a factor of roughly 2 *for all* α. Thus, plugging in our algorithm for solving $\mathcal{I}_2$ in the LBFL-algorithm in [17] (and choosing a suitable α), already yields an improved approximation factor of 218 for LBFL.

Proof of Theorem 1. Recall that $h(\alpha) = 1 + \frac{4}{\alpha} + \frac{4\alpha}{2\alpha-1} + 4\sqrt{\frac{6}{2\alpha-1}}$. Note that $2g(\alpha) + 1 \leq h(\alpha)$ for all $\alpha \in [0,1]$; we use this upper bound throughout below. Combining Theorem 5 and the bounds in Lemmas 3 and 4, we obtain a solution to $\mathcal{I}$ of cost at most $F_b + \big(2g(\alpha)+1\big)C_b + 2g(\alpha)C^* \leq F_b + h(\alpha)C_b + \big(h(\alpha)-1\big)C^*$

$$\leq F^*\big(1+\gamma h(\alpha)\big) + C^*\Big(2h(\alpha) - 1 + \tfrac{2}{\gamma}\Big) + 2\gamma\alpha MR^*(\alpha)h(\alpha) + 2\alpha MR^*(\alpha).$$

Part (i) follows by plugging in the values of α and γ, and using the bound $R^*(\alpha) \leq \frac{C^*}{M(1-\alpha)}$. Let $\beta = 0.67$. For part (ii), we set $\gamma = \frac{K}{\sqrt{h(\alpha)}}$, where $K = \left(\ln^2(1/\beta) \cdot \mathrm{E}_\alpha\left[h(\alpha)\right] / \left(\frac{\int_\beta^1 h(x)dx}{1-\beta}\right)\right)^{\frac{1}{4}}$. Hence, the cost incurred is at most

$$F^*\big(1+K\sqrt{h(\alpha)}\big) + C^*\Big(2h(\alpha) - 1 + \tfrac{2}{K}\sqrt{h(\alpha)}\Big) + 2K\alpha MR^*(\alpha)\sqrt{h(\alpha)} + 2\alpha MR^*(\alpha).$$

[1] A subtle point is that typically local-search algorithms terminate only with an "approximate" local optimum. However, one can then execute all delete moves that improve the solution cost, and thereby obtain a delete-optimal solution.

We now bound the expected cost incurred when one chooses α randomly according to the stated density function. This will also yield an explicit expression for K (as a function of β), thus showing that K (and hence, γ) can be computed efficiently. We note that $\mathrm{E}\left[\sqrt{X}\right] \leq \sqrt{\mathrm{E}[X]}$ and utilize Chebyshev's Integral inequality (see [7]): if f and g are non-increasing and non-decreasing functions respectively from $[a,b]$ to $\mathbb{R}_+$, then $\int_a^b f(x)g(x)dx \leq \frac{(\int_a^b f(x)dx)(\int_a^b g(x)dx)}{b-a}$. Observe that $h(\alpha)$ decreases with α. Recall that $\beta = 0.67$. We have the following.

$$\mathrm{E}_\alpha[h(\alpha)] = c_2(\beta) := \tfrac{4}{\beta\ln(1/\beta)} - \tfrac{4}{\ln(1/\beta)} + \tfrac{8\sqrt{6}\left(\pi/4 - \tan^{-1}(\sqrt{2\beta-1})\right)}{\ln(1/\beta)} + \tfrac{2\ln(1/(2\beta-1))}{\ln(1/\beta)} + 1$$

$$\mathrm{E}_\alpha[\alpha M R^*(\alpha)] = M\Big(\int_\beta^1 R^*(x)dx\Big)/\ln(1/\beta) \leq C^*/\ln(1/\beta).$$

$$\mathrm{E}_\alpha\left[\alpha M R^*(\alpha)\sqrt{h(\alpha)}\right] \leq \Big[M\Big(\int_\beta^1 R^*(x)dx\Big)\tfrac{\int_\beta^1 dx\sqrt{h(x)}}{1-\beta}\Big]/\ln(1/\beta) \leq \frac{C^*\sqrt{c_3(\beta)}}{\ln(1/\beta)}, \quad \text{where}$$

$$c_3(\beta) := \frac{\int_\beta^1 h(x)dx}{1-\beta} = \Big[4\ln\Big(\frac{1}{\beta}\Big) + 4\sqrt{6}(1-\sqrt{2\beta-1}) + 3(1-\beta) + \ln\Big(\frac{1}{2\beta-1}\Big)\Big]/(1-\beta).$$

The second inequality follows since $\left(\int_\beta^1 dx\sqrt{h(x)}\right)/(1-\beta) = \mathrm{E}_{\alpha\sim\text{uniform in }[\beta,1]}\left[\sqrt{h(\alpha)}\right]$. These bounds yield $K = \left(\ln^2(1/\beta)c_2(\beta)/c_3(\beta)\right)^{0.25}$, and the total cost is at most

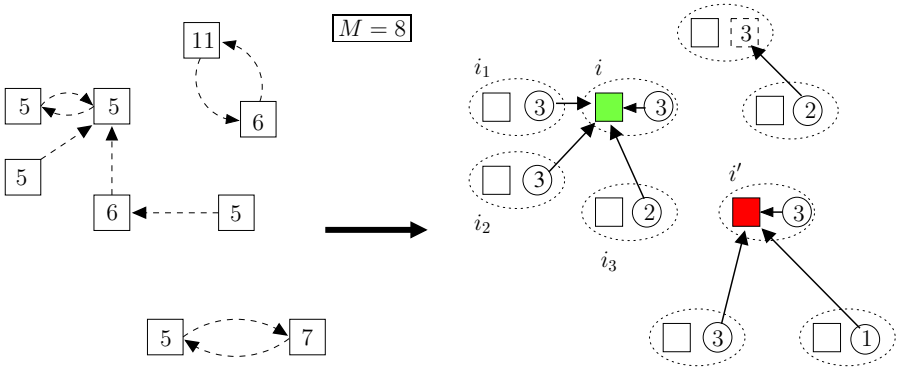
$$F^*\Big(1+\big(\tfrac{\ln^2(1/\beta)(c_2(\beta))^3}{c_3(\beta)}\big)^{\frac{1}{4}}\Big)+C^*\Big(2c_2(\beta)-1+4\big(\tfrac{c_2(\beta)c_3(\beta)}{\ln^2(1/\beta)}\big)^{\frac{1}{4}}+\tfrac{2}{\ln(1/\beta)}\Big) < 82.59(F^*+C^*). \quad \square$$

4 Solving Instance $\mathcal{I}_2(\alpha)$

We now describe our algorithm for solving instance $\mathcal{I}_2(\alpha)$ and analyze its performance guarantee, thereby proving Theorem 5. As mentioned earlier, one of the key differences between our algorithm and the one in [17] is that instead of reducing $\mathcal{I}_2$ to capacitated facility location (CFL), we solve $\mathcal{I}_2$ by reducing it to a new problem that we call *capacity-discounted UFL* (CDUFL). CDUFL is a special case of CFL where all facilities with non-zero opening cost are uncapacitated (i.e., have infinite capacity). Perhaps surprisingly, despite this special structure, CDUFL inherits the intractability of CFL with respect to LP-based approximation guarantees: the natural LP-relaxation for CDUFL has bad integrality gap, and there is no known LP-relaxation with constant integrality gap. However, we show in Section 4.2 that a simple local-search algorithm for CDUFL yields a better approximation ratio than the current-best approximation for CFL.

Recall that $\mathcal{I}_2$ has only the points in $\mathcal{F}' \subseteq \mathcal{F}$, and there are $n_i \geq \alpha M$ co-located clients at each $i \in \mathcal{F}'$. Let $l(i) = \min_{i' \in \mathcal{F}', i' \neq i} c_{ii'}$. To avoid confusion, we refer to the facilities and clients in the CDUFL instance as supply points and demand points respectively. The CDUFL instance created to solve $\mathcal{I}_2$ resembles

the CFL instance created in [17]; the difference is that supply points with non-zero opening costs are now uncapacitated. At each $i \in \mathcal{F}'$, we create an uncapacitated supply point with opening cost $\delta \min\{n_i, M\} l(i)$, where δ will be fixed later. If $n_i > M$ we create a second supply point at i with capacity $n_i - M$ and zero opening cost. If $n_i < M$, we create a demand point at i with demand $M - n_i$. Let $\mathcal{I}'$ denote this CDUFL instance (see Fig. 1). Let $\mathcal{F}^u$, $\mathcal{F}^c$ denote respectively the set of uncapacitated and capacitated supply points of $\mathcal{I}'$. Roughly speaking, satisfying a demand point i by non-co-located supply points translates to leaving facility i open in the $\mathcal{I}_2$ solution; hence, its demand is set to $M - n_i$, which is the number of additional clients it needs. Conversely, opening the uncapacitated supply point at i and supplying demand points from i translates to closing i in the $\mathcal{I}_2$ solution and transferring its co-located clients to other open facilities.



(a) $\mathcal{I}_2$ (b) $\mathcal{I}'$, and a solution S for $\mathcal{I}'$

Fig. 1. (a) An $\mathcal{I}_2$ instance. Each box denotes a facility and the number inside it is the number of co-located clients; $i \dashrightarrow i'$ indicates that i' is the closest facility to i.
(b) The corresponding $\mathcal{I}'$ instance. The boxes and circles represent supply points and demand points respectively, and points inside a dotted oval are co-located. A solid box denotes an uncapacitated supply point, and a dashed box denotes a capacitated facility whose capacity is shown inside the box. The number inside a circle is the demand of that demand point. The arrows indicate a solution S to $\mathcal{I}'$, where i and i' are the two open uncapacitated supply points.

Lemma 7 ([17]). *There exists a solution to $\mathcal{I}'$ with facility cost $F \le \delta C^*_{\mathcal{I}_2}$ and assignment cost $C \le C^*_{\mathcal{I}_2}$.*

Theorem 8. *(i) Given any CDUFL instance, one can efficiently compute a solution with facility-opening cost $\widehat{F} \le F^{\mathrm{sol}} + 2C^{\mathrm{sol}}$ and assignment cost $\widehat{C} \le F^{\mathrm{sol}} + C^{\mathrm{sol}}$, where F^{sol} and C^{sol} are the facility and assignment costs of an arbitrary solution to the CDUFL instance.*
(ii) Thus, Lemma 7 implies that one can compute a solution to $\mathcal{I}'$ with facility cost $F_{\mathcal{I}'}$ and assignment cost $C_{\mathcal{I}'}$ satisfying $F_{\mathcal{I}'} \le (2+\delta)C^*_{\mathcal{I}_2}$, $C_{\mathcal{I}'} \le (1+\delta)C^*_{\mathcal{I}_2}$.

4.1 Mapping an $\mathcal{I}'$-solution to an $\mathcal{I}_2$-solution

An $\mathcal{I}'$-solution need not directly translate to an $\mathcal{I}_2$ solution because an open supply point i may not supply (and hence, transfer) exactly n_i units of demand (e.g., i and i' in Fig. 1(b)). Since we have uncapacitated supply points, we have to consider both the cases where i supplies more than n_i demand (which is not encountered in [17]), and less than n_i demand. Suppose that we are given a solution S to $\mathcal{I}'$ with facility cost F^S and assignment cost C^S (see Fig. 1(b)). Again, we abuse notation and use F^S to also denote the set of supply points that are opened in S. Let N_i initialized to n_i keep track of the number of clients at location $i \in \mathcal{F}'$. Our goal is to reassign clients (using S as a template) so that at the end we have $N_i = 0$ or $N_i \geq M$ for each $i \in \mathcal{F}'$. We may assume that: (i) $\mathcal{F}^c \subseteq F^S$; (ii) if S opens an uncapacitated supply point located at some $i \in \mathcal{F}'$ with $n_i > M$, then the demand assigned to the capacitated supply point at i equals its capacity $n_i - M$; (iii) for each $i \in \mathcal{F}'$ with $n_i \leq M$, if the supply point at i is open then it serves the entire demand of the co-located demand point; and (iv) at most one *uncapacitated* supply point serves, maybe partially, the demand of any demand point; we say that this uncapacitated supply point satisfies the demand point. We reassign clients in three phases.

A1. **Removing Capacitated Supply Points.** Consider any $i \in \mathcal{F}'$ with $n_i > M$. Let i^1 and i^2 denote respectively the capacitated and uncapacitated supply points located at i. If i^1 supplies x units to the demand point at location i', we transfer x clients from location i to i'. Now if i^1 has $y > 0$ leftover units of capacity in S, then we "move" y clients to i^2 (which is not open in S). We update the N_is accordingly. This reassignment effectively gets rid of all capacitated supply points. Thus, there is now exactly one uncapacitated supply point and at most one demand point at each location $i \in \mathcal{F}'$; we refer to these simply as supply point i and demand point i below.

Let X_i be the total demand from other locations assigned to supply point i. Let $\mathcal{F}^G = \{i \in \mathcal{F}' : N_i < X_i\}$, $\mathcal{F}^R = \{i \in \mathcal{F}' : N_i \geq X_i > 0\}$, and $\mathcal{F}^B = \{i \in \mathcal{F}' : X_i = 0\}$ ($\mathcal{F}^B$ is the set of supply points not opened in S). Note that $N_i \geq \min\{n_i, M\} \geq \alpha M$ for all $i \in \mathcal{F}'$, and $N_i = \min\{n_i, M\}$ for all $i \in \mathcal{F}^R \cup \mathcal{F}^G$.

A2. **Taking Care of $\mathcal{F}^R$ and Demand Points Satisfied by $\mathcal{F}^R$.** For each $i \in \mathcal{F}^R$, if i supplies x units to demand point i', we move x clients from i to i', and update $N_i, N_{i'}$. We now have $N_i = \min\{n_i, M\} - X_i$ residual clients at each $i \in \mathcal{F}^R$, which we must reduce to 0, or increase to at least M.

We follow the same procedure as in [17]. For each $i \in \mathcal{F}^R$, we include an edge (i, i') where $i' \in \mathcal{F}'$ is the facility nearest to i (recall that $c_{ii'} = l(i)$). We use an arbitrary but fixed tie-breaking rule here, so each component of the resulting digraph is a directed tree rooted at either (i) a node $r \in \mathcal{F}' \setminus \mathcal{F}^R$, or (ii) a 2-cycle $(r, r'), (r', r)$, where $r, r' \in \mathcal{F}^R$. We break up each component Γ into a collection of smaller components. Essentially, we move the residual clients of supply points in Γ bottom-up from the leaves up to the root, cut off Γ at the first node u that accumulates at least M clients, and recurse on

the portion of Γ not containing u. More precisely, let Γ_u denote the subtree of Γ rooted at node $u \in \Gamma$ (if u belongs to a 2-cycle then we do not include the other node of this 2-cycle in Γ_u). If $\sum_{i\in\Gamma} N_i < M$, or if Γ is of type (i) and all children u of the root satisfy $\sum_{i\in\Gamma_u} N_i < M$, we leave Γ unchanged. Otherwise, let u be a deepest (i.e., furthest from root) node in Γ such that $\sum_{i\in\Gamma_u} N_i \geq M$. We delete the arc leaving u. If this disconnects u from $\Gamma\setminus\Gamma_u$, then we recurse on $\Gamma \setminus \Gamma_u$. Otherwise u must belong to the root 2-cycle of Γ. Let r' be the other node of this 2-cycle. If $\sum_{i\in\Gamma_{r'}} N_i \geq M$, we delete r''s outgoing arc (thus splitting Γ into Γ_u and $\Gamma_{r'}$).

After applying the above procedure (to all components), if we are left with a component of type (ii) with $\sum_{i\in \text{ component}} N_i \geq M$, we convert it to type (i) by arbitrarily deleting one of the arcs of the 2-cycle. Let T be a component at the end of this process. If T rooted at a node r, we move the N_i residual clients of each non-root node $i \in T$ to r. Otherwise, T is of type (ii) with root $\{r, r'\}$, and we have $\sum_{i\in T} N_i < M$. Let $i' \in \mathcal{F}^B$ be the location nearest to $\{r, r'\}$; we move the N_i residual clients of each $i \in T$ to i'. Update the N_is to reflect the above reassignment. Observe that we now have $N_i = 0$ or $N_i \geq M$ for each $i \in \mathcal{F}^R$, and each $i \in \mathcal{F}^B$ has $n_i \geq M$, or is a demand point satisfied by a supply point in $\mathcal{F}^G$.

For example, executing step (A1 and) A2 on the solution shown in Fig. 1(b) results in $i' \in \mathcal{F}^R$ having one client left after moving its co-located clients to the bottom two facilities; this residual client is then transferred to i_3.

A3. **Taking Care of $\mathcal{F}^G$ and Demand Points Satisfied by $\mathcal{F}^G$.** For $i \in \mathcal{F}^G$, let $D(i)$ be the set of demand points $j \in \mathcal{F}'$, $j \neq i$ satisfied by i, and let $D'(i) = \{j \in D(i) : N_j < M\}$. Note that $D(i) \subseteq \mathcal{F}^B$. Phase A2 may only increase N_j for all j in $\mathcal{F}^B \cup \mathcal{F}^G$, so $N_j \geq \alpha M$ for all $j \in \mathcal{F}^G \cup \big(\bigcup_{i\in\mathcal{F}^G} D(i)\big)$. Fix $i \in \mathcal{F}^G$. We reassign clients so that $N_j = 0$ or $N_j \geq M$ for all $j \in \{i\} \cup D'(i)$, without decreasing N_j for $j \in D(i) \setminus D'(i)$. Doing this for all supply points in $\mathcal{F}^G$ will complete our task. Define $Y_j = M - N_j$ for $j \in D'(i)$. (1) If $\sum_{j\in D'(i)} Y_j \leq N_i$, for each $j \in D'(i)$, if i supplies x units to j, we transfer x clients from i to j. If i is now left with less than M residual clients, we move these residual clients to the location in $D(i)$ nearest to i. (2) If $\sum_{j\in D'(i)} Y_j > N_i$, set $i_0 = i$, and $D'(i) = \{i_1, \ldots, i_t\}$, where $c_{i_1 i} \leq \ldots \leq c_{i_t i}$. Let $\ell = t - \left\lfloor \frac{\sum_{r=0}^{t} N_{i_r}}{M} \right\rfloor = \left\lceil \frac{\sum_{r=1}^{t} Y_{i_r} - N_{i_0}}{M} \right\rceil$ (so $1 \leq \ell < t$ since $N_{i_0} + N_{i_1} \geq M$), which is the unique index such that $\sum_{r=\ell+1}^{t} Y_{i_r} \leq \sum_{r=0}^{\ell} N_{i_r} < \sum_{r=\ell+1}^{t} Y_{i_r} + M$. This enables us to transfer Y_{i_q} clients to each i_q, $q = \ell+1, \ldots, t$ from the locations $i_\ell, \ldots, i_0$—we do this by transferring all clients of i_r (where $1 \leq r \leq \ell$) before considering i_{r-1}—and be left with at most M residual clients in $\{i_0, \ldots, i_\ell\}$. We argue that these residual clients are all concentrated at i_0 and i_1, with i_1 having at most $(1-\alpha)M$ residual clients. We transfer these residual clients to $i_{\ell+1}$.

In the solution shown in Fig. 1(b), we have $Y_{i_1} = 3 = Y_{i_2}$, $Y_{i_1} = 1$, $N_i = 5$, so case 2 applies; we transfer 1 client to i_3 and 9 clients to i_2 from $\{i, i_1\}$.

Theorem 9. *The above algorithm returns an $\mathcal{I}_2$-solution of cost at most $\frac{F^S}{\delta\alpha} + C^S\big(\frac{1}{\alpha} + \frac{2\alpha}{2\alpha-1}\big)$. Thus, taking S to be the solution mentioned in part (ii) of Theorem 8, and $\delta = \sqrt{\frac{2/\alpha}{1/\alpha+(2\alpha)/(2\alpha-1)}}$, we obtain a solution to $\mathcal{I}_2(\alpha)$ satisfying the approximation bound stated in Theorem 5.*

Proof. Let S_2 denote the solution computed for $\mathcal{I}_2$. For a supply point i opened in S, let C_i^S denote the cost incurred in supplying demand from i to the demand points satisfied by i. At various steps, we transfer clients between locations according to the assignment in the CDUFL solution S, and the cost incurred in doing so can be charged to the C_i^Ss of the appropriate supply points. So the cost of phase A1 is $\sum_{i\in\mathcal{F}^c} C_i^S$, and the cost of the first step of phase A2 is $\sum_{i\in\mathcal{F}^R} C_i^S$. As in [17], we can bound the remaining cost of phase A2, incurred in transferring clients according to the tree edges, by $F^S/\delta\alpha + \big(\sum_{i\in\mathcal{F}^R} C_i^S\big)/(2\alpha-1)$.

Finally, consider phase A3 and some $i \in \mathcal{F}^G$. If $\sum_{j\in D'(i)} Y_j \le N_i$, then the cost incurred is at most $C_i^S + M\cdot\frac{C_i^S}{X_i} \le C_i^S\big(1+\frac{1}{\alpha}\big)$ (as $X_i > N_i \ge \alpha M$). Now consider the case $\sum_{j\in D'(i)} Y_j > N_i$. For any $i_q \in \{i_{\ell+1},\ldots,i_t\}$ and any $i_r \in \{i_0,\ldots,i_\ell\}$, we have $c_{i_r i_q} \le 2c_{i i_q}$, so the cost of transferring $Y_{i_q} \le M - n_{i_q}$ clients to each i_q, $q=\ell+1,\ldots,t$ is at most $2C_i^S$. Observe that $(t-\ell+1)M > \sum_{r=0}^{t} N_{i_r}$, i.e., $M + \sum_{q=\ell+1}^{t} Y_{i_r} > \sum_{r=0}^{\ell} N_{i_r}$, so after this reassignment, there are less than M residual clients in $i_0,\ldots,i_\ell$. By our order of transferring clients, all these residual clients are at i_0, i_1 (otherwise we would have at least $N_{i_0}+N_{i_1} \ge M$ residual clients) with at most $M - N_{i_0} \le (1-\alpha)M$ of them located at i_1. The cost of reassigning these residual clients is at most $(1-\alpha)Mc_{ii_1} + Mc_{ii_{\ell+1}} \le (1-\alpha)M\cdot\frac{C_i^S}{\sum_{r=1}^{t} Y_{i_r}} + M\cdot\frac{C_i^S}{\sum_{r=\ell+1}^{t} Y_{i_r}}$, since C_i^S is the total cost of supplying at least Y_{i_r} demand to each i_r, $r=1,\ldots,t$. The latter expression is at most $C_i^S\big(\frac{1-\alpha}{\alpha} + \frac{1}{2\alpha-1}\big)$, since $\sum_{r=1}^{t} Y_{i_r} > N_{i_0} \ge \alpha M$, $\sum_{r=\ell+1}^{t} Y_{i_r} > \sum_{r=0}^{\ell} N_{i_r} - M \ge (2\alpha-1)M$.) Thus, the cost of S_2 is at most $\frac{F^S}{\delta\alpha} + \sum_{i\in\mathcal{F}^c} C_i^S + \sum_{i\in\mathcal{F}^R} C_i^S\cdot\big(1+\frac{1}{2\alpha-1}\big) + \sum_{i\in\mathcal{F}^G} C_i^S\cdot\max\big\{1+\frac{1}{\alpha}, 2+\frac{1-\alpha}{\alpha}+\frac{1}{2\alpha-1}\big\} \le \frac{F^S}{\delta\alpha} + C^S\big(\frac{1}{\alpha}+\frac{2\alpha}{2\alpha-1}\big)$. So if S is the solution given by part (ii) of Theorem 8, the cost of S_2 is at most $\big(\frac{2}{\delta\alpha} + \frac{1}{\alpha} + (1+\delta)(\frac{1}{\alpha}+\frac{2\alpha}{2\alpha-1})\big)C^*_{\mathcal{I}_2}$, and plugging in the value of δ yields the $g(\alpha) = \frac{2}{\alpha} + \frac{2\alpha}{2\alpha-1} + 2\sqrt{\frac{2}{\alpha^2}+\frac{4}{2\alpha-1}}$ approximation stated in Theorem 5. □

4.2 A Local-Search Based Approximation Algorithm for CDUFL

We now describe our local-search algorithm for CDUFL, which leads to the proof of Theorem 8. Let $\widehat{\mathcal{F}} = \widehat{\mathcal{F}}^u \cup \widehat{\mathcal{F}}^c$ be the facility-set of the CDUFL instance, where $\widehat{\mathcal{F}}^u \cap \widehat{\mathcal{F}}^c = \emptyset$. Here, $\widehat{\mathcal{F}}^u$ are the uncapacitated facilities with opening costs $\{\widehat{f}_i\}$, and facilities in $\widehat{\mathcal{F}}^c$ have (finite) capacities $\{u_i\}$ and zero opening costs. Let $\widehat{\mathcal{D}}$ be the set of clients and $\widehat{c}_{ij}$ be the cost of assigning client j to facility i. The goal is to open facilities and assign clients to open facilities (respecting the capacities) so as to minimize the sum of the facility-opening and client-assignment costs. We can find the best assignment of clients to open facilities by solving a network flow problem, so we focus on determining the set of facilities to open.

The local-search algorithm consists of add(i'), delete(i), swap(i, i') moves, which respectively, add a facility i' not currently open, delete a facility i that is currently open, and swap facility i that is open with facility i' that is not open. We note that *all* previous (local-search) algorithms for CFL with non-uniform capacities use moves that are more complicated than the moves above. The algorithm repeatedly executes the best cost-improving move until no such move exists. We may assume without loss of generality that each client has unit demand.

Analysis. Let $\widehat{S}$ denote a local-optimum returned by the algorithm, with facility-opening cost $\widehat{F}$ and assignment cost $\widehat{C}$. Let sol be an arbitrary CDUFL solution, with facility-cost F^{sol} and assignment cost C^{sol}. We also use $\widehat{F}$ and F^{sol} to denote the set of open facilities in $\widehat{S}$ and sol respectively. We may assume that $\widehat{\mathcal{F}}^c \subseteq \widehat{F} \cap F^{\mathrm{sol}}$. For a facility i, we use $\widehat{\mathcal{D}}_{\widehat{S}}(i)$ and $\widehat{\mathcal{D}}_{\mathrm{sol}}(i)$ to denote respectively the (possibly empty) set of clients served by i in $\widehat{S}$ and sol. For a client j, let $\widehat{C}_j$ and C_j^{sol} be the assignment cost of j in $\widehat{S}$ and sol respectively.

We borrow ideas from the analysis of the corresponding local-search algorithm for UFL in [2], but to handle capacities we need to reassign clients more carefully to analyze the change in assignment cost due to a local-search move. In particular, unlike the analysis in [2], where upon deletion of a facility $s \in \widehat{F}$ we reassign only the clients currently assigned to s, in our case (as in the analysis of local-search algorithms for CFL), we need to perform a more "global" reassignment (i.e., even clients not assigned to s may get reassigned) along certain paths in a suitable graph. This also means that we need to construct a suitable mapping between paths instead of the client-mapping considered in [2].

Consider a directed graph G with node-set $\widehat{\mathcal{D}} \cup \widehat{\mathcal{F}}$, and arcs from i to all clients in $\widehat{\mathcal{D}}_{\widehat{S}}(i)$ and arcs from all clients in $\widehat{\mathcal{D}}_{\mathrm{sol}}(i)$ to i, for every facility i. Via standard flow-decomposition, we can decompose G into a collection of (simple) paths $\mathcal{P}$, and cycles $\mathcal{R}$, so that (i) each facility i appears as the starting point of $\max\{0, |\widehat{\mathcal{D}}_{\widehat{S}}(i)| - |\widehat{\mathcal{D}}_{\mathrm{sol}}(i)|\}$ paths, and the ending point of $\max\{0, |\widehat{\mathcal{D}}_{\mathrm{sol}}(i)| - |\widehat{\mathcal{D}}_{\widehat{S}}(i)|\}$ paths, and (ii) each client j appears on a unique path P_j or on a cycle. Let $\mathcal{P}^{\mathsf{st}}(s) \subseteq \mathcal{P}$ and $\mathcal{P}^{\mathsf{end}}(o) \subseteq \mathcal{P}$ denote respectively the collection of paths starting at s and ending at o, and $\mathcal{P}(s, o) = \mathcal{P}^{\mathsf{st}}(s) \cap \mathcal{P}^{\mathsf{end}}(o)$. For a path $P = \{i_0, j_0, i_1, j_1, \ldots, i_k, j_k, i_{k+1} := o\} \in \mathcal{P}$, define $\widehat{\mathcal{D}}(P) = \{j_0, \ldots, j_k\}$, $\mathit{head}(P) = j_0$, and $\mathit{tail}(P) = j_k$. A *shift* along P means that we reassign client j_r to i_{r+1} for each $r = 0, \ldots, k$ (opening o if necessary). Note that this is feasible, since if $o \in \widehat{\mathcal{F}}^c$, we know that $|\widehat{\mathcal{D}}_{\widehat{S}}(o)| \leq |\widehat{\mathcal{D}}_{\mathrm{sol}}(o)| - 1 \leq u_o - 1$. Let $\mathit{shift}(P) := \sum_{j \in \widehat{\mathcal{D}}(P)} \big(C_j^{\mathrm{sol}} - \widehat{C}_j\big)$ be the increase in assignment cost due to this reassignment, which is an upper bound on the actual increase in assignment cost if o is added to $\widehat{F}$. Let $\mathit{cost}(P) := \sum_{j \in \widehat{\mathcal{D}}(P)} \big(C_j^{\mathrm{sol}} + \widehat{C}_j\big)$. We define a shift along a cycle $R \in \mathcal{R}$ similarly, letting $\mathit{shift}(R) := \sum_{j \in \widehat{\mathcal{D}} \cap R} \big(C_j^{\mathrm{sol}} - \widehat{C}_j\big)$. By considering a shift operation for every path and cycle in $\mathcal{P} \cup \mathcal{R}$ (i.e., suitable add moves) and adding the resulting inequalities, we get the following result.

Lemma 10. $\widehat{C} \leq F^{\mathrm{sol}} + C^{\mathrm{sol}}$.

To bound $\widehat{F}$, we only need paths starting at facilities in $\widehat{F} \setminus F^{\text{sol}}$. Note that facilities in $(\widehat{F} \setminus F^{\text{sol}}) \cup (F^{\text{sol}} \setminus \widehat{F})$ are *uncapacitated*. To avoid excessive notation, for a facility $o \in F^{\text{sol}} \setminus \widehat{F}$, we now use $\mathcal{P}^{\mathsf{end}}(o)$ to refer to the collection of paths ending in o that start in $\widehat{F} \setminus F^{\text{sol}}$. (As before, $\mathcal{P}(s,o)$ is the set of paths that start at s and end at o.) Let $\text{capt}_s \subseteq F^{\text{sol}} \setminus \widehat{F}$ be the facilities captured by s. For any $o \in F^{\text{sol}} \setminus \widehat{F}$, we can obtain a 1-1 mapping $\pi : \mathcal{P}^{\mathsf{end}}(o) \mapsto \mathcal{P}^{\mathsf{end}}(o)$ such that if $P \in \mathcal{P}(s,o)$, $\pi(P) = P' \in \mathcal{P}(s',o)$ then (i) if $o \notin \text{capt}_s$, we have $s \neq s'$; (ii) if $s = s'$, then $P = P'$; and (iii) $\pi(P') = P$. Say that $o \in F^{\text{sol}} \setminus \widehat{F}$ is *captured* by s if $|\mathcal{P}(s,o)| > \frac{|\mathcal{P}^{\mathsf{end}}(o)|}{2}$. Call a facility in $\widehat{F} \setminus F^{\text{sol}}$ *good* if $\text{capt}_s = \emptyset$, and *bad* otherwise. For a bad facility s, let $o_s \in \text{capt}_s$ be the facility nearest to s.

Lemma 11. *Let s be a facility in $\widehat{F} \setminus F^{\text{sol}}$.*

$$\textit{If } s \textit{ is good, } \widehat{f}_s \leq \sum_{P \in \mathcal{P}^{\mathsf{st}}(s)} \textit{shift}(P) + \sum_{o \notin \widehat{F}, P \in \mathcal{P}(s,o)} \textit{cost}\big(\pi(P)\big). \tag{1}$$

$$\textit{If } s \textit{ is bad, } \widehat{f}_s \leq \sum_{o \in \text{capt}_s} \widehat{f}_o + \sum_{P \in \mathcal{P}^{\mathsf{st}}(s)} \textit{shift}(P) + \sum_{\substack{o \notin \widehat{F}, P \in \mathcal{P}(s,o): \\ \pi(P) \neq P}} \textit{cost}\big(\pi(P)\big) + \sum_{\substack{o \in \text{capt}_s \setminus \{o_s\} \\ P \in \mathcal{P}(s,o): \pi(P) = P}} \textit{cost}(P). \tag{2}$$

Proof Sketch of Theorem 8. We focus on part (i); part (ii) follows directly from part (i) and Lemma 7. Lemma 10 bounds $\widehat{C}$. Consider adding (1) for all good facilities and (2) for all bad facilities, and the vacuous equality $\widehat{f}_i = \widehat{f}_i$ for all $i \in \widehat{F} \cap F^{\text{sol}}$. The LHS of the resulting inequality is precisely $\widehat{F}$. The $\widehat{f}_i$s on the RHS add up to give at most F^{sol}. One can argue that each path $P \in \bigcup_{s \in \widehat{F} \setminus F^{\text{sol}}} \mathcal{P}^{\mathsf{st}}(s)$ contributes at most $\textit{shift}(P) + \textit{cost}(P) = 2 \sum_{j \in \widehat{\mathcal{D}}(P)} C_j^{\text{sol}}$ to the RHS. Thus the RHS is at most $F^{\text{sol}} + 2C^{\text{sol}}$, and we obtain that $\widehat{F} \leq F^{\text{sol}} + 2C^{\text{sol}}$. □

References

1. Ahmadian, S., Swamy, C.: Improved approximation guarantees for lower-bounded facility location. CS arXiV (September 2012)
2. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM Journal on Computing 33(3), 544–562 (2004)
3. Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location problems. SIAM Journal on Computing 34(4), 803–824 (2005)
4. Guha, S., Meyerson, A., Munagala, K.: Facility location with demand dependent costs and generalized clustering (2000) (manuscript)
5. Guha, S., Meyerson, A., Munagala, K.: Hierarchical placement and network design problems. In: Proceedings of the 41st FOCS, pp. 603–612 (2000)
6. Hajiaghayi, M., Mahdian, M., Mirrokni, V.: The facility location problem with general cost functions. Networks 42, 42–47 (2003)
7. Hardy, G., Littlewood, J., Pólya, G.: Inequalities. Cambridge Univ. Press (1952)

8. Karger, D.R., Minkoff, M.: Building Steiner trees with incomplete global knowledge. In: Proceedings of the 41st FOCS, pp. 613–623 (2000)
9. Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. Journal of Algorithms 37(1), 146–188 (2000)
10. Li, S.: A 1.488 approximation algorithm for the uncapacitated facility location problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 77–88. Springer, Heidelberg (2011)
11. Lim, A., Wang, F., Xu, Z.: A transportation problem with minimum quantity commitment. Transportation Science 40(1), 117–129 (2006)
12. Mahdian, M., Pál, M.: Universal facility location. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 409–421. Springer, Heidelberg (2003)
13. Mirchandani, P., Francis, R. (eds.): Discrete Location Theory. John Wiley and Sons, Inc., New York (1990)
14. Shmoys, D.B.: The design and analysis of approximation algorithms: facility location as a case study. In: Hosten, S., Lee, J., Thomas, R. (eds.) Trends in Optimization. AMS Proc. of Symposia in Applied Math., vol. 61, pp. 85–97 (2004)
15. Shmoys, D.B., Tardos, É., Aardal, K.I.: Approximation algorithms for facility location problems. In: Proceedings of the 29th STOC, pp. 265–274 (1997)
16. Sviridenko, M.: An improved approximation algorithm for the metric uncapacitated facility location problem. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 240–257. Springer, Heidelberg (2002)
17. Svitkina, Z.: Lower-bounded facility location. Trans. on Algorithms 6(4) (2010)
18. Zhang, J., Chen, B., Ye, Y.: A multi-exchange local search algorithm for the capacitated facility location problem. Math. of Oper. Research 30, 389–403 (2005)

A 4-Approximation for the Height of Drawing 2-Connected Outer-Planar Graphs*

Therese Biedl

David R. Cheriton School of Computer Science, University of Waterloo,
Waterloo, ON N2L 3G1, Canada
biedl@uwaterloo.ca

Abstract. A graph drawing algorithm aims to create a picture of the graph, usually with vertices drawn at grid points while keeping the grid-size small. Many algorithms are known that create planar drawings of planar graphs, but most of them bound the height of the drawing in terms of the number of vertices. In this paper, we give an algorithm that draws 2-connected outer-planar graphs such that the height is a 4-approximation of the optimal height.

1 Introduction

Graph drawing is the art of creating a pretty picture of a graph. Since "pretty" is hard to define, common measures used are to minimize the number of edge-crossings and to keep the area small (presuming all coordinates are integers.)

It has been known for many years that any planar graph has a straight-line drawing without crossing in an $O(n) \times O(n)$-grid [13,20]. It is also known that an $\Omega(n) \times \Omega(n)$-grid is required for some planar graphs [12]. Many other papers have tried to decrease the constants for the width and height, sometimes by trading off a smaller width for a larger height. See e.g. [5,4].

For some subclasses of planar graphs, drawings with $o(n^2)$ area are possible. In an earlier paper, it was shown that any outer-planar graph has a so-called visibility representation in an $O(\log n) \times O(n)$-grid [1]. Many other papers have since dealt with drawing subclasses of planar graphs in $o(n^2)$ area, such as straight-line drawings of outer-planar graphs [6,10,14], and drawings of series-parallel graphs [2,11,22].

For most of these results, the output of the algorithm is a drawing that is guaranteed to have area $O(f(n))$, where $f(n)$ is some function in the number of vertices n. To show that such an algorithm is good, the usual approach has been to give an example of a graph that is required to have area $\Omega(g(n))$ in any drawing, for some function $g(n)$ that is close to $f(n)$. Thus, the usual approach has been to give bounds that are optimal in the worst-case. They are also approximation-algorithms for some classes of graphs (such as $\Omega(n)$-outer-planar graphs), but for many graphs the given bounds may be significantly too large.

* Research partially supported by NSERC and by the Ross and Muriel Cheriton Fellowship.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 272–285, 2013.

Relatively few papers exist that draw all graphs with optimal area or height (or at least provably approximate it.) Specifically, a drawing of a graph is called a *drawing of height* h if all y-coordinates are in $\{1, \ldots, h\}$. (Here we always assume that the drawing has been rotated so that the height is no larger than the width.) For some applications of graph drawing minimizing the height is more important than minimizing the area. For example, for metro maps that are hung in the head space of metro cars, there is only very limited height available, while the width can be quite large. For this reason some papers have focused on making the smaller dimension as small as possible (e.g. [5]).

We are not aware of any proof that minimizing the height is NP-hard, though minimizing the area is NP-hard, both for orthogonal drawings if crossings are allowed [9] and for straight-line drawings [16]. One of the few papers that optimize the height/area for all graphs in a subclass is by Mondal et al. [18] and works for planar graphs of treewidth 3 that are triangulated. Very recently, Mondal et al. showed how to compute drawings of minimum height for trees [17].

Quite closely related to drawings of height h are *h-level drawings*, which are drawings of height h where edges must connect different levels (and for *proper* drawings, edges must connect adjacent levels.) Testing whether a graph has a proper level drawing is NP-hard [15], but given an h, testing whether a graph has drawing on h levels is fixed-parameter tractable in h [7].

The latter paper was among the first to prove the strong connection between the height of drawings and the so-called pathwidth $pw(G)$ of a graph G. In particular, any planar graph that has a drawing of height h has pathwidth at most h [8]. However, the pathwidth is not always proportional to the minimum height: There exists a planar graph of pathwidth 3 that requires $\Omega(n)$ width and height in any planar drawing [2]. But for trees, Suderman showed that the pathwidth approximates the optimum height: Any tree T has a planar drawing of height at most $\frac{3}{2}pw(T) - 1$ [21].

This paper proves a similar result as Suderman, but for a 2-connected outer-planar graph G (detailed definitions are given below.) Specifically, G has a flat visibility representation of height $4pw(G) - 3$. The algorithm therefore produces a height that is within a factor of 4 of the optimum. (The algorithm works for any outer-planar graph, but the proof of the 4-approximation bound required 2-connectivity.) Furthermore, the flat visibility representation can be transformed into straight-line drawings of the same height.

2 Definitions

The reader is assumed to be familiar with basic graph-theoretic terms. In the following, let $G = (V, E)$ be a simple graph with n vertices V and m edges E. Throughout the paper, G is always *planar*, i.e., it can be drawn without crossing. Indeed, G is always *outer-planar*, i.e., it has a drawing without crossings such that all vertices are on the *outer-face* (the infinite connected region outside the drawing.) Any finite region defined by such a drawing is called an *interior face*, and a face is often identified with the set of vertices and edges that are adjacent to it.

A graph is called *maximal outer-planar* if one cannot add any edges to it and retain an outer-planar simple graph. In a maximal outer-planar graph, the outer-face consists of a simple cycle of length n, and every interior face is a triangle. The *dual tree* of a maximal outer-planar graph consists of placing a vertex for every interior face and connecting two vertices if and only if the corresponding faces share an edge. It is easy to see that the result is indeed a tree and has maximum degree 3.

A graph is said to have *pathwidth* k if there exists an order of the vertices $v_1, \ldots, v_n$ such that for any $j \geq k$, there are at most k vertices among $\{v_1, \ldots, v_j\}$ that have a neighbour in $\{v_{j+1}, \ldots, v_n\}$. For trees, the pathwidth can be described using the notation of a *main path* introduced by Suderman [21].

Definition 1. *Let T be a tree of pathwidth $p > 0$. A* main path *of T is a path P such that every component of $T - P$ has pathwidth at most $p - 1$.*

Every tree of pathwidth $p > 0$ has a main path [21], but it is not unique. One may assume that a main path ends at leaves of the dual tree, for if it doesn't, then it can simply be extended into a leaf and remains a main path. Fig. 1 illustrates a maximal outer-planar graph and a main path in its dual tree.

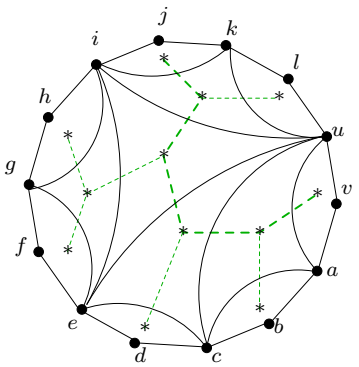


Fig. 1. A maximal outer-planar graph, its dual tree (dashed), and a main path (thick dashed.)

A *drawing* of a graph consists of assigning a point or an axis-aligned box to every vertex, and a curve between the points/boxes of u and v to every edge (u, v). The drawing is called *planar* if no two elements of the drawing intersect unless the corresponding elements of the graph do. Thus in a planar drawing no vertex points/boxes coincide, no edge curve self-intersects, no edge curves intersect each other (except at common endpoints), and no edge curve intersects a vertex point/box other than its endpoints. In this paper all drawings are required to be planar.

The most commonly considered type of drawing is a *straight-line drawing*, where vertices are represented by points and edges are drawn as straight-line

segments. This paper also uses *visibility representations*, where vertices are represented by axis-aligned boxes and edges are drawn as horizontal or vertical straight-line segments. A visibility representation is called a *flat visibility representation* if every vertex-box is degenerated into a horizontal segment. See also Fig. 2. (In our drawings, we thicken boxes slightly, so that horizontal segments appear as boxes of small height.)

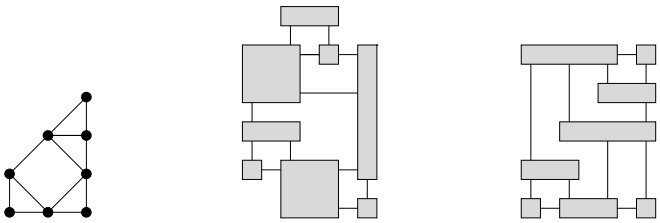


Fig. 2. The same graph in a straight-line drawing, a visibility representation, and a flat visibility representation

In all drawings, the defining elements (i.e., points of vertices, corners of boxes of vertices, and attachment points of edges to vertex-boxes) must be placed at points with integer coordinates. A drawing is said to have *width* w and *height* h if (possibly after translation) all such points are placed on the $[1, w] \times [1, h]$-grid. The height is thus measured by the number of *rows*, i.e., horizontal lines with integer y-coordinates that are occupied by the drawing.

3 Visibility Representations

The algorithm to create a flat visibility representation of an outer-planar graph G first converts G into a maximal outer-planar graph H by adding edges. Then it computes the dual tree T of H, which has maximum degree 3. Then it draws H recursively as follows:

- If T is a path, then directly create a drawing of height 2.
- If $pw(T) \geq 1$, then draw the graph of a main path P of T with height 2, and merge the subgraphs defined by the components of $T - P$ after drawing them recursively.

To allow the last merging step to be done with adding too much height, there will be restrictions on an edge (u, v) on the outer-face (with u before v in clockwise order) as follows: $\{u, v\}$ is said to *span the top row* if the box of u occupies the top left corner and the box of v occupies the top right corner. For example, in the rightmost picture of Fig. 2, the two vertices in the top span the top row. We also say that (u, v) is *adjacent to a main path* if there exists a main path of T that contains the interior face that is adjacent to u.

The following main lemma implies a recursive drawing algorithm.

Lemma 1. *Let H be a maximal outer-planar graph with edge (u, v) on the outer-face, with u before v in clockwise order. Let T be the dual tree of H.*

1. *H has a flat visibility representation with $\{u, v\}$ spanning the top row that has height $\max\{2, 4pw(T)\}$.*
2. *If (u, v) is adjacent to a main path of T, then H has a flat visibility representation with $\{u, v\}$ in the top row that has height $\max\{3, 4pw(T) - 3\}$.*
3. *If (u, v) is adjacent to a main path of T, then H has a flat visibility representation with $\{u, v\}$ spanning the top row that has height $\max\{4, 4pw(T) - 2\}$.*

Proof. As a first ingredient, let us draw a graph G_P whose dual tree T is a path $P = f_1, f_2, \ldots, f_k$. Here each f_i is a vertex of T and hence a face of G_P; the name f_i is used for both vertex and face since the meaning should be clear from the context.

Create a flat visibility representation of G_P with height 2 in the obvious way: Draw the faces $f_1, \ldots, f_k$ as squares from left to right, and place each vertex of G_P so that it reaches the squares of all faces it belongs to. This uniquely determines the placement of all vertices except at f_1 and f_k (where the vertex of degree 2 could go on either row). We want u and v to be in the same row, which happens automatically unless u or v is the degree-2 vertex at f_1 or f_k. In the latter case, choose the row for the degree-2 vertex so that u and v are in the same row; otherwise place the degree-2 vertex arbitrarily. See also Fig. 3. After possible rotation, we may assume that u and v are now both in the top row.

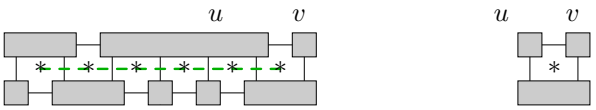


Fig. 3. How to draw a graph G_P whose dual tree is a path P. (Left) The graph whose dual tree is the main path of Fig. 1. (Right) A 1-vertex path P.

The proof of the lemma is now by induction on the pathwidth of T. In the base case $pw(T) = 0$. This means that tree T is a singleton vertex, and the drawing of Fig. 3 has height 2 and $\{u, v\}$ spans the top row; this proves all claims. For the inductive step, we first show (2), then (3), and then (1) since each uses the other.

Claim (2): Let $f_1, \ldots, f_k$ be a main path P to which (u, v) is adjacent, where f_1 and f_k are leaves of T. For each face f_i, $i = 1, \ldots, k$, let T_i be the subtree of $T - P$ whose root is adjacent to f_i. See also Fig. 4. Faces f_1 and f_k have no such subtree since they are leaves of T. Any f_i, $1 < i < k$ has at most one such subtree since it has two neighbours on P and $\deg(f_i) \leq 3$. By definition of a main path, T_i has treewidth at most $pw(T) - 1$.

Draw the graph G_P formed by the faces $f_1, \ldots, f_k$ as explained above; this places $\{u, v\}$ in the top row. Let G_i be the subgraph of H for which T_i is the dual tree. Thus, G_i is a maximal outer-planar subgraph whose dual tree has pathwidth at most $pw(T) - 1$. Let (u_i, u_i) be the edge that G_i shares with face f_i, with u_i clockwise before v_i on G_i. By induction, G_i has a drawing with $\{u_i, v_i\}$ spanning the top row with height $h_i \leq \max\{2, 4pw(T_i)\} \leq \max\{2, 4pw(T) - 4\}$.

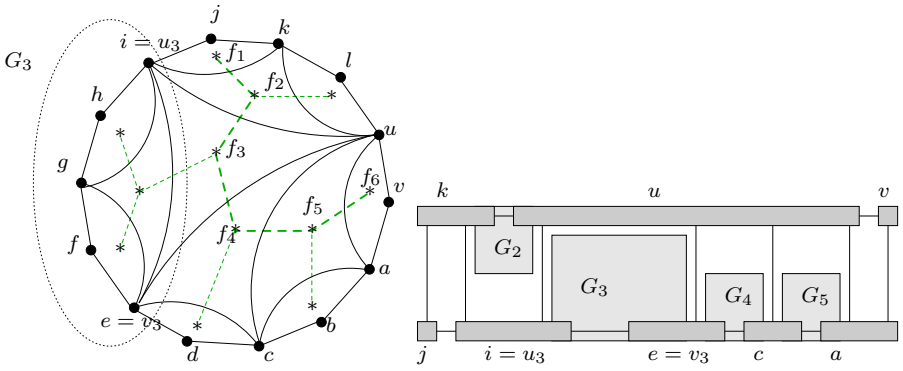


Fig. 4. Definition of G_i, and how to insert its drawing into the drawing of G_P

Now take the drawing of G_P and expand it vertically by adding $\max_i\{h_i\} - 1$ rows. For each $i = 2, \ldots, k$, if the drawing of G_i has W_i columns, then add W_i columns between the drawings of u_i and v_i in G_P. (Note that u_i and v_i are horizontally adjacent in G_P, since they are not incident to f_1 or f_k.)

Recall that G_i was drawn, using induction, with $\{u_i, v_i\}$ spanning the top row. Specifically, u_i is on the top left corner, since u_i comes before v_i in clockwise order around the outer-face of G_i. If $\{u_i, v_i\}$ are in the bottom row of G_P, then we flip the drawing of G_i vertically and insert it in the space that we made between u_i and v_i in G_P. If $\{u_i, v_i\}$ are in the top row of G_P, then to make u_i and v_i match up we first rotate the drawing of G_i by 180°, then flip it vertically, and then insert it into its space in G_P. See Fig. 4.

The drawing of G_i has height at most $\max\{2, 4pw(T) - 4\}$. Inserting G_i into G_P re-uses the row that contains u_i and v_i, so at most $\max\{1, 4pw(T) - 5\}$ rows are added to the two rows of the drawing of G_P. The final height hence is $\max\{3, 4pw(T) - 3\}$ which completes the proof of (2).

Claim (3): By (2), graph H can be drawn with u and v in the top row, with height $\max\{3, 4pw(T) - 3\}$. Now *release u and v* by adding a row and relocating them into it, so that $\{u, v\}$ spans the new top row. (This is quite similar in spirit to the modifications introduced in our earlier paper [1].)

More precisely, add a new row above the existing drawing. Move u and v into this new row, with u occupying everything from the top left corner to its rightmost column, and v occupying everything else in the top row. If x was a

neighbour of u, then it either was connected to u by a vertical line (which can simply be extended to continue to the new position of u), or it was the unique vertex to the left of u in the top row. (Here having flat visibility representation is crucial, so that any such neighbour is in the top row.) In the latter case, x can now add a vertical line towards the new position of u, since u spans the whole range above x. Similarly connect any neighbour of v to the new position of v.

Thus, releasing u and v adds one unit of height and achieves that $\{u, v\}$ spans the top row. The result then holds by (2).

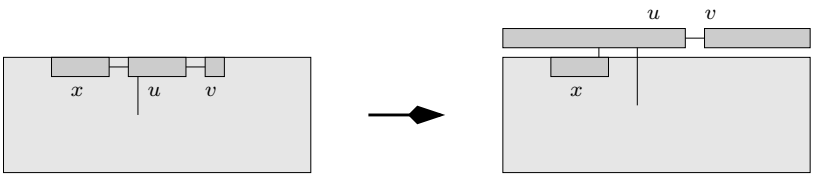


Fig. 5. Releasing u and v

Claim (1): Let $P' = f'_1, \ldots, f'_{k'}$ be a main path of T. Let f be the interior face that is adjacent to (u, v). In tree T, there is a unique path that connects f to a vertex $f'_{j'}$ that belongs to the main path. Note that $1 \neq j' \neq k'$, otherwise one could simply extend the main path to f and be in case (2). See also Fig. 6.

Let P be the path that consists of part of the main path $f'_j, f'_{j'+1}, f'_{j'+2}, \ldots, f'_{k'}$, the path from $f'_{j'}$ to f, and then continues from f until it reaches a leaf of T. Enumerate P as $f_1, \ldots, f_k$ with $f_j = f'_{j'}$ and $f_k = f'_{k'}$. The drawing now proceeds exactly as in case (2), i.e., define the subtree T_i of $T - P$ that is attached to f_i, and draw its corresponding graph G_i recursively. Draw the graph G_P induced by the faces $f_1, \ldots, f_k$ in two rows, and insert the drawings of $G_2, \ldots, G_{k-1}$ after adding sufficiently many rows and columns.

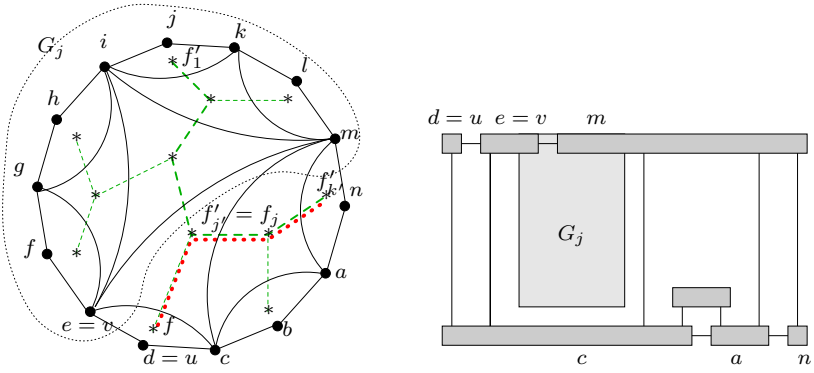


Fig. 6. The example from Fig. 1, but using (d, e) as edge (u, v). The path P used is marked dotted.

It remains to analyze the height. Recall that any subtree of $T - P'$ has pathwidth at most $pw(T) - 1$ since P' was a main path. For any $i \neq j$, subtree T_i is a subtree of $T - P'$ and hence has pathwidth at most $pw(T) - 1$. So for $j \neq i$, graph G_i is drawn with height at most $\max\{2, 4pw(T_i)\} \leq \max\{2, 4pw(T) - 4\}$.

A special case is T_j, which contains the rest of the main path, $f'_1, \ldots, f'_{j'-1}$ and hence may well have pathwidth $pw(T)$. If $pw(T_j) < pw(T)$, then we deal with T_j as with the other subtrees and draw it with height at most $\max\{2, 4pw(T) - 4\}$. So presume $pw(T_j) = pw(T)$. Then $f'_1, \ldots, f'_{j-1}$ is a main path of T_j, since all its subtrees have (by definition of main path of T) pathwidth at most $pw(T) - 1 = pw(T_j) - 1$. Observe that the edge (u_j, v_j) (i.e., the edge shared by G_j and G_P) is incident to $f'_{j'-1}$, and hence adjacent to this main path of T_j. Therefore T_j can be drawn using case (3) with height $\max\{4, 4pw(T) - 2\}$.

So all drawings of all subgraphs have height at most $\max\{4, 4pw(T) - 2\}$. Since merging these drawings into the 2-row drawing of G_P reuses one row, the height of the drawing with u and v in the top row is at most $\max\{5, 4pw(T) - 1\}$. Now release u and v as in case (3) and obtained a drawing with height $\max\{6, 4pw(T)\}$ where $\{u, v\}$ spans the top row. This proves (3) unless $pw(T) = 1$.

If $pw(T) = 1$, then T is a cater-pillar, i.e., it consists of a path with leaves attached. It is easy to draw H using three rows such that one of $\{u, v\}$ (say u) is in the top row and v (which has degree 2 since (u, v) is not adjacent to a main path) is in the middle row. Now relocate both u and v to a newly added row on the top and re-connect to their neighbours as illustrated in Fig. 7; this gives a drawing of height 4 as desired. □

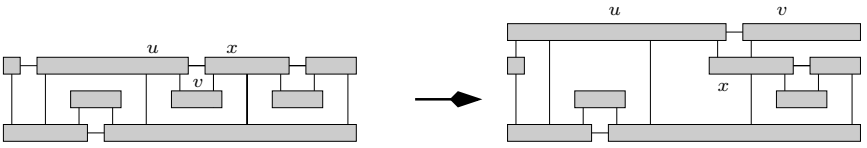


Fig. 7. Drawing of a graph where the dual tree is a caterpillar, and how to transform it into one where $\{u, v\}$ spans the top, even if v was not in the top row

The width of the drawings created with this algorithm are not especially small, but for completeness' sake we analyze it as well. For technical reasons, it will be helpful to re-define the term "leaf". If T is a tree with at least two vertices, then a *leaf* of T is a vertex of degree 1 in T. We use $\ell(T)$ to denote the number of leaves in T. If T is a tree with only one vertex, then we define $\ell(T) := 2$.

Lemma 2. *The flat visibility representation created with Lemma 1 has width at most* $|T| + \ell(T) - 1$.

Proof. If T is a singleton vertex, then its drawing has width 2 and $\ell(T) = 2$, which proves the claim.

Now assume that $pw(T) \geq 1$. Let $P = f_1, \ldots, f_k$ be the path of T for which we drew the corresponding graph G_P first. (P is a main path in case (2) and the combination of part of a main path, the path to f, and a path from f to a leaf in case (3)). Let T_i be the subtree of $T - P$ that is adjacent to f_i; we drew the subgraph G_i whose dual is T_i recursively and merged it into the drawing of G_P. Let $I \subseteq \{2, \ldots, k-1\}$ be all those indices i for which T_i is non-empty.

For any $i \in I$, by induction graph G_i is drawn with width at most $|T_i| + \ell(T_i) - 1$. The drawing of G_P is drawn with width $|P| + \ell(P) - 1$. Therefore the width of the combined drawing is at most

$$|P| + \ell(P) - 1 + \sum_{i \in I} (|T_i| + \ell(T_i)) - 1) = |T| + \ell(P) - 1 + \sum_{i \in I} (\ell(T_i) - 1).$$

It remains to count $\ell(T)$, which equals the number of degree-1 vertices in T since $pw(T) \geq 1$ and so T is not a singleton vertex. Any leaf in P is also a leaf in T since the path begins and ends at leaves of T. But not every leaf in T_i is a leaf in T: if f_i' is the face of T_i that shares an edge with f_i, then f_i' may be a leaf in T_i but not in T.

If f_i' had degree 1 in T_i, then f_i' added one $\ell(T_i)$, but adds nothing to $\ell(T)$. If f_i' had degree 0 in T_i, then f_i added two to $\ell(T_i)$, but in T it is a vertex of degree 1 and hence adds only one to $\ell(T)$. Hence, for each subtree T_i, there is one vertex that may have added one more unit to $\ell(T_i)$ than it adds to $\ell(T)$. Therefore

$$\ell(P) + \sum_{i \in I} \ell(T_i) \leq \ell(T) + |I|.$$

The result now follows after combining the two equations. □

Theorem 1. *Any 2-connected outer-planar graph G has a flat visibility representation of height $4pw(G) - 3$ and width $\lceil \frac{3}{2}(n-2) \rceil$. Moreover, such a representation can be found in linear time.*

Proof. Since G is a 2-connected outer-planar graph, one can add edges to it until it is a maximal outer-planar graph H such that the dual graphs G^* and H^* satisfy $pw(H^*) \leq pw(G^*) + 1$ [3, Lemma 5]. Also, $pw(G^*) \leq pw(G)$ [3, Lemma 4]. If T is the dual tree of H, then $pw(T) = pw(H^*) - 1$ [3, Lemma 3]. Putting it all together, one can add edges to G to obtain a maximal outer-planar graph H such that the dual tree T of H satisfies $pw(T) = pw(H^*) - 1 \leq pw(G^*) + 1 - 1 \leq pw(G)$.

Find a main path of T and let (u, v) be an edge on the outer-face of H adjacent to the main path. Then draw H with height $\max\{3, 4pw(T) - 3\}$ using Lemma 1(2). Observe that $\max\{3, 4pw(T) - 3\} \leq 4pw(G) - 3$ since $pw(T) \leq pw(G)$ and $pw(G) \geq 2$ since G is 2-connected. This proves the height-bound. The width is at most $|T| + \ell(T) - 1$ by Lemma 2. Since for a maximal outer-planar graph the dual tree T has maximum degree 3, it has at most $\lceil |T|/2 + 1 \rceil$ leaves. So the width is at most $\frac{3}{2}|T| = \frac{3}{2}(n-2)$ since H has $n-2$ interior faces.

Finding the completion to a maximal outer-planar graph can be done in linear time [3]. Given this, we can compute T in linear time. The pathwidth of T can be computed in linear time [19]. But we also need to find main paths, both of T

and of all the subtree of T encountered by our algorithm. Suderman [21] showed in detail how to do this for the subtrees needed for his drawing algorithm; our subtrees are defined similarly and the same techniques can be applied.

As for the actual drawing algorithm, this can easily be implemented in linear time if we store rows and columns as abstract objects in a sorted list. Then merging the drawing of a subgraph takes constant time by inserting its list of rows/columns into the overall lists. Once all recursions are finished, compute the final coordinates of graph G by traversing these lists and assigning integers in order, and hence obtain the drawing in linear time. □

Since the pathwidth is a lower bound on the height [8], this gives:

Corollary 1. *There exists a linear-time 4-approximation for the height of visibility representations of 2-connected outer-planar graphs.*

A few comments on this result:

- The drawings do not preserve the planar embedding, because the drawing of the subgraph G_i is flipped vertically. Similarly as in [1], one can modify the algorithm to create orthogonal box-drawings that do preserve the embedding and have height $O(pw(T))$, by routing edge (u_i, v_i) "around" the drawing of G_i.
- The algorithm works for any outer-planar graph, and 2-connectivity is used only when adding edges to obtain a maximal outer-planar graph H of pathwidth $\approx pw(G)$. If such a result were known for all outer-planar graphs, the 4-approximation would hold for all outer-planar graphs. But this remains an open problem.
- Any tree has pathwidth at most $\log_3(2n+1)$ [19], thus the height-bound is $O(\log n)$. So in the worst-case the construction here has asymptotically the same height as the one in [1], but it is better for graphs with small pathwidth.

Our algorithm created flat visibility representations. The main motivation for using this drawing model was that it made the algorithm simpler: One can easily insert extra space for subgraphs, and since boxes have height 1, we can apply the operation to release vertices.

A more common graph drawing style represents vertices as points, and edges as straight lines, or (in *poly-line drawings*) as contiguous sequences of straight lines. As shown in [1], the visibility representation can be converted to a poly-line drawing of asymptotically the same width and height, which implies:

Corollary 2. *Any 2-connected outer-planar graph G has a poly-line drawing of height $O(pw(G))$ and width $O(n)$.*

4 Straight-Line Drawings

We now show that the drawings can also be easily transformed into straight-line drawings of the same height, if we allow increasing the width.

Theorem 2. *Let Γ be a flat visibility representation of a graph G that has height h. Then there exists a straight-line drawing Γ' of G of height h.*

Proof. For any vertex v, use $x_l(v), x_r(v)$ and $y(v)$ to denote leftmost and rightmost x-coordinate and (unique) y-coordinate of the box that represents v in Γ. Use $X(v)$ and $Y(v)$ to denote the (initially unknown) coordinates of v in Γ'. For any vertex set $Y(v) = y(v)$; this proves the height-bound.

Let $v_1, \ldots, v_n$ be the vertices sorted by $x_l(.)$, breaking ties arbitrarily. The algorithm determines $X(.)$ for each vertex by processing vertices in this order and expanding the drawing Γ'_{i-1} created for $v_1, \ldots, v_{i-1}$ into a drawing Γ'_i of $v_1, \ldots, v_i$. Throughout, it will hold that $Y(v) = y(v)$ for all vertices, and for any row, the left-to-right order of vertices will be the same in Γ' (as far as it has been built yet) as it was in Γ.

Suppose $X(v_g)$ has been computed for all $g < i$ already. To find $X(v_i)$, determine lower bounds for it by considering all predecessors of v_i and taking the maximum over all of them. (For each vertex v_i, the *predecessors* of v_i are the neighbours of v_i that come earlier in the order $v_1, \ldots, v_n$.) A first (trivial) lower bound for $X(v_i)$ is that it needs to be to the right of anything in row $y(v_i)$. Thus, if Γ'_{i-1} contains a vertex or part of an edge at point $(X, y(v_i))$, then $X(v_i) \geq \lfloor X \rfloor + 1$ is required.

Next consider any predecessor v_g of v_i with $y(v_g) \neq y(v_i)$. Since v_g and v_i are not in the same row, they must see each other vertically in Γ, which means that $x_r(v_g) \geq x_l(v_i)$. See also Fig. 8. So if v_g has a neighbour v_k to its right in Γ, then $x_\ell(v_k) > x_r(v_g) \geq x_\ell(v_i)$, which implies that $k > i$, so v_k has not been added to Γ'_{i-1}. Since the order of the vertices in each row is unchanged, therefore v_g is the rightmost vertex in its row in Γ'_{i-1} and can see towards infinity on the right. But then v_g can also see the point $(+\infty, y(v_i))$, or in other words, there exists some X_g such that v_g can see all points $(X, y(v_i))$ for $X \geq X_g$. Impose the lower bound $X(v_i) \geq \lfloor X_g \rfloor + 1$ on the x-coordinate of v_i.

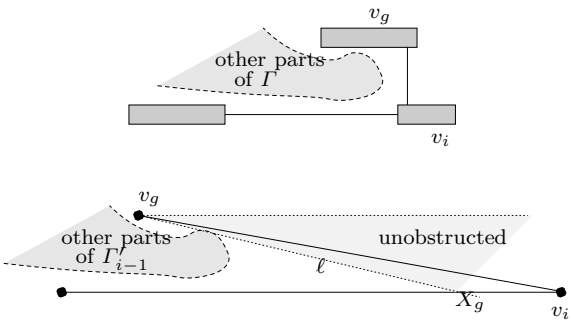


Fig. 8. Transforming a flat visibility drawing into a straight-line drawing with unchanged y-coordinates

Now let $X(v_i)$ be the smallest value that satisfies the above lower bounds (from the row $y(v_i)$ and from all predecessors of v_i in different rows.) Set $X(v_i) = 0$ if there were no such lower bounds. Directly by construction, placing v_i at $(X(v_i), y(v_i))$ allows it to be connected with straight-line segments to all its predecessors. This includes the predecessor (if any) that is in the row $y(v_i)$, since it can be horizontally connected to v_i. (Here having a flat visibility representation is crucial, because there is only one such predecessor and it is placed in the same row.) This gives a drawing Γ'_i of $v_1, \ldots, v_i$ as desired, and the result follows by induction. □

Corollary 3. *Any outer-planar graph has a straight-line drawing of height $O(\log n)$, and any 2-connected outer-planar graph G has a straight-line drawing of height $O(pw(G))$.*

Unfortunately, while our transformation keeps the height intact, the width can increase dramatically. It is not hard to construct a flat visibility representations of height h and width $O(n)$ for which the resulting straight-line drawing has width $\Omega((h-2)^{(n-3)})$; see Fig. 9. But this is (asymptotically) the worst that can happen. With some calculations (left to the reader), one can show the following bound by induction on i:

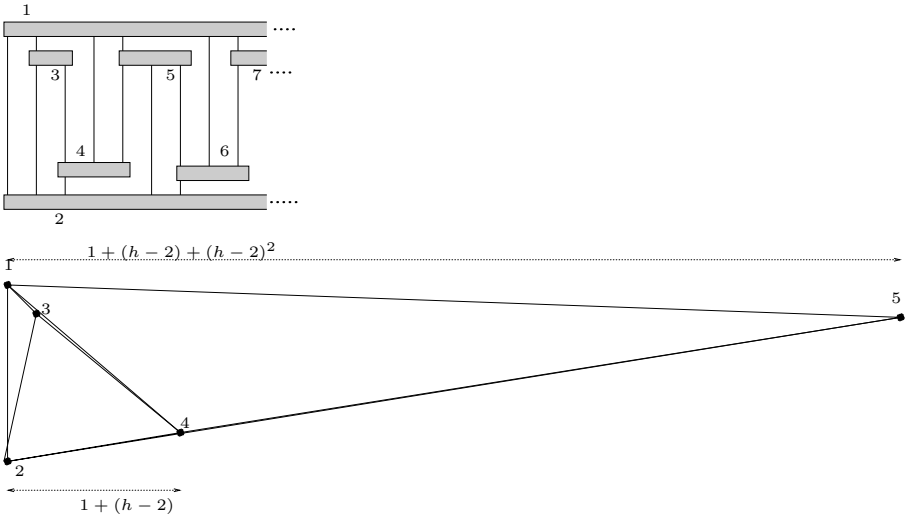


Fig. 9. A flat visibility representation for which the corresponding straight-line drawing has exponential width. Vertices are numbered in the order in which they are processed. Vertex i is placed with x-coordinate $1 + (h-2) + \ldots + (h-2)^{i-3}$ for $i \geq 3$, and leaves an edge with slope $\pm 1/(1 + (h-2) + \ldots + (h-2)^{i-3})$.

Lemma 3. *For $h \geq 3$, all x-coordinates are in $O((h-2)^n)$. More precisely, if vertex v_i is not on the first or last row, then $X(v_i) \leq 1 + (h-2) + (h-2)^2 + \ldots + (h-2)^{i-2}$.*

It remains open whether some other construction could create straight-line drawings with smaller width, perhaps by rearranging which vertex is in which row, or at the expense of some height.

5 Conclusion and Open Problems

This paper presented an algorithm to draw 2-connected outer-planar graphs, with the objective of keeping the height as small as possible. While the pathwidth $pw(G)$ of such a graph G is an easy lower bound, the algorithm created drawings of height $4pw(G) - 3$ and hence is a 4-approximation algorithm for the height.

The paper leaves many open problems:

- Is our algorithm also an $O(1)$-approximation for outer-planar graphs that are not 2-connected? In particular, can we add edges to any outer-planar graph G such that the resulting graph H is 2-connected and outer-planar and has pathwidth $O(pw(G))$? To our surprise, no such result appears to be known.
- The bound for the width of the visibility representations was $O(n)$. Can this be reduced, at least if the maximum degree is small? Can we create drawings where the width is also within a constant factor of the optimum?
- Is it possible to find drawings of optimal height for outer-planar graphs in polynomial time? Perhaps even of optimal area? (The NP-hardness reduction for the area [16] uses 2-outer-planar graphs.)
- Are there straight-line drawings of outer-planar graphs that have height $O(pw(G))$ and polynomially bounded width?
- Is there an $O(1)$-approximation algorithm for the height of series-parallel graphs, or other generalizations of outer-planar graphs?
- What are other approximation algorithms in graph drawing? Is is possible to find drawings for which the area is an $O(1)$-approximation?

References

1. Biedl, T.: Drawing outer-planar graphs in $O(n \log n)$ area. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 54–65. Springer, Heidelberg (2002), Full-length version appeared in [2]
2. Biedl, T.: Small drawings of outerplanar graphs, series-parallel graphs, and other planar graphs. Discrete and Computational Geometry 45(1), 141–160 (2011)
3. Bodlaender, H.L., Fomin, F.V.: Approximation of pathwidth of outerplanar graphs. Journal of Algorithms 43(2), 190–200 (2002)
4. Brandenburg, F.-J.: Drawing planar graphs on (8/9)*n^2 area. Electronic Notes in Discrete Mathematics 31, 37–40 (2008)
5. Chrobak, M., Nakano, S.-I.: Minimum-width grid drawings of plane graphs. Comput. Geom. 11(1), 29–54 (1998)
6. Di Battista, G., Frati, F.: Small area drawings of outerplanar graphs. Algorithmica 54(1), 25–53 (2009)

7. Dujmovic, V., Fellows, M., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F., Whitesides, S., Wood, D.: On the parameterized complexity of layered graph drawing. Algorithmica 52, 267–292 (2008)
8. Felsner, S., Liotta, G., Wismath, S.: Straight-line drawings on restricted integer grids in two and three dimensions. Journal of Graph Algorithms and Applications 7(4), 335–362 (2003)
9. Formann, M., Wagner, F.: The VLSI layout problem in various embedding models. In: Möhring, R.H. (ed.) WG 1990. LNCS, vol. 484, pp. 130–139. Springer, Heidelberg (1991)
10. Frati, F.: Straight-line drawings of outerplanar graphs in $O(dnlogn)$ area. In: Canadian Conference on Computational Geometry (CCCG 2007), pp. 225–228 (2007)
11. Frati, F.: Lower bounds on the area requirements of series-parallel graphs. Discrete Mathematics & Theoretical Computer Science 12(5), 139–174 (2010)
12. de Fraysseix, H., Pach, J., Pollack, R.: Small sets supporting Fary embeddings of planar graphs. In: ACM Symposium on Theory of Computing (STOC 1988), pp. 426–433 (1988)
13. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10, 41–51 (1990)
14. Garg, A., Rusu, A.: Area-efficient planar straight-line drawings of outerplanar graphs. Discrete Applied Mathematics 155(9), 1116–1140 (2007)
15. Heath, L.S., Rosenberg, A.L.: Laying out graphs using queues. SIAM Journal on Computing 21(5), 927–958 (1992)
16. Krug, M., Wagner, D.: Minimizing the area for planar straight-line grid drawings. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 207–212. Springer, Heidelberg (2008)
17. Mondal, D., Alam, M.J., Rahman, M. S.: Minimum-layer drawings of trees. In: Katoh, N., Kumar, A. (eds.) WALCOM 2011. LNCS, vol. 6552, pp. 221–232. Springer, Heidelberg (2011)
18. Mondal, D., Nishat, R.I., Rahman, M.S., Alam, M.J.: Minimum-area drawings of plane 3-trees. J. Graph Algorithms Appl. 15(2), 177–204 (2011)
19. Scheffler, P.: A linear-time algorithm for the pathwidth of trees. In: Bodendieck, R., Henn, R. (eds.) Topics in Combinatorics and Graph Theory, pp. 613–620. Physica-Verlag, Heidelberg (1990)
20. Schnyder, W.: Embedding planar graphs on the grid. In: ACM-SIAM Symposium on Discrete Algorithms (SODA 1990), pp. 138–148 (1990)
21. Suderman, M.: Pathwidth and layered drawings of trees. International Journal of Computational Geometry and Applications 14(3), 203–225 (2004)
22. Tayu, S., Nomura, K., Ueno, S.: On the two-dimensional orthogonal drawing of series-parallel graphs. Discrete Applied Mathematics 157(8), 1885–1895 (2009)

Approximation Algorithms for the Wafer to Wafer Integration Problem*

Trivikram Dokka[1], Marin Bougeret[2], Vincent Boudet[2], Rodolphe Giroudeau[2], and Frits C.R. Spieksma[1]

[1] ORSTAT, K.U.Leuven, Naamsestraat 69, B-3000 Leuven, Belgium
{trivikram.dokka,frits.spieksma}@econ.kuleuven.be
[2] LIRMM Montpellier, France

Abstract. Motivated by the yield optimization problem in semiconductor manufacturing, we model the wafer to wafer integration problem as a special multi-dimensional assignment problem (called WWI-m), and study it from an approximation point of view. We give approximation algorithms achieving an approximation factor of $\frac{3}{2}$ and $\frac{4}{3}$ for WWI-3, and we show that extensions of these algorithms to the case of arbitrary m do not give constant factor approximations. We argue that a special case of the yield optimization problem can be solved in polynomial time.

Keywords: wafer-to-wafer integration, approximation, computational complexity, efficient algorithm.

1 Introduction

Consider the following problem. Given are m sets V_i, $i = 1, ..., m$. Each set contains n p-dimensional vectors; each entry of each vector is a nonnegative integer. We define the *cost* of vector $u = (u_1, u_2, \ldots, u_p)$ as follows: $c(u) = \sum_{i=1}^{p} u_i$. Given a pair of vectors u, v, we can construct the vector $u \vee v$ by defining the operation $\vee$ as follows:

$$u \vee v = (\max(u_1, v_1), \max(u_2, v_2), \ldots, \max(u_p, v_p)).$$

Notice that $(u \vee v) \vee w = u \vee (v \vee w)$.

Consider now an m-tuple, ie, a set of m vectors $u^1, u^2, \ldots, u^m \in V_1 \times V_2 \times \ldots \times V_m$. The cost of an m-tuple equals $c(u^1 \vee u^2 \vee \ldots \vee u^m)$. Our problem, that we denote by WWI (see Section 1.1), is to find n disjoint m-tuples such that each vector is used exactly once, while total cost is minimum. In the figure below an instance with $m = 3, n = p = 2$ is depicted; notice that this instance has the property that each vector is a 0-1 vector; the value of an optimal solution to this instance equals 2 (this is achieved by joining the first vector of V_1, the second vector of V_2, and the first vector of V_3 into $(1, 0)$; the remaining three vectors form $(0, 1)$).

We were motivated to look at this optimization problem by an application in the semi-conductor industry, that we now proceed to describe.

* This research was supported by OT Grant OT/07/015.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 286–297, 2013.

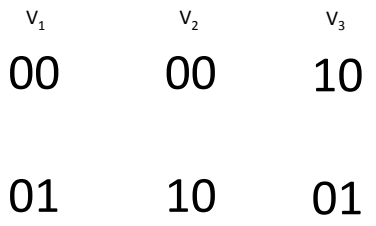


Fig. 1. WWI instance; $m = 3, n = p = 2$

1.1 The Application

Our understanding of the semi-conductor industry, and in particular the wafer-to-wafer production process is primarily based upon Reda et al. [6], Verbree et al. [9], Taouil and Hamdioui [8]. In the semi-conductor industry, Through Silicon Vias (TSV) based three-Dimensional Stacked Integrated Circuits (3D-SIC) is an emerging technology that provides large benefits: a smaller footprint, a higher interconnect density between stacked dies, higher performance, and lower power consumption due to shorter wires when compared to planar IC's. One of the key steps in the production of 3D-SIC's is stacking. There are three different ways of stacking: (1) wafer to wafer (2) die to wafer (3) die to die (see [6]). Of these three approaches, wafer to wafer stacking offers the highest manufacturing throughput coupled with other advantages. However, wafer to wafer stacking approach suffers from a drawback that it may have a low yield. The main motivation of this paper is to study this yield optimization problem in the wafer to wafer integration process.

The yield optimization problem in the semi-conductor industry can be informally described as follows: there are m lots of wafers called wafer lots, with each wafer lot consisting of n wafers. A wafer consists of a string of bad dies and good dies; in our context this translates to a '0' in case of a good die, and a '1' in case of a bad die (such a string corresponds to a vector in the description of WWI). The objective is to form n stacks (a stack corresponds to an m-tuple) by integrating one wafer from each lot (a set V_i) while optimizing the yield i.e., minimizing the total number of bad dies in the resulting stacks (which is equivalent to maximizing the total number of good dies in the resulting stacks). Integrating two wafers can be seen as superimposing the two corresponding strings; in this operation the position in the merged string is only 'good' when the two corresponding entries are good, otherwise it is 'bad'. Due to this reason we call the above problem the wafer to wafer integration (WWI) problem. We refer to it as WWI-m, where m is the number of wafer-lots.

Notice that the yield optimization problem described here is a special case of WWI, since instances of the yield optimization problem have 0-1 vectors (instead of vectors with arbitrary integral entries).

Dimensions of typical instances occurring in the semi-conductor industry have values of m, n, and p ranging as follows: $3 \leq m \leq 10, 25 \leq n \leq 75, 500 \leq p \leq 1000$ (see [6][9]).

1.2 Goal and Related Work

Our main intention in this paper is to formulate the WWI-m as a combinatorial optimization problem and study it from an approximation point of view. Usually, the yield optimization problem is formulated as a maximization problem, however, we feel that studying the minimization problem is especially relevant from approximation point of view. Indeed, owing to the fact that in the yield optimization instances, the number of bad dies in each wafer is typically much less than the number of good dies, it make sense to be able to approximate the (smaller) minimization optimum instead of the (larger) maximization optimum.

There is increasing attention for the yield optimization in the literature. One example is the contribution [6]. In [6] the problem is formulated as an multi-index assignment problem; further, computational performance with straightforward heuristics is reported. Some recent work on this problem is also reported in [8] [9]. As we will show, WWI can be seen as a multi-index assignment problem where the costs have a certain structure. Three dimensional assignment problems with so-called decomposable costs have been studied in Crama and Spieksma [4] and Burkard et al. [3]. Multi-dimensional assignment extensions of this cost structure appear in Bandelt et al. [1]. An survey on multi-dimensional assignment problems can be found in [7] and chapter 10 of [2].

1.3 Our Results

Our results can be summarized as follows:

- We present an IP-formulation that is an alternative to the traditional formulation given in [6]. This alternative formulation contains fewer variables, and may be more suited from a computational perspective (see Section 2).
- We prove that the yield optimization problem is NP-hard (see Section 3).
- We give two simple approximation algorithms for WWI-3, one with a $\frac{3}{2}$ performance guarantee, and one with a $\frac{4}{3}$ performance guarantee (see Section 4.1). We also show that natural extensions of these algorithms to the case of arbitrary m fail to provide a constant-factor guarantee (see Section 4.2).
- We show that, in case of a fixed m and a fixed p, the yield optimization problem is solvable in polynomial time (see Section 5).

2 Problem Formulation

In subsection 2.1 we give a straightforward formulation of the yield optimization problem in wafer to wafer integration as a m-dimensional axial assignment problem, see also [6]. Section 2.2 presents an alternative IP-formulation that may be more suited from a computational perspective.

2.1 IP Formulation

We set $K = V_1 \times V_2 \times \ldots \times V_m$, ie, K corresponds to the set of m-tuples. Next, for each $a \in K$, there is a binary variable x_a indicating whether m-tuple a is selected ($x_a = 1$) or not ($x_a = 0$). The formulation is now as follows (see also [6])

$$\min \quad \sum_{a \in K} c(a) \cdot x_a \tag{1}$$

$$\sum_{a:\ u \in a} x_a = 1 \ \text{ for each } \ u \in \cup_{i=1}^{m} V_i, \tag{2}$$

$$x_a \in \{0,1\} \ \text{ for each } a \in K. \tag{3}$$

Observe that constraints (2) ensure that each vector u is in an m-tuple.

2.2 Alternative IP Formulation

In this section we give an IP formulation that is different from the classical formulation and contains fewer variables.

In this formulation, we model the problem by treating V_1 as the *hub*. Each vector in $\cup_{i=2}^{m} V_i$ is assigned to a vector in V_1; this decision is modelled by a binary variable as follows. There is a variable $z_{u,v}$, where $u \in V_1$ and $v \in \cup_{i=2}^{m} V_i$, such that:

$$\begin{aligned} z_{u,v} &= 1 \text{ if vectors } u \text{ and } v \text{ are contained in the same } m\text{-tuple,} \\ &= 0 \text{ otherwise.} \end{aligned}$$

In addition, we introduce variables $y_{u,\ell}$ as follows.

$$y_{u,\ell} = \text{the value in the } \ell\text{-th position of the } m\text{-tuple containing vector } u \in V_1.$$

$$\min \sum_{u \in V_1} \sum_{\ell=1}^{p} y_{u,\ell} \tag{4}$$

$$\sum_{u \in V_1} z_{u,v} = 1 \quad \text{for each } \ v \in \cup_{i=2}^{m} V_i, \tag{5}$$

$$\sum_{v \in V_i} z_{u,v} = 1 \quad \text{for each } \ u \in V_1, \text{ for each } i = 2, \ldots, m, \tag{6}$$

$$y_{u,\ell} \geq \max(u_\ell, v_\ell) \cdot z_{u,v} \quad \text{for each } u \in V_1, \text{for each } v \in \cup_{i=2}^{m} V_i, 1 \leq \ell \leq p, \tag{7}$$

$$z_{u,v} \in \{0,1\} \quad \text{for each } u \in V_1, \text{ for each } v \in \cup_{i=2}^{m} V_i. \tag{8}$$

Here, $u_\ell(v_\ell)$ denotes the ℓ^{th} entry in the vector $u(v)$. Observe that this alternative formulation has very few variables ($O(mn^2 + np)$) when compared to the number of variables in classical assignment formulation ($O(n^m)$). Even for reasonably small instances it will be difficult to solve the resulting problem with IP solvers using the classical formulation, whereas we might be able to solve them using (4)-(8).

3 The Complexity of WWI

In this section we describe a reduction from MAX-3DM to WWI. Recall that for a given pairwise disjoint sets X,Y,Z, and a set of ordered triples $T \subseteq X \times Y \times Z$, a *matching* in T is a subset of $M \subseteq T$ in which no two ordered triples in M agree in any coordinate. The goal of the MAXIMUM 3-DMENSIONAL MATCHING problem (shortly, MAX-3DM) is to find a matching in T of maximum cardinality.

Kann [5] showed that 3-bounded MAX-3DM is APX-complete.

Reduction. Consider an arbitrary instance I of MAX-3DM with three sets $X = \{x_1, ..., x_q\}$, $Y = \{y_1, ..., y_q\}$, and $Z = \{z_1, ..., z_q\}$, and a subset $T \subseteq X \times Y \times Z$. Let the number of triples be denoted by $|T|$.

Starting from the instance I of MAX-3DM, we now build a corresponding instance I' of WWI-3 by specifying V_i $(i = 1, 2, 3)$, as follows:

- for each element in $x_i \in X$ there is a vector $v_{1i} \in V_1$
- for each element in $y_j \in Y$ there is a vector $v_{2j} \in V_2$
- for each element in $z_k \in Z$ there is a vector $v_{3k} \in V_3$
- each vector has length $|T|$ i.e., $p = |T|$; in fact, for each triple $e = (x_i, y_j, z_k) \in T$, there is a position in each vector corresponding to that triple. The three vectors v_{1i}, v_{2j}, and v_{3k} corresponding to triple (x_i, y_j, z_k), have a '0' in that position, all other vectors have a '1' in that position.

This completes the description of WWI-3 instance.

It is easy to see that a solution to an instance of MAX-3DM with value k corresponds to a solution to the corresponding instance of WWI-3 with cost $pq - k$. Thus we can state the following theorem:

Theorem 1. *WWI-3 is NP-hard, even for the special case of* $0-1$ *vectors (i.e., yield optimization).*

Notice that, when the yield optimization problem would have been formulated as a maximization problem, straightforward reductions from MAX-kDM imply that a constant factor approximation for the maximization version of WWI-m would imply P=NP.

4 Approximation Algorithms for WWI-m

In this section we first prove that a straightforward algorithm (called heuristic H) for WWI-3 is a $\frac{3}{2}$ approximation algorithm. We show how a simple modification of this heuristic allows us to improve the worst-case ratio to $\frac{4}{3}$. Finally, we show that a natural extension of heuristic H to WWI-m can perform arbitrarily bad.

4.1 The Case $m = 3$

Algorithm 1. Heuristic H

1. Solve an assignment problem between V_1 and V_2, based on costs $c(u \vee v)$, $u \in V_1$,$v \in V_2$. Call the resulting matching M.
2. Solve an assignment problem between M and V_3 based on costs $c((u \vee v) \vee w)$, $u \vee v \in M$, $w \in V_3$.

Theorem 2. *Heuristic H is a $\frac{3}{2}$-approximation algorithm for WWI-3. This bound is tight.*

Proof. We first introduce some notation. Let OPT denote the value of an optimal solution, and let $cost(H)$ refer to the value of the solution found by H. Let $c(V_i)$ equal total cost of the vectors in V_i, ie, $c(V_i) = \sum_{u \in V_i} c(u)$, for $i = 1, 2, 3$. Let c_{12}^{OPT} denote the value of a partial optimal solution restricted to $V_1 \times V_2$, ie, when we remove from the optimal solution the vectors from V_3; the total weight that remains equals c_{12}^{OPT}. Recall that M refers to matching found by H in the first step, and let c_{12}^{H} be the value of the partial solution obtained after Step 1 of the heuristic H.

Let us call x (y) the amount with which the value of a partial optimal (heuristic) solution increases when vectors from V_3 are matched optimally to the optimal (heuristic) pairs from $V_1 \times V_2$. Thus, by definition:

$$x = OPT - c_{12}^{OPT} \tag{9}$$

Clearly, the following inequality is valid:

$$c(V_3) \leq OPT \tag{10}$$

(9) and (10) imply

$$c(V_3) - x \leq c_{12}^{OPT}. \tag{11}$$

Consider a set U consisting of n p-dimensional vectors with total cost $c(U) = \sum_{u \in U} c(u)$. In addition, consider a set V, also consisting of n p-dimensional vectors. Let us now assign the vectors from V to the vectors of U using as a cost $c(u \vee v)$ for each $(u, v) \in U \times V$. Let the value of the resulting optimal solution be denoted by $c(U \times V)$. We say that an amount equal to $c(V) - (c(U \times V) - c(U))$ from V is *covered* by U (or equivalently, we say that U is *able to cover an amount of* $c(V) + c(U) - c(U \times V)$ from V).

Consider now the partial heuristic solution found after Step 1, ie, consider M.

Lemma 1. *There exists a feasible assignment of the vectors in V_3 to the pairs from M such that at least the amount $\frac{1}{2}(c(V_3) - x)$ from V_3 is covered by M.*

Argument: To argue that the lemma is true, consider the partial optimal solution restricted to $V_1 \times V_2$. Apparently, these n vectors are able to cover an amount of $c(V_3) - x$ from V_3 when assigning the strings from V_3 to these vectors (since

$OPT = c_{12}^{OPT} + x$). However, each vector in $V_1 \times V_2$ consists of p numbers, each one arising from either V_1 or V_2. Thus, we can partition the amount covered $c(V_3) - x$ into two disjoint parts: one part covered by numbers from vectors in V_1, one part covered by numbers from vectors in V_2. It follows that if one considers the following two assignments: one where the vectors from V_3 are assigned to the vectors from V_1 as in the optimal solution, and one where the vectors from V_3 are assigned to the vectors from V_2 as in the optimal solution, that at least one of these solutions will cover $\frac{1}{2}(c(V_3) - x)$. This proves the lemma.

We can now derive

$$\begin{aligned}
\text{cost}(H) &= c_{12}^{H} + y \\
&\leq c_{12}^{H} + c(V_3) - (\frac{1}{2}c(V_3) - \frac{1}{2}x) \\
&= c_{12}^{H} + \frac{1}{2}c(V_3) + \frac{1}{2}x \\
&\leq c_{12}^{OPT} + \frac{1}{2}c(V_3) + \frac{1}{2}x \\
&\leq c_{12}^{OPT} + \frac{1}{2}[c_{12}^{OPT} + x] + \frac{1}{2}x \\
&\leq \frac{3}{2}c_{12}^{OPT} + \frac{3}{2}x = \frac{3}{2}OPT.
\end{aligned}$$

The first inequality follows from Lemma 1, the second inequality follows from the fact that the heuristic, in Step 1, computes an optimum assignment between sets V_1 and V_2 whose costs cannot exceed c_{12}^{OPT}, the third inequality follows from (11) and the final inequality follows from the definition of x. Tightness follows from the instance depicted in Figure 1: observe that, for this instance, OPT = 2, whereas heuristic H might find a solution with value 3. □

A minor modification of heuristic H (denoted by H_{heavy}) allows us to improve the worst-case ratio without actually increasing the computational effort. Indeed, let us slightly modify H by ensuring that in Step 1 the *heaviest* set V_i is present, ie, we ensure that the set V_i for which $c(V_i)$ is maximal, is assigned to some $V_j, j \neq i$ in the first step.

Theorem 3. *Heuristic H_{heavy} is a $\frac{4}{3}$-approximation algorithm for WWI-3. This bound is tight.*

Algorithm 2. Heuristic H_{heavy}

0. Let $j = \arg\max_{i=1,2,3} c(V_i)$.
1. Solve an assignment problem between V_j and some $V_i, i \neq j$, based on costs $c(u \vee v)$, $u \in V_j, v \in V_i$. Call the resulting matching M.
2. Solve an assignment problem between M and the remaining set $V_k, k \neq j, k \neq i$ based on costs $c((u \vee v) \vee w)$, $u \vee v \in M$, $w \in V_k$.

Proof. Let us assume, without loss of generality, that set V_1 is the heaviest set. Thus, we have $c(V_1) \geq c(V_2)$ as well as $c(V_1) \geq c(V_3)$. Even more, let us assume (again wlog) that in Step 1 of H_{heavy} sets V_1 and V_2 are assigned to each other. We distinguish three cases.

Case 1: $0 \leq c(V_1) \leq \frac{1}{3}\text{OPT}$.
This case is trivial since any feasible solution is in fact optimal: $\text{cost}(H_{heavy}) \leq c(V_1) + c(V_2) + c(V_3) \leq 3 \cdot \frac{1}{3}\text{OPT} = \text{OPT}$.

Case 2: $\frac{1}{3}\text{OPT} < c(V_1) \leq \frac{2}{3}\text{OPT}$.
This case is similar to the analysis in Theorem 2. We derive:

$$\begin{aligned}\text{cost}(H_{heavy}) &= c_{12}^{H_{heavy}} + y \\ &\leq c_{12}^{OPT} + c(V_3) - (\frac{1}{2}c(V_3) + \frac{1}{2}x) \\ &= c_{12}^{OPT} + \frac{1}{2}c(V_3) + \frac{1}{2}x \\ &\leq \text{OPT} + \frac{1}{2}c(V_3) \leq \frac{4}{3}OPT.\end{aligned}$$

The last inequality follows from the assumption in this particular case, and the fact that $c(V_3) \leq c(V_1)$.

Case 2: $\frac{2}{3}\text{OPT} < c(V_1) \leq \text{OPT}$.
We denote by Q the weight from V_3 that is covered by V_1 when we solve an assignment problem between V_1 and V_3. The following is true:

$$c(V_1) + c(V_3) - Q \leq \text{OPT}. \tag{12}$$

We now derive:

$$\begin{aligned}\text{cost}(H_{heavy}) = c_{12}^{H_{heavy}} + y \ &\leq c_{12}^{H_{heavy}} + c(V_3) - Q \\ &\leq c_{12}^{OPT} + c(V_3) - Q \\ &\leq c_{12}^{OPT} + \text{OPT} - c(V_1) \\ &\leq \text{OPT} + \frac{1}{3}\text{OPT} = \frac{4}{3}OPT.\end{aligned}$$

The first inequality follows from Step 2 of H_{heavy} (the weight added to $c_{12}^{H_{heavy}}$ will not exceed the 'uncovered' weight from V_3 when we assign the vectors from V_3 to V_1), the second from Step 1 of H_{heavy}, the third inequality follows from (12), and the last inequality follows from the assumption in this particular case.

Tightness follows from the instance depicted in Figure 2: observe that, for this instance, OPT = 6, whereas heuristic H_{heavy} might find a solution with value 8. □

V_1	V_2	V_3
100000	000010	001000
010000	000001	000100
001000	100000	000010
000100	010000	000001
000000	000000	000000
000000	000000	000000

Fig. 2. H_{heavy}: an instance where OPT = 6 and cost(H_{heavy}) = 8

An obvious improvement to heuristic H and H_{heavy} would consist of a heuristic that runs H for all possible pairs in the first step, add the remaining set in the last step, and then choosing the best of the three feasible solutions found. Interestingly, this heuristic (which involves solving 6 assignment problems) does not have a lower worst case ratio than H_{heavy} (which only solves two assignment problems). This also follows from the example depicted in Figure 2.

Notice that heuristic H, in contrast to H_{heavy} can be seen as an online algorithm for a natural, online variant of WWI-3. Indeed, consider the setting where the sets V_1, V_2, and V_3 arrive sequentially over time, and that, before the arrival of a next set, the just arrived set V_i must be assigned to the partial tuples. Results given above imply directly:

Corollary 1. *Heuristic H is a $\frac{3}{2}$ competitive algorithm.*

Clearly, in this framework, H_{heavy} is not an online algorithm.

4.2 The Case of Arbitrary m

A natural extension of heuristic H to the case of arbitrary m is as follows. We iteratively assign set V_i to the existing partial tuples from $V_1 \times V_2 \times \ldots \times V_{i-1}$. Let us call the resulting heuristic H_{seq}. The performance of H_{seq} can be arbitrarily bad as can be seen from the description of the following instances. To understand these instances, it can be helpful to see each vector as a circle with p positions; in such a circle, the 1s, as well as the 0s, will appear consecutively. Let $v_{i,j}$ denote the j-th vector from V_i. Formally, the instances are described as follows:

Choose m such that there exists a value of p with $m = p(p-1)+1$ (thus, in these instances, the length of a vector increases with m), and set $n = p$.

- for each $k \in \{1, \ldots, p-1\}$, there are 1s in position $i-(k-1)p$ to position $i-(k-1)p+k-1$ (modulo p) in vector $v_{i,1}$, for each $i \in \{(k-1)p+1, kp\}$.
- There is a 1 in each position of the vector $v_{(p(p-1)+1,1}$.
- Each other vector is an all-zero vector.

Notice that the cost of an optimal solution equals p, whereas H_{seq} may find a solution with cost $m = p(p-1)+1$.

Corollary 2. *The worst case ratio of H_{seq} is at least $O(\sqrt{m})$.*

An instance with $p = 3$ is depicted in Figure 3.

V_1	V_2	V_3	V_4	V_5	V_6	V_7
100	000	000	110	000	000	111
000	010	000	000	011	000	000
000	000	001	000	000	101	000

Fig. 3. H_{seq}: an instance where OPT = 3 and cost(H_{seq}) = 7

Another natural heuristic to consider is the so-called Multiple Hub-Heuristic (see [1]), which can be informally described as in Algorithm (3):

The performance of the multiple hub heuristic MH can be arbitrarily bad. Indeed, consider the following instance. The length of each vector equals 2, ie, $p = 2$, and consider some even value for the number of sets m. let $n = \frac{m}{2} + 1$. The first vector of each of the sets V_i, $i = 1, 2, \ldots, \frac{m}{2}$ is specified as follows: For $i = 1, 2, \ldots, n$, put $v_{i,1} = (1\ 0)$; for $i = \frac{m}{2} + 1, \ldots, m$ put $v_{i1} = (0\ 1)$. All other vectors in the instance are equal to (0 0). It can be seen that $OPT = 2$ whereas cost(MH) $= \frac{m}{2} + 1$.

Corollary 3. *The worst case ratio of MH is $O(m)$.*

Notice that this performance is in contrast with the performance of the multiple hub-heuristic for other variants of decomposable minimum cost m-dimensional assignment problems, see [1].

Algorithm 3. Multiple-Hub-Heuristic MH

for h = 1 to m **do**
  for i = 1 to m **do**
    1. Solve an assignment problem between V_h and V_i, $i \neq h$, based on costs $c(u \vee v)$, $u \in V_h$,$v \in V_i$. Call the resulting matching M_{hi}.
  end for
  Combine all M_{hi}, to construct M_h.
end for
Output the min-cost solution of all M_h.

5 Fixed p

In this section we consider the yield optimization problem, i.e., we consider instances that feature 0-1 vectors only. We will argue that instances of the yield optimization problem with a fixed p can be solved in polynomial time (for each fixed m).

Consider a solution of the yield optimization problem. It consists of n 0-1 vectors. Thus, we can classify these n 0-1 vectors as belonging to at most 2^p different types (each type corresponding to a distinct 0-1 vector of length p). We use the symbol t to index these types.

We say that a vector from type t is *compatible* with a vector from type s if the vector of type t has a '1' in each of the positions where the vector of type s has a '1'. We write type t is *compatible* with a vector from type s as $t \succ s$. Further, given an instance of the yield optimization problem, we let k_s^i denote the number of 0-1 vectors of type s in set V_i, $s = 1, \ldots, 2^p$, $i = 1, \ldots, m$.

We construct the following formulation that features variables x_t:

$$x_t = \text{number of 0-1 vectors of type } t \text{ in the final solution, } t = 1, \ldots, 2^p.$$

We also need "transportation" type variables; for each $i = 1, \ldots, m, s, t = 1, \ldots, 2^p$:

$$z_{s,t}^i = \text{number of 0-1 vectors of type } s \text{ from set } V_i \text{ assigned to class } t.$$

The formulation (with parameter c_t referring to the number of '1's in a vector from type t):

$$\min \sum_{t=1}^{2^p} c_t x_t \tag{13}$$

$$\sum_{s:\ t \succ s} z_{s,t}^i = x_t \quad \text{for each } t = 1, \ldots, 2^p, i = 1, \ldots, m, \tag{14}$$

$$\sum_{t:\ t \succ s} z_{s,t}^i = k_s^i \quad \text{for each } s = 1, \ldots, 2^p, i = 1, \ldots, m, \tag{15}$$

$$x_t \quad \text{integer} \qquad \text{for each } t = 1, \ldots, 2^p, \tag{16}$$

The objective function (13) minimizes the total cost. Constraints (14)-(15) are the familiar transportation constraints. Notice further that integrality of x_t implies integrality of $z_{s,t}^i$.

Observe that this formulation involves $O(2^p)$ binary variables, $O(m2^{2p})$ continuous variables, and $O(m2^p)$ constraints.

Lemma 2. *Formulation (13)-(16) is correct.*

Proof. See Appendix.

When we fix p and m the above formulation has a fixed number of variables and constraints. Thus we can use Lenstra's algorithm to solve this IP in polynomial time. This implies:

Corollary 4. *For each fixed p, and for each fixed m, the yield maximization problem can be solved in polynomial time.*

It is true, however, that for the values of p encountered in practice, the above sketched approach is not practical.

References

1. Bandelt, H., Crama, Y., Spieksma, F.: Approximation algorithms for multi-dimensional assignment problems with decomposable costs. Discrete Applied Mathematics 49, 25–50 (1994)
2. Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. SIAM (2009)
3. Burkard, R., Rudolf, R., Woeginger, G.J.: Three-dimensional axial assignment problems with decomposable cost coefficients. Dicrete Applied Mathematics 65, 123–139 (1996)
4. Crama, Y., Spieksma, F.: Approximation algorithms for three-dimensional assignment problems with triangle inequalities. European Journal of Operational Research 60, 273–279 (1992)
5. Kann, V.: Maximum bounded 3-dimensional matching is max snp complete. Information Processing Letters 37, 27–35 (1991)
6. Reda, S., Smith, L., Smith, G.: Maximizing the functional yield of wafer-to-wafer integration. IEEE Transactions on VLSI Systems 17, 1357–1362 (2009)
7. Spieksma, F.C.R.: Multi-index assignment problems: complexity, approximation, applications. In: Pitsoulis, L., Pardalos, P. (eds.) Nonlinear Assignment Problems, Algorithms and Applications, pp. 1–12. Kluwer Academic Publishers (2000)
8. Taouil, M., Hamdioui, S.: Layer redundancy based yield improvement for 3d wafer-to-wafer stacked memories. In: IEEE European Test Symposium, pp. 45–50 (2011)
9. Verbree, J., Marinissen, E., Roussel, P., Velenis, D.: On the cost-effectiveness of matching repositories of pre-tested wafers for wafer-to-wafer 3d chip stacking. In: IEEE European Test Symposium, pp. 36–41 (2010)

Appendix

Proof of Lemma 2

Proof. Consider a feasible solution to the yield optimization problem. This solution prescribes for each type of vector in each set V_i how many of these vectors are assigned to a vector of type t. This determines the $z^i_{s,t}$ values; clearly, these values will satisfy constraints (14)-(16), since our solution is valid. Vice versa, consider $z^i_{s,t}$ values that satisfy (14)-(16). One can construct a feasible solution to WWI-m as follows: (1) Create a set X of n vectors with x_t vectors of type t. (2) Solve an assignment problem between X and V_i, for each $i = 1, \ldots, m$, based upon a bipartite graph $G = (L \cup R, E)$ involving a node in L for each vector in X a node in R for each vector in V_i, and two nodes are connected if the vector in X is compatible with the vector in V_i. This assignment problem will have a feasible solution since $z^i_{s,t}, x_t$ satisfy (14)-(16). (3) Construct m-tuples of vectors by matching m vectors one from each V_i together in an m-tuple if they all are matched to same vector in X in (2). This corresponds directly to a feasible solution. □

Author Index