Evripidis Bampis
Klaus Jansen (Eds.)

LNCS 5893

Approximation and Online Algorithms

7th International Workshop, WAOA 2009
Copenhagen, Denmark, September 2009
Revised Papers

Springer

Evripidis Bampis Klaus Jansen (Eds.)

Approximation and Online Algorithms

7th International Workshop, WAOA 2009
Copenhagen, Denmark, September 10-11, 2009
Revised Papers

Springer

Volume Editors

Evripidis Bampis
Université d'Evry Val d'Essonne, IBISC, CNRS FRE 3190
Place des Terasses, Tour Evry 2, 91000 Evry Cedex, France
E-mail: bampis@gmail.com

Klaus Jansen
Universität Kiel, Institut für Informatik
Olshausenstr. 40, 24098 Kiel, Germany
E-mail: kj@informatik.uni-kiel.de

Library of Congress Control Number: 2010924119

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

ISSN 0302-9743
ISBN-10 3-642-12449-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-12449-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The 7th Workshop on Approximation and Online Algorithms (WAOA 2009) 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 2009 took place in Copenhagen, Denmark, during September 10–11, 2009. The workshop was part of the ALGO 2009 event that also hosted ESA 2009, IWPEC 2009, and ATMOS 2009. The previous WAOA workshops were held in Budapest (2003), Rome (2004), Palma de Mallorca (2005), Zurich (2006), Eilat (2007), and Karlsruhe (2008). The proceedings of these previous WAOA workshops have appeared as LNCS volumes 2909, 3351, 3879, 4368, 4927, and 5426, respectively.

Topics of interest for WAOA 2009 were: algorithmic game theory, approximation classes, coloring and partitioning, competitive analysis, computational finance, cuts and connectivity, geometric problems, inapproximability results, mechanism design, network design, packing and covering, paradigms for design and analysis of approximation and online algorithms, parameterized complexity, randomization techniques, real-world applications, and scheduling problems.

In response to the call for papers, we received 62 submissions. Each submission was reviewed by at least three referees, and the vast majority by at least four 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.

We are grateful to Andrei Voronkov for providing the EasyChair conference system, which was used to manage the electronic submissions, the review process, and the electronic PC meeting. It made our task much easier. We would also like to thank all the authors who submitted papers to WAOA 2009 as well as the local organizers of ALGO 2009.

December 2009

Evripidis Bampis
Klaus Jansen

Organization

Program Co-chairs

Evripidis Bampis	University of Evry, France
Klaus Jansen	University of Kiel, Germany

Program Committee

Vincenzo Auletta	University of Salerno, Italy
Evripidis Bampis	University of Evry, France
Nikhil Bansal	IBM T.J. Research Center Watson, USA
Constantinos Daskalakis	Microsoft Research and MIT, USA
Christoph Durr	CNRS, Ecole Polytechnique, France
Friedrich Eisenbrand	EPFL Lausanne, Switzerland
Thomas Erlebach	University of Leicester, UK
Klaus Jansen	University of Kiel, Germany
Christos Kaklamanis	University of Patras, Greece
Sanjeev Khanna	University of Pennsylvania, USA
Monaldo Mastrolilli	IDSIA Lugano, Switzerland
Yuval Rabani	Technion Haifa, Israel
Nicolas Schabanel	CNRS, University Paris Diderot, France
Roberto Solis-Oba	University of Western Ontario, Canada
Neal Young	University of California Riverside, USA
Guochuan Zhang	Zhejiang University Hangzhou, China

Referees

Susanne Albers
Eric Angel
Elliot Anshelevich
Yossi Azar
Marin Bougeret
Andreas Brandstädt
Jarosław Byrka
Ioannis Caragiannis
Lin Chen
Marek Chrobak
Michael Elkin
Diodato Ferraioli
Dimitris Fotakis
Pierre Fraigniaud
Hiroshi Fujiwara
Stefan Funke
Mira Gonen
laurent gourvès
Fabrizio Grandoni
Danny Hermelin
Tobias Jacobs
Panagiotis Kanellopoulos
Jochen Könemann
Elias Koutsoupias
Sören Laue
Van Bang Le
Daniel Lokshtanov
Bin Lu

Shuang Luan
Páll Melsted
Julian Mestre
Ioannis Milis
Jérôme monnot
Hannes Moser
Nikolaus Mutsanas
Tim Nonner
Christina Otte
Konstantinos Panagiotou
Evi Papaioannou
Fanny Pascual
Ulrich Pferschy
Lars Prädel
Dömötör Pálvölgyi
Eric Remila
Thomas Rothvoss
Heiko Röglin
Laura Sanità
Ulrich M. Schwarz
Danny Segev
Bruce Shepherd
Gregory Valiant
Carmine Ventre
Andreas Wiese
Prudence W.H. Wong
Haifeng Xu
Deshi Ye
Vassilis Zissimopoulos

Table of Contents

On the Competitiveness of the Online Asymmetric and Euclidean Steiner Tree Problems

Spyros Angelopoulos

Max-Planck-Institut für Informatik
Saarbrücken 66123, Germany

Abstract. This paper addresses the competitiveness of online algorithms for two Steiner Tree problems. In the first problem, the underlying graph is directed and has bounded asymmetry, namely the maximum weight of antiparallel links in the graph does not exceed a parameter α. Previous work on this problem has left a gap on the competitive ratio which is as large as logarithmic in k. We present a refined analysis, both in terms of the upper and the lower bounds, that closes the gap and shows that a greedy algorithm is optimal for all values of the parameter α.

The second part of the paper addresses the Euclidean Steiner tree problem on the plane. Alon and Azar [SoCG 1992, Disc. Comp. Geom. 1993] gave an elegant lower bound on the competitive ratio of any deterministic algorithm equal to $\Omega(\log k/\log\log k)$; however, the best known upper bound is the trivial bound $O(\log k)$. We give the first analysis that makes progress towards closing this long-standing gap. In particular, we present an online algorithm with competitive ratio $O(\log k/\log\log k)$, provided that the optimal offline Steiner tree belongs in a class of trees with relatively simple structure. This class comprises not only the adversarial instances of Alon and Azar, but also all rectilinear Steiner trees which can be decomposed in a polylogarithmic number of rectilinear full Steiner trees.

1 Introduction

Problem statements and motivation The *Steiner tree* problem occupies a central place in combinatorial optimization, and has been studied extensively under many variants. The standard setting involves an underlying graph $G = (V, E)$, with a non-negative weight function $c : E \rightarrow \mathbb{R}^+$ over its edges (which reflects the *cost* of the edge). Given a subset $K \subseteq V$ of vertices also called *terminals*, the objective is to find a tree of minimum cost that spans all vertices in K. Here, the cost of the tree is defined as the sum of the cost of all the edges in the tree.

In this paper we focus on the following Steiner tree problems.

• In the *asymmetric* Steiner tree problem, the underlying graph is directed, and a specific vertex $r \in V$ is designated as the *root*. We define the *asymmetry* α of graph G as the maximum ratio of the cost of antiparallel links in G. More formally, let A denote the set of pairs of vertices in V such that if the pair u, v is in A, then either $(v, u) \in E$ or $(u, v) \in E$ (i.e, there is an edge from u to v or an edge from v to u or both). Then the edge asymmetry is defined as

$$\alpha = \max_{\{v,u\} \in A} \frac{c(v,u)}{c(u,v)}$$

Given a set $K \subseteq V$ of terminals, we seek the minimum cost *arborescence* rooted at r that spans all vertices in K. In addition, our aim is to express the performance of the algorithm in terms of the parameters k and α.

• In the *Euclidean* Steiner tree problem on the plane, there is no underlying graph. Instead, the input consists of k points (terminals) on the plane and the objective is to construct a connected graph that spans all terminals so as to minimize the total length (based on Euclidean distances). The case in which the distance between points of the plane is given by the rectilinear metric is known as the *Rectilinear Steiner tree* problem. Here, the tree consists only of horizontal and vertical segments. Clearly, any solution to the Euclidean Steiner tree problem can be translated to a solution to the rectilinear Steiner tree problem, at the expense of a constant-factor increase of the solution cost.

Steiner tree problems are often used in formulating multicast routing over computer networks, in that information must be disseminated from a designated source to all members of a subscribing group (namely the set of terminals K). The asymmetric Steiner tree problem is motivated by the observation that a directed graph is a more appropriate and realistic representation of a real network. A typical communication network consists of links asymmetric in the quality of service they provide. This observation led Ramanathan [9] to define the concept of graph asymmetry as a measure of network homogeneity. According to this measure, undirected graphs are the class of graphs of asymmetry $\alpha = 1$, whereas directed graphs in which there is at least one pair of vertices v, u such that $(v, u) \in E$, but $(u, v) \notin E$ are graphs with unbounded asymmetry ($\alpha = \infty$).

On the other hand, the Euclidean Steiner tree problem can be used in modeling problems where a number of facilities is given (on a certain planar region), and we want to guarantee connectivity among any given pair of these facilities (e.g., by building a network of roads, or by deploying a communications network). The objective translates into a solution that is as cheap as possible.

In this work we address the above problems from the point of view of *online* algorithms, in that the set K of terminals is not known in advance, but rather is revealed as a sequence of requests. Upon a new request for terminal t, the algorithm must guarantee a directed path from the root to t, in the case of the asymmetric Steiner tree problem, or simply that t is connected to the current solution, in the case of the Euclidean Steiner tree problem.

Related Work. We review some important results that pertain to online Steiner tree problems. For graphs of either constant or unbounded asymmetry, the competitive ratio is tight. For the former class, Imase and Waxman [8] showed that a simple greedy algorithm is optimal and achieves competitive ratio $\Theta(\log k)$. Berman and Coulston [5] extended the result to the Generalized Steiner Problem by providing a more sophisticated algorithm. Westbrook and Yan [10] showed that in directed graphs (of unbounded asymmetry), the competitive ratio of any algorithm can be as bad as $\Theta(k)$.

The first study of the online asymmetric Steiner tree problem is due to Faloutsos *et al.* [7] who showed that a simple greedy algorithm (to which we refer to as GREEDY) has competitive ratio $O(\min\{\alpha \log k, k\})$. The algorithm works by connecting each requested terminal u to the current arborescence by buying the edges in a least-cost directed path from the current arborescence to u. On the negative side, they showed a lower bound of $\Omega\left(\min\left\{\frac{\alpha \log k}{\log \alpha}, k\right\}\right)$ on the competitive ratio of every deterministic algorithm. A better analysis of GREEDY [2] provides an improved upper bound on the competitiveness of GREEDY, namely $O\left(\min\left\{\alpha\frac{\log k}{\log\log \alpha}, k\right\}\right)$. The same work showed a corresponding lower bound of $\Omega\left(\min\left\{\frac{\alpha \log k}{\log\log k}, k^{1-\epsilon}\right\}\right)$ on the competitiveness of any deterministic algorithm (and for every constant $0 < \epsilon < 1$). In recent work [3], a more careful analysis proves that GREEDY has a competitive ratio equal to $O\left(\min\left\{\max\left\{\alpha\frac{\log k}{\log \alpha}, \alpha\frac{\log k}{\log\log k}\right\}, k\right\}\right)$. The result almost matches the lower bound of $\Omega\left(\min\left\{\max\left\{\alpha\frac{\log k}{\log \alpha}, \alpha\frac{\log k}{\log\log k}\right\}, k^{1-\epsilon}\right\}\right)$ (where ϵ is any arbitrarily small constant) due to [7] and [2].

It is important to note that when $\alpha \in \Omega(k)$ the lower bound on the competitive ratio due to [7] is $\Omega(k)$, which is obviously tight (using the trivial upper bound of $O(k)$ for GREEDY). Thus the problem is interesting only when $\alpha \in o(k)$.

Even thought the bound of [3] is almost-tight, from a worst-case perspective a gap still remains. Indeed, [3] is tight when either $\alpha \in O(k^{1-\epsilon})$ (for some constant $\epsilon \in (0,1)$), or $\alpha \in \Omega(k)$, that is, for a broad range of values of asymmetry. However, when the asymmetry is such that $\alpha \in O(k)$, and $\alpha \neq O(k^{1-\epsilon})$, for any positive constant $\epsilon < 1$ (e.g, when $\alpha = k/\text{polylog}(k)$), the gap between upper and lower bounds can be large, namely logarithmic in k.

Concerning the online Steiner Tree in the Euclidean plane, Alon and Azar [1] presented an elegant adversarial construction that guarantees a lower bound of $\Omega(\log k/\log\log k)$. The only known upper bound is the $O(\log k)$ bound that applies to any distances induced by undirected graphs (and hence by extension to Euclidean distances). No better upper bounds have been known.

Contributions of this paper. In the first part of this paper, we give a tight bound for the online asymmetric Steiner tree problem. In particular, we show that when α is in the range of values for which [3] is not tight, the competitive ratio of GREEDY is $\Theta\left(\frac{\log(k/\alpha)}{\log\log(k/\alpha)}\right)$, and the bound is tight for any deterministic algorithm (c.f. Theorem 1 and Theorem 2). This completely resolves the problem of determining the optimal competitive ratio, for all values of asymmetry.

There are several reasons that motivate the close study of this problem. First, as argued earlier, for certain values of the asymmetry, a gap as large as logarithmic (in k) remains. From a worst-case point of view the gap is large, even though it occurs for a relatively narrow range of values of α (recall that $\Theta(\log k)$ is the competitive ratio in undirected graphs).

More importantly, algorithms for Steiner tree problems are often used as subroutines in solving more complex problems, and in many cases the competitive ratio for the complex problem is a function of the competitive ratio of the Steiner tree algorithm. A representative example is the problem of *file allocation* in general undirected networks. The influential work of Bartal *et al.* [4] shows that if there exists a c-competitive algorithm for the online Steiner tree problem (in undirected graphs), then it is possible to derive a randomized $(2+\sqrt{3})c$-competitive algorithm for file allocation. Note that, as with multicasting, file allocation is a problem motivated by distributed networks, and once again, it would only be natural to study it under directed graphs. Therefore, we expect that strict optimality results for Steiner trees will provide a useful tool in resolving other online network optimization problems, in settings which are more realistic in practice.

In the second part of the paper, we address the online Euclidean Steiner tree problem on the plain. Bridging the gap between the known lower and the upper bounds has been an outstanding open question in the area of competitive analysis for more than 15 years. Unfortunately, only a trivial upper bound is known. We make progress by showing that if the underlying optimal tree observes certain structural properties, then there exists an algorithm that is optimal. In particular, we require that the optimal, off-line tree is "close" (w.r.t. cost) to a rectilinear tree that is union of at most a polylogarithmic number of *basic trees* (c.f. Theorem 3 and Corollary 1). A formal definition of a basic tree is given in Section 3. The definition includes, for instance, any *Full Rectilinear Steiner tree* (or FST for brevity); see, e.g., [6] for a precise definition of a FST.

We need to further motivate our assumption on optimal trees. First, the lower bound of Alon and Azar, although fairly technical, is based on an instance for which the optimal Steiner tree is a single basic tree. Thus our result states that if we aim to improve the lower bound, more complicated constructions will be required. Second, the result provides a connection between the number of basic trees and the performance of an algorithm. This is along the lines of known results on approximation algorithms, which relate the (offline) construction of rectilinear Steiner trees and FST's (see [12] and [11] for representative examples). Third, the result follows from techniques used in the analysis of the greedy algorithm for the online asymmetric Steiner tree problem.

2 A Tight Bound for Online Asymmetric Steiner Trees

2.1 The Lower Bound

The main result of this section is the following lower bound:

Theorem 1. *The competitive ratio of every deterministic algorithm is* $\Omega\left(\alpha\frac{\log(k/\alpha)}{\log\log(k/\alpha)}\right)$, *assuming that* $\alpha \in o(k)$.

For the proof of Theorem 1 we will construct an adversarial input, namely a graph G and an appropriate sequence σ of terminal requests. The graph G will be defined using $\widehat{G}$ as a building block, where $\widehat{G}$ is a parameterized version of the adversarial graph used in the construction of the lower bound in [2].

Construction of the adversarial graph G. We will begin by defining some auxiliary constructions. Let v, u be two vertices in a graph, and let the directed edge (v, u) have cost c, whereas the antiparallel edge $\overline{e} = (u, v)$ has cost αc, where α is the asymmetry of the graph in which u, v belong. We say that we *insert a vertex w at height h in e* if we introduce a new vertex w and replace $e, \overline{e}$ with new edges of costs $c(v, w) = c - h$, $c(w, u) = h$, $c(u, w) = \alpha h$, and $c(w, v) = \alpha(c - h)$ (for the sake of visualisation, we should think of v as located higher than u). Note that the insertion maintains the asymmetry of the graph.

Second, let $T_1 = \{v_1, \ldots, v_l\}$ and $T_2 = \{v'_1, \ldots, v'_l\}$ denote two disjoint sets of l vertices each (again, we should think of vertices in T_1 as being located "higher" than vertices in T_2). In addition, we require that T_1 and T_2 have the property that all edges of the form $e_i = (v_i, v'_i)$ have the same cost, say c, and all antiparallel edges have cost αc. Informally, the "downwards" direction is cheap, whereas the "upwards" direction is expensive , and this is a property that will be maintained throughout the construction. Let E denote the set $\{e_i : i \in [1, l]\}$. We call index $i \in [1, l]$ the *i-th column*, and require that l is a power of 2, and that it is large compared to k (e.g., at least 2^k). On the collection of columns E we will define a construction called *block* which we denote by $B(E, m, h)$. Here m and h are parameters used in the construction, with $h \leq c$ and $m < k$, and E is the set of columns induced by T_1 and T_2.

In a nutshell, $B(E, m, h)$ is built by inserting appropriate vertices (and edges) in E, in a layered fashion. There are m *layers* inserted in total. Layer i consists of l vertices inserted, one for each column, at height equal to $\Theta(h/m)$. Let $w^{i,1}, \ldots, w^{i,l}$ denote these l vertices. We group the vertices of layer i into appropriate disjoint groups of the same size, denoted by $S^{i,1}, \ldots S^{i,s_i}$, which we call *$s$-sets*. Also, for each group of layer i, say $S^{i,j}$, we add a vertex $u^{i,j}$, as well as edges from every w-vertex in $S^{i,j}$ to $u^{i,j}$, of cost $c^{i,j}$, whereas the antiparallel edges from ($u^{i,j}$ to every w-vertex in $S^{i,j}$) have cost $\alpha c^{i,j}$. The partition of vertices of each layer into s-sets, as well as the precise values of $c^{i,j}$ are functions of the parameters m, c, h and α. Figure 1 gives an example of a block. The precise construction of a block is fairly complicated; we will focus only on important properties.

We say that an s-set *crosses* a certain set of columns if and only if the set of columns in which the vertices of S lie intersects the set of columns in question. Two s-sets cross each other iff the intersection of the sets of columns crossed by each one is non-empty. Note that there is a 1-1 correspondence between the sets $S^{i,j}$ and vertices $u^{i,j}$, which means that several properties/definitions pertaining to s-sets carry over to the corresponding u vertices (e.g., we will say that two u vertices cross if their s-sets cross).

Let $T_1 = \{v_1, \ldots, v_l\}$ and $T_2 = \{v'_1, \ldots, v'_l\}$ denote two sets of l vertices each (here l has to be large compared to k, although it suffices to set $l > 2^k$), with

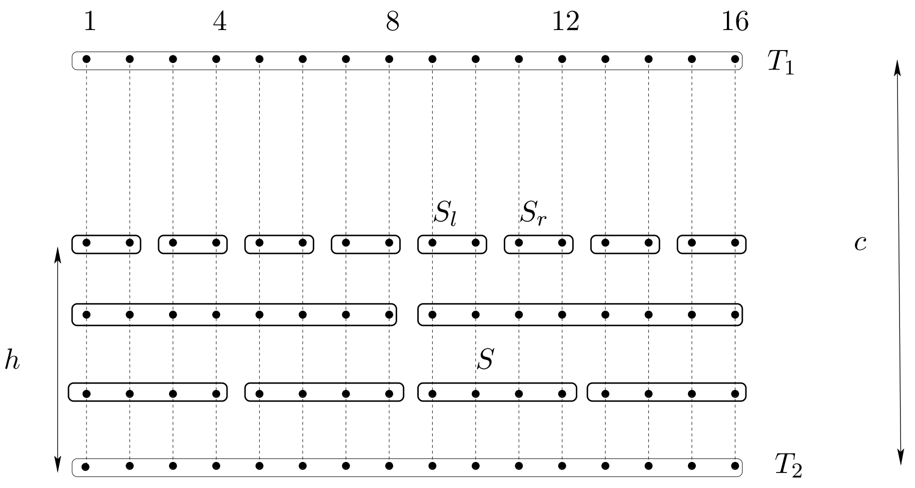


Fig. 1. An example of a block construction. Here $l = 16$ (i.e., there are 16 columns), and the block consists of three layers (the first layer consists of 4 s-sets, the second of 2 s-sets and the third of 8 s-sets. For simplicity, the figure illustrates only s-sets: in reality, each s-set is associated with a u-vertex. Note that according to the definition of Section 2.1 the two s-sets S_l and S_r are both children of the s-set denoted by S.

$c(v_i, u_i) = 1$ and $c(u_i, v_i) = \alpha$. Let E denote the set of columns induced by T_1 and T_2. There is also a root r and edges from r to each v_i of infinitesimally small cost, and the same holds for the antiparallel edges from v_i to r). In [2], graph $\widehat{G}$ then is derived by adding a single block $B(E, k, 1)$, where k is the total number of terminals requested by the adversary.

We proceed with the description of the adversarial graph G. At a high level, the construction is based on *phases* of insertions of appropriately defined blocks, unlike $\widehat{G}$ which consists of a single block. We begin with T_1, T_2, E and r defined as above. Given the set of columns E, in the first phase we add the block $B_{1,1} = (E, m, m/k)$, where $m \leq k$ is a function of k and α that will be specified later. Inductively, let $B_{i,1}, B_{i,2}, \ldots$ denote the blocks added during the i-th phase in the construction of G. Consider the highest layer added so far, namely the highest layer added at phase i, and let $\{S_{i,1}, S_{i,2}, \ldots\}$ denote the collection of s-sets at this layer. For a set $S_{i,j}$, in the above collection of highest s-sets, let $T_{i,j} \subset T_1$ denote the set of vertices in T_1 of the same column index as vertices in $S_{i,j}$. The pairs $(T_{i,j}, S_{i,j})$ induce a partition of the columns of E into disjoint subsets of columns (of equal size). More precisely, let $E_{i,j}$ denote the set of edges (v_p, w), with $p \in [1, l]$ such that $w \in S_{i,l}$. Then, phase $i + 1$ consists of adding the blocks $B_{(i+1),j} = B(E_{i,j}, m, m/k)$, for all j for which the column sets $E_{i,j}$ are defined.

We want to ensure that $\Theta(k)$ layers are added in total, since the adversarial request sequence on G will request one u-vertex per layer. Thus the above process is comprised of $\Theta(k/m)$ phases. An illustration is given in Figure 2.

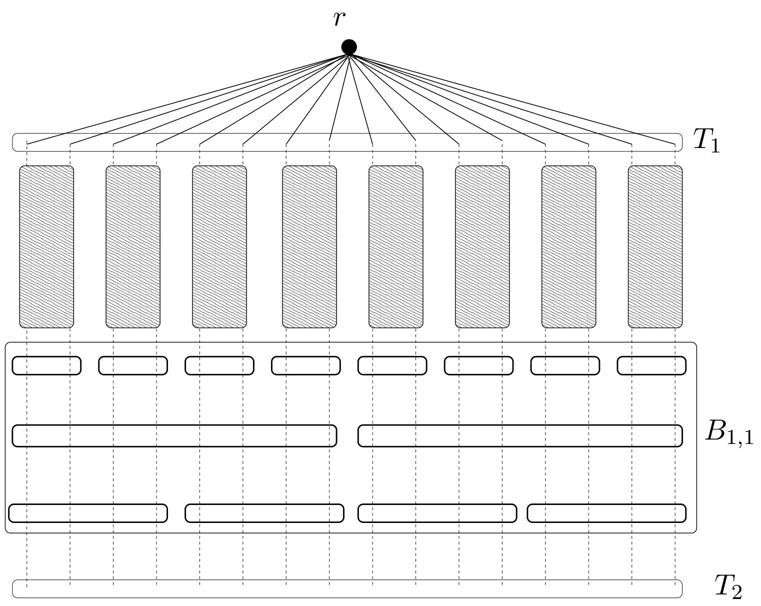


Fig. 2. An example of the construction of G, for the case in which the process requires two phases. Here, each phase will insert a block as depicted in Figure 1, as evident in the shape of $B_{1,1}$. In particular, the eight shaded regions corresponds to the eight blocks of the form $B_{2,1} \ldots B_{2,8}$.

The algorithm–adversary game. We will describe how to construct an adversarial sequence of requests σ in G. We begin by describing, in high-level, the algorithm/adversary game in $\widehat{G}$, and the corresponding request sequence $\widehat{\sigma}$. Later on, we will built upon this game in order to define σ.

• **The adversarial game on $\widehat{G}$, and the sequence $\widehat{\sigma}$**
Consider $\widehat{G}$ with k levels in total, and let x be the solution of $x^x = k$, hence $x = \Theta(\log k / \log\log k)$. The adversarial request sequence $\widehat{\sigma}$ consists of u-vertices exclusively, and is comprised of x *rounds*: In particular, in round $i \leq x$, $\Theta((x+1)^i)$ terminals are requested.

Every time a new terminal, say u, is requested in $\widehat{\sigma}$, the online algorithm must guarantee the existence of a directed path from the root to u, possibly buying new edges. Among the many such paths the algorithm may establish, the adversary will fix one such path, to which we refer as the *connection path for* u, denoted by $p(u)$. For the lower-bound analysis, we charge parts (edges) of a connection path, then so long as we guarantee that each edge is charged at most once, the total cost of all charged edges is a lower bound on the algorithm's cost. We need to summarize some properties of the sequence $\widehat{\sigma}$, and how the edges of the connection paths are charged.

As argued in [2], for every requested terminal u, the connection path $p(u)$ can be of one of the following two forms:

– Connection *from* r: A connection path which originates from r.
– Connection path *from above/below*. A connection path which originates from a previously requested vertex that lies higher (resp. lower) than the requested vertex u.

A main design aim in the construction of $\widehat{G}$ and $\widehat{\sigma}$ is the presence of consecutive pairs of requests in a *parent/child* relation. The precise definition is as follows: Consider the three s-sets S, S_l, S_r in $\widehat{G}$ with the property that S crosses columns $i, \ldots j$ (with $j - i + 1$ even number) whereas S_l and S_r cross columns $i, \ldots (i + j - 1)/2$ and $(i + j + 1)/2 + 1, \ldots j$, respectively. In addition, S_l and S_r are at the same layer in $\widehat{G}$ and both higher than S. We call S_l and S_r the *left* and *right* children of S, respectively, and S their *parent*. The definition extends to the corresponding u-vertices of these s-sets in the natural way: two u-vertices are in parent/child relation if their corresponding s-sets are in such relation.

The sequence $\widehat{\sigma}$ is defined in such a way that almost all consecutive pairs of requested terminals in $\widehat{\sigma}$ are in parent/child relation. For such terminals, the adversary takes advantage of the structure of $\widehat{G}$: Suppose that u is the last requested terminal and that the children of u (say u_l, u_r) are present in $\widehat{G}$. Then it is easy to argue that no matter what the choice of the connection path $p(u)$, the path will cross[1] exactly one of u_l, u_r: if it crosses u_l, then the next terminal to be requested is u_r, and vice versa.

We summarize the important properties concerning $\widehat{\sigma}$. We say that a terminal u requested in the i-th round of the game is *typical*, if the next requested terminal in $\widehat{\sigma}$ is a child of u in $\widehat{G}$. It turns out that there are $\Theta((x+1)^i)$ typical terminals in round i, thus almost all terminals in a round are typical. We also define the *depth* of u in $\widehat{G}$ as the total cost of the directed path from the root r to the s-set to which u corresponds.

Property 1 (cost property in $\widehat{\sigma}$). Let u be a typical request of round i, which is at depth d in $\widehat{G}$. Then:

(i) If $p(u)$ is from the root, then the algorithm is charged cost $\Omega(d)$.
(ii) If $p(u)$ is from above/below, then the algorithm is charged cost $\Omega\left(\frac{a}{(x+1)^i}\right)$.

The following set of properties will be instrumental in deriving the adversarial sequence σ in G. The first property implies that an offline algorithm suffices to buy edges of a single column (in the "downwards" direction), namely a column that crosses the last terminal in $\widehat{\sigma}$, as well as the corresponding directed edges from s-sets to requested terminals. The second property will be useful in applying iteratively the game in $\widehat{G}$ to the design of the game in G, one time per phase.

Property 2 (structural property in $\widehat{\sigma}$). Let u be the current request in $\widehat{\sigma}$, then:

(i) Every column that crosses u crosses all previously requested terminals in $\widehat{\sigma}$.

[1] We say that the connection path for a terminal crosses an s-set (or its corresponding u-vertex) if it contains (vertical) edges in a column that is crossed by the s-set.

(ii) No connection path of any previously requested terminal crosses any columns of u, and no edges in columns that cross u are charged by the adversary (with the exception of u itself).

• The adversarial game on G, and the sequence σ

We now show how to use $\widehat{G}$ and $\widehat{\sigma}$ in order to construct an adversarial sequence σ for G. At a high level, the game between the algorithm and the adversary in G proceeds in $\Theta(k/m)$ *phases*: the i-th phase requests u-vertices which belong in a block in G that was added in the i-th phase of its construction. Within this specific block, terminals are requested in a manner similar to requests in $\widehat{\sigma}$: this is because $\widehat{G}$ essentially consists of a single block.

We describe how σ is derived, as well as the actions of the adversary. In the first phase, the adversary requests a total of m u-vertices in the lowest block, namely $B_{1,1}$. This is essentially identical to the game played in $\widehat{G}$, i.e., vertices are requested in rounds. The only difference lies in the cost charged for the connection paths (informally, in G, the vertices of $B_{1,1}$ are in a higher depth than vertices in $\widehat{G}$). However this does not affect the decisions of the adversary, since we can still classify connection paths (from the root, or from above/below), and we can still classify terminals as typical, in the same manner as in $\widehat{\sigma}$. In particular, the last requested terminal of the first phase satisfies Property 2.

Inductively, suppose that the adversary has requested terminals in phase i, and suppose that every terminal requested in phases up to and including phase i satisfies Property 2. We will describe the requests of phase $i+1$. Let u denote the last terminal requested in the i-th phase, and S its corresponding s-set. From construction of G, there is a unique block, say $B_{i+1,j}$, for some index j, which was added in iteration $i+1$ and which is defined over the set of columns which are crossed by S. From the structural property, and in particular, Property 2(ii), right before u is requested no columns crossed by u are charged by the adversary. This implies that, once again, the adversary can play the same game in $B_{i+1,j}$ as the game played in $\widehat{G}$, and charge columns in identical manner, without ever charging an edge more than once. Phase $i+1$ then is comprised by the u-vertices requested *in rounds* within block $B_{i+1,j}$. It also follows from the construction of G, as well as the induction hypothesis, that any terminal requested during this phase satisfies the structural property. The sequence σ is the sequence derived by combining the requests of all phases: there are $\Theta(k/m)$ phases in total, and $\Theta(m)$ requests per phase, hence there are $\Theta(k)$ requests in σ in total.

Because of the correspondence between phases and blocks, we say that a terminal requested in σ is typical if it is typical in the corresponding block in which it belongs. Based on this,we derive the following property concerning the cost of typical terminals in σ. Here, x is set to be $\Theta(\log m/\log\log m)$.

Property 3. Let u be a typical request of phase j and round i, at depth d in G.

(i) If $p(u)$ is from the root, then the algorithm is charged cost $\Omega(d)$.
(ii) If $p(u)$ is from above/below, then the algorithm is charged cost $\Omega\left(\frac{a}{(x+1)^i}\frac{m}{k}\right)$.

2.2 Analysis

For the purpose of the analysis, we need the following lemma:

Lemma 1. *For the sequence of requests σ on graph G, the following hold:*

- *The cost of the optimal offline algorithm is bounded by a constant.*
- *The cost of any deterministic algorithm is $\Omega\left(\min\left\{\frac{k}{m},\ \alpha \cdot \frac{\log m}{\log\log m}\right\}\right)$*

Once we prove Lemma 1, then Theorem 1 follows by selecting a value m, so as to minimize the cost expression. In particular, we choose $m = \Theta\left(\frac{k}{\alpha}\frac{\log\log(k/\alpha)}{\log(k/\alpha)}\right)$. Substituting this value in the cost expression, it is easy to see that both k/m and $\alpha \cdot \frac{\log m}{\log\log m}$ are in $\Omega\left(\alpha \cdot \frac{\log(k/\alpha)}{\log\log(k/\alpha)}\right)$, and the theorem is proved.

Proof sketch of Lemma 1. The upper bound on the cost of the optimal algorithm follows from the fact that there exists a column that crosses all terminals in σ: an offline algorithm will buy all downwards edges in this column, as well as edges from the s-sets crossed by the column in question to all u-vertices in σ. The cost of the latter edges is such that their total contribution is small.

For the lower bound on the cost of any algorithm, the rough idea is that we classify each phase into one of two possible groups, depending on whether the algorithm pays its cost mostly due to rule (i) or rule (ii) of Property 3. We can prove that the first group contributes a total cost of $\Omega(k/m)$, whereas the second group contributes $\Omega(\alpha \log m/\log\log m)$. □

2.3 The Upper Bound

Theorem 2. *Suppose that α is such that $\alpha \in \omega(k^{1-\epsilon})$, for every constant $\epsilon \in (0,1)$, and also $\alpha \in o(k)$. Then for every request sequence σ, the cost paid by* Greedy *is $c_{GR}(\sigma) = O\left(\alpha \cdot \frac{\log(k/\alpha)}{\log\log(k/\alpha)}\right)c(T^*)$, where $k = |\sigma|$, and T^* is the optimal arborescence for σ.*

Proof sketch. Partition T^* into $\Theta(\alpha)$ edge-disjoint trees $T_1, T_2, \ldots$ each containing $\Theta(k/\alpha)$ terminals. Then the total cost for serving the *first* request in each of these subtrees is small. On the other hand, we can show that the cost for serving all remaining terminals within T_i is $O\left(\alpha\frac{\log(k/\alpha)}{\log\log(k/\alpha)} \cdot c(T_i)\right)$. The lemma follows from the edge-disjointness of the T_i's. □

3 A Non-trivial Bound for Online Euclidean Steiner Tree on the Plane

In this section we present an algorithm for the online Steiner tree problem in the Euclidean plane, and prove that achieves better competitive ratio than $O(\log k)$, assuming the optimal Steiner tree has certain structural properties. The algorithm is as follows: Suppose that terminal u is requested, and let T_{curr} denote the current Steiner tree built so far. The algorithm first finds a path of minimum cost from u to the current tree, say p_u, and buys this path. Let $c(p_u)$ denote the cost of the path in question. In addition, the algorithm buys horizontal and vertical segments of size $2c(p_u)$ each, such that u is at the center of each segment[2]. Hence in serving request u, the algorithm pays a total cost equal to $5c(p_u)$.

Before proceeding with the main result, we will show that the algorithm is optimal when the underlying optimal Steiner tree has a relatively simple structure, in particular, when it is a *basic* tree with respect to the set of requested terminals. We begin with the definition of a basic tree.

Definition 1. *Let $K' = \{u_1, \ldots, u_{k'}\}$ denote a set of k' terminals, and let T' be a tree that spans K' on the Euclidean plane. We call the tree T'* basic (with respect to K') *if the following conditions are met: T' consists of a segment P, which is either horizontal or vertical, and which we call the* backbone *of T, as well as k' disjoint paths $t_1, \ldots t_{k'}$, one for each terminal in K', which we call the* terminal paths*. Here, each terminal path t_i is such that u_i is one of the end-points of path t_i; the other end-point must be part of the backbone P.*

The following is the main technical result, which states that if the optimal Steiner tree for a set of terminals K is "close" to a basic tree wrt K, then the online algorithm we proposed achieves the best-possible competitive ratio.

Theorem 3. *Let K be a set of k terminals, and let T^* denote the optimal Steiner tree for K. If there exists a basic tree T such that $c(T) = O(c(T^*))$, then the cost of the online algorithm for requests in K is $O(\log k/\log\log k) \cdot c(T^*)$.*

The proof of Theorem 3 is based on techniques developed for the analysis of the greedy algorithm for the online asymmetric Steiner tree in [3]. More precisely, the central technical result in [3] is first derived for a class of directed graphs (called *combs*) whose structure is very similar to the structure of a a basic tree (see the statement of Theorem 3 in [3]). However, there are some important technical difficulties, in that the analysis in [3] yields a high cost with respect to the backbone of a comb (which would translate to a high cost with respect to the backbone of a basic tree for our algorithm). Instead, we need a refined analysis, that takes into account the geometric nature of the problem.

Specifically, we will rely on the following property (Lemma 2). In informal terms, the lemma states that if a tree T' is basic wrt a set of terminals K', then the online algorithm (on requests drawn from the set K') incurs cost which has only a linear dependency in the backbone cost.

Lemma 2. *Let $K' = \{u_1, \ldots u_{k'}\}$ be a set of k' terminals, and let T' be a basic tree wrt K', with backbone P of cost $c(P)$, and k' terminal paths $t_1, \ldots t_{k'}$. Suppose that terminals in K' arrive online. Then the cost of the algorithm is $O(c(P) + k' \cdot c_{\max})$, where $c_{\max}$ is the cost of the longest path among terminal paths $t_1, \ldots t_{k'}$.*

[2] The resulting graph may not be a tree, however this is not an issue since we only require a connected graph that spans all terminals.

It is worth pointing out that the construction for the lower bound of Alon and Azar [1] is such that the optimal Steiner tree is a basic rectilinear tree. Note also that any full (rectilinear) Steiner tree (FST) is trivially a basic tree. Motivated by the decomposition of rectilinear Steiner trees to edge-disjoint FST's , we can also decompose every rectilinear Steiner tree into a collection of edge-disjoint basic trees. We can thus extend our result to all instances whose optimal Steiner tree consists of a small number of basic Steiner trees, in this decomposition.

Corollary 1. *Let K denote a set of k terminals, and let T^* denote the optimal rectilinear Steiner tree for K. If T^* can be decomposed into a polylogarithmic (in k) number of basic trees, then the cost of the online algorithm for requests in K is $O(\log k/\log\log k) \cdot C(T^*)$.*

References

[1] Alon, N., Azar, Y.: On-line Steiner trees in the Euclidean plane. Discrete and Computational Geometry 10, 113–121 (1993)
[2] Angelopoulos, S.: Improved bounds for the online Steiner tree problem in graphs of bounded edge-asymmetry. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 248–257 (2007)
[3] Angelopoulos, S.: A near-tight bound for the online Steiner tree problem in graphs of bounded asymmetry. In: Proceedings of the 16th Annual European Symposium on Algorithms, pp. 76–87 (2008)
[4] Bartal, Y., Fiat, A., Rabani, Y.: Competitive algorithms for distributed data management. Journal of Computer and System Sciences 51(3), 341–358 (1995)
[5] Berman, P., Coulston, C.: Online algorithms for Steiner tree problems. In: Proceedings of the 29th Symp. on the Theory of Computing, pp. 344–353 (1997)
[6] Bern, M., Eppstein, D.: Approximation Algorithms for NP-hard problems. In: Hochbaum, D.S. (ed.) Approximation Algorithms for Geometric Problems, Ch. 8. PWS Publishing Company (1997)
[7] Faloutsos, M., Pankaj, R., Sevcik, K.C.: The effect of asymmetry on the on-line multicast routing problem. Int. J. Found. Comput. Sci. 13(6), 889–910 (2002)
[8] Imase, M., Waxman, B.: The dynamic Steiner tree problem. SIAM Journal on Discrte Mathematics 4(3), 369–384 (1991)
[9] Ramanathan, S.: Multicast tree generation in networks with asymmetric links. IEEE/ACM Transactions on Networking 4(4), 558–568 (1996)
[10] Westbrook, J., Yan, D.C.K.: Linear bounds for online Steiner problems. Information Processing Letters 55(2), 59–63 (1995)
[11] Winter, P., Zachariasen, M.: Euclidean Steiner minimum trees: An improved exact algorithm. Networks 30(3), 149–166 (1997)
[12] Zachariasen, M.: Rectilinear full Steiner tree generation. Networks 2(33) (1999)

Extension of the Nemhauser and Trotter Theorem to Generalized Vertex Cover with Applications

Reuven Bar-Yehuda[1], Danny Hermelin[2,⋆], and Dror Rawitz[3]

[1] Department of Computer Science, Technion IIT, Haifa 32000, Israel
reuven@cs.technion.ac.il

[2] Department of Computer Science, University of Haifa, Haifa 31905, Israel
danny@cri.haifa.ac.il

[3] Department of Electrical Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel
rawitz@eng.tau.ac.il

Abstract. The Nemhauser&Trotter Theorem provides an algorithm which is frequently used as a subroutine in approximation algorithms for the classical Vertex Cover problem. In this paper we present an extension of this theorem so it fits a more general variant of Vertex Cover, namely the Generalized Vertex Cover problem, where edges are allowed not to be covered at a certain predetermined penalty. We show that many applications of the original Nemhauser&Trotter Theorem can be applied using our extension to Generalized Vertex Cover. These applications include a $(2-\frac{2}{d})$-approximation algorithm for graphs of bounded degree d, a PTAS for planar graphs, a $(2-\frac{\lg\lg n}{2\lg n})$-approximation algorithm for general graphs, and a $2k$ kernel for the parameterized Generalized Vertex Cover problem.

1 Introduction

Given a graph $G = (V, E)$ with vertex weights, the classical Vertex Cover problem asks to find a minimum weight subset of vertices $S \subseteq V$ that covers all edges in G, *i.e.* a subset S with $S \cap e \neq \emptyset$ for all $e \in E$. The Vertex Cover problem is one of the most well-studied problems in theoretical computer science and discrete mathematics in general, a study dating back to König's classical early 1930s result [1], and probably even prior to that. In 1972, Karp listed the decision version of Vertex Cover in his famous list of initial twenty-one NP-complete problems [2].

One of the most well known results about Vertex Cover is the half-integrality of the LP-relaxation of the standard integer programming formulation of Vertex Cover (see, e.g., [3]). This result directly implies a 2-approximation algorithm for vertex cover (as observed by Hochbaum [4]). In 1975, only three years after the publication of Karp's famous NP-complete list, Nemhauser and Trotter published their seminal paper [5] in which they present a reduction that

⋆ Supported by the Adams Fellowship, Israel Academy of Sciences and Humanities.

reduces the problem of finding a vertex cover in an arbitrary graph G, to that of finding a vertex cover in a subgraph of G whose total weight is not much more than the weight of any of its vertex covers. This reduction is based on the half-integrality of Vertex Cover, and it adds additional structure to the Vertex Cover problem in general. Indeed, after applying the Nemhauser&Trotter reduction, one can use the total weight of the graph as a yardstick for analyzing approximate solutions, rather than use the weight of the optimal solution of which there is rarely any knowledge of. Below is a precise statement of the Nemhauser&Trotter Theorem:

Theorem 1 (Nemhauser&Trotter [5]). *Let (G, w) be an instance of* Vertex Cover*, with $G = (V, E)$ and $w : V \to \mathbb{Q}^{\geq 0}$. Then there is a polynomial-time algorithm that partitions the vertices of G into three subsets, V_1, V_0 and $V_{1/2}$, such that:*

(i) if $S_{1/2}$ is an α-approximate solution for $(G[V_{1/2}], w)$, then $V_1 \cup S_{1/2}$ is an α-approximate solution for (G, w), for all $\alpha \geq 1$, and
(ii) the w-weight of any vertex cover in $G[V_{1/2}]$ is at least $\frac{1}{2}\sum_{v \in V_{1/2}} w(v)$.

The first condition of the theorem implies that we can restrict our attention to $G[V_{1/2}]$, ignoring vertices of V_1 and V_0 in G. The second condition of the theorem implies that finding a vertex cover of $G[V_{1/2}]$ that is properly contained in $V_{1/2}$ is guaranteed to give a vertex cover of G whose weight is strictly less than twice the optimum.

It is important to note that the vertex sets V_0, $V_{1/2}$ and V_1 correspond to the values given to the variables by some half-integral optimal solution x^* to the LP-relaxation of vertex cover. Namely, $V_i = \{u : x^*_u = i\}$, for every $i \in \{0, 1/2, 1\}$. It is not hard to verify that Property (ii) of Theorem 1 follows directly from the half-integrality of the optimal solution x^*. Moreover, Property (i) holds for every LP-based approximation algorithm since the optimal fraction solution of $G[V_{1/2}]$ is $(1/2, \ldots, 1/2)$. Nemhauser&Trotter [5] proved that taking V_1 into the solution is a local optimization step, namely that even a non LP-based approximation algorithm may be used to augment V_1.

The Nemhauser&Trotter Theorem is an essential part of Hochbaum's $(2 - \frac{2}{d})$-approximation algorithm for graphs of bounded degree d [6], and of the $(2 - \frac{\lg \lg n}{2 \lg n})$-approximation algorithm for general graphs given in [7]. In fact, many known approximation algorithms for Vertex Cover and its special cases use the Nemhauser&Trotter Theorem as a subroutine. Two other good examples are the PTASs of Lipton-Tarjan [8] and Baker [9] for Vertex Cover in planar graphs[1], where one finds an optimal solution in a large fraction of the graph, and adds all remaining vertices to get a solution for the entire graph. We mention also Chen *et al.* [10] who observed that the Nemhauser&Trotter Theorem gives a $2k$ kernel for the parameterized variant of Vertex Cover in the parameterized complexity setting, when the parameter taken is the total weight of the required vertex cover (see also [11]).

[1] Baker's algorithm [9] originally did not use the Nemhauser&Trotter Theorem, but adding it as a preprocessing step makes the analysis somewhat simpler.

In this paper we focus on a natural generalization of Vertex Cover, which can be thought of as the "prize-collecting" version of the problem. In this variant, each edge in the given graph is allowed to be left uncovered at a certain predetermined penalty. Thus, the input now consists of a graph with vertex and edge weights, and the goal is to minimize the total weight of vertices selected to a solution, plus the total weight of edges not covered by the solution. Observe that this is in fact a generalization of Vertex Cover, since we return to the original problem by setting all edge weights to ∞. We call this generalization of Vertex Cover, the Generalized Vertex Cover problem (GVC):

Instance: A graph $G = (V, E)$ and a weight function $w : V \cup E \to \mathbb{Q}^{\geq 0}$.
Solution: A subset S of V.
Measure: $cost(S) = \sum_{v \in S} w(v) + \sum_{e \in E, e \cap S = \emptyset} w(e)$.

The first to consider GVC was Hochbaum [12], who provided a 2-approximation algorithm, and also pointed out that GVC is polynomial-time solvable in bipartite graphs. The time complexity of this algorithm was later improved in [13], where a d-approximation algorithm for GVC in d-hypergraphs was given as well. Hassin and Levin [14] studied a problem that extends GVC in which one pays a penalty for not covering an edge and a smaller penalty for covering an edge only by one of its end-points. They presented a 2-approximation algorithm for this problem. We further note that other "prize collecting covering" problems were also studied extensively in the literature. This includes the paper by Hassin and Tamir [15] who considered the prize collecting variant of the Facility Location on the Real Line problem, and the work of Goemans and Williamson [16] who presented approximation algorithms for the prize collecting versions for Triangle Inequality Traveling Salesman and Steiner Tree. See also [17,18] for other prize collecting facility location problems.

Due to the importance that the Nemhauser&Trotter Theorem plays in designing approximation algorithms for Vertex Cover and its special cases, a natural question to ask is whether a similar theorem can be found for GVC. Observe that the theorem does not carry-on immediately to the more general case due to the different way that the edges now come into play; in fact, this poses a difficulty even in stating the theorem for the more general case. The main result of this paper overcomes these difficulties and gives an affirmative answer to the question above by proving a slightly different variant of the Nemhauser&Trotter Theorem, which is essentially the same for most algorithmic applications. The following is a precise statement of our result:

Theorem 2. *Let (G, w) be an instance of* GVC, *with $G = (V, E)$ and $w : V \cup E \to \mathbb{Q}^{\geq 0}$. Then there is a polynomial-time algorithm that partitions the vertices of G into three subsets, V_1, V_0 and $V_{1/2}$, and constructs another weight function $\widetilde{w} : V \cup E \to \mathbb{Q}^{\geq 0}$, such that:*

(i) if $S_{1/2}$ is an α-approximate solution for $(G[V_{1/2}], \widetilde{w})$, then $V_1 \cup S_{1/2}$ is an α-approximate solution for (G, w), for all $\alpha \geq 1$, and
(ii) the $\widetilde{w}$-cost of any subset $S \subseteq V_{1/2}$ is at least $\frac{1}{2} \sum_{v \in V_{1/2}} \widetilde{w}(v)$.

Observe the difference in the second condition of the theorem which is necessary, since any subset of vertices is a potential solution in GVC. This is what makes the proof of the theorem in the generalized case more challenging. Another challenge is that as the edges in GVC play a different role, we are not guaranteed the combinatorial structure provided by the original theorem. For instance, in the original theorem the subset V_0 in the partition had to be an independent set, as otherwise any vertex cover had to include at least one vertex from V_0. In our case we can never require such a condition; we must assume that there can be an edge between any pair of vertices in V_0. Furthermore, in the original theorem, V_1 must separate V_0 from $V_{1/2}$ in G. Again, in our case this is not necessarily so, which makes the reduction more difficult since we insist that the resulting subgraph be induced, *i.e.* obtained only by deleting vertices, a fact that allows carrying-on *hereditary* properties of G through the reduction.

With the help of Theorem 2, we can show that many algorithms for Vertex Cover which use the Nemhauser&Trotter Theorem as a subroutine, can be modified so that they apply also for the more general case. In particular, we obtain the following new results for GVC as almost immediate corollaries of Theorem 2:

1. A $(2 - 2/d)$-approximation algorithm for graphs of bounded degree d.
2. A PTAS for planar graphs.
3. A $(2 - \lg\lg n/2\lg n)$-approximation algorithm for general graphs.
4. A $2k$ kernel for parameterized GVC.

The reader should not be misled into thinking our results imply that Vertex Cover and GVC are in fact the same in any graph class. To see that this is not so, note that while Vertex Cover is polynomial-time solvable in complete graphs, GVC in complete graphs is essentially as hard to approximate as Vertex Cover in any general graph (and thus it cannot be approximated within $10\sqrt{5}-21 \approx 1.36$, unless P=NP [19]). This can be seen by the following reduction from Vertex Cover in general graphs to GVC in complete graphs: Given a graph G, transform G into a complete graph G' by adding all necessary edges, and assign a weight to these edges such that their total weight is substantially smaller than the weight of any vertex in the graph. All original edges are assigned a weight of ∞, and the vertex weights remain the same. It is not difficult to see that any α-approximate vertex cover for G is also an $(\alpha+\epsilon)$-approximate generalized vertex cover of G', for any $\epsilon > 0$ as small as we want, and vice versa.

Our work is also related to the recent work of Könemann *et al.* [20], who presented a reduction from partial covering to prize collecting covering, or in our context, from Partial Vertex Cover to GVC. The Partial Vertex Cover problem is another natural generalization of Vertex Cover, where now the goal is to find a minimum weight subset of vertices that covers a prespecified number of edges in the graph. Könemann *et al.* [20] showed how to transform a specific class of α-approximation algorithms for GVC into an $(\frac{4}{3}\alpha + \epsilon)$-approximation algorithms for Partial Vertex Cover. The algorithm in our Theorem 2 can be used in conjunction with an algorithm from this specific class that works under the assumption that the cost of any generalized vertex cover is at least half of the total weight of the vertices, and so Theorem 2 combined with Könemann *et al.* gives a more refined reduction from PARTIAL VERTEX COVER to GVC.

The rest of the paper is devoted to proving Theorem 2 along with all of its applications mentioned above. In the next section, we discuss some preliminaries necessary for our proof, and in particular we review the local-ratio method which plays an important part in many of our results. In Section 3 we provide all details of the proof of Theorem 2, and in Section 4 we discuss all applications mentioned above. Due to space limitations several proofs are omitted.

2 Preliminaries

In this section we discuss notation and previous work that is necessary for presenting our results. In particular, we introduce terminology that is used for proving Theorem 2, and briefly review the local-ratio technique which we use throughout the paper.

We consider graphs that have weights assigned to their vertices and edges. Let $G = (V, E)$ be a graph given along with a weight function $w : V \cup E \to \mathbb{Q}^{\geq 0}$. For any edge $\{u, v\} \in E$, we write $w(u, v)$ as a shorthand for $w(\{u, v\})$. For a subset of vertices $S \subseteq V$, we let $w(S) = \sum_{v \in S} w(v)$, and for a pair of subsets $S_1, S_2 \subseteq V$, we use $w(S_1, S_2)$ for the weight of edges with one end-point in S_1 and one end-point in S_2 (edges whose endpoints are in $S_1 \cap S_2$ are counted only once), namely $w(S_1, S_2) = \sum_{u \in S_1} \sum_{v \in S_2} w(u, v) - \sum_{u,v \in S_1 \cap S_2} w(u, v)$. Observe that the cost of a subset $S \subseteq V$ in G is $\text{cost}(S) = w(S) + w(V \setminus S, V \setminus S)$. Let $\text{opt}(G, w)$ denote the cost of the optimal generalized vertex cover in (G, w). For an $\alpha > 0$, we say that S is *α-approximate*, if $\text{cost}(S) \leq \alpha \cdot \text{opt}(G, w)$. Also, we call any subset $S \subseteq V$ *feasible*, if it has cost less then ∞.

The Local-Ratio Technique [7] is central in our proof of Theorem 2, and is also used in its applications. The technique in most part is based on the Local-Ratio Lemma, which in our terms can be stated as follows:

Lemma 1 (Local-Ratio [7]). *Let (G, w_1) and (G, w_2) be two instances of* GVC*, with $G = (V, E)$ a graph and $w_1, w_2 : V \cup E \to \mathbb{Q}^{\geq 0}$ two weight functions. If $S \subseteq V$ is α-approximate both in (G, w_1), and in (G, w_2), then S is also α-approximate in $(G, w_1 + w_2)$.*

The following definition hints on how to select a good weight function.

Definition 1 (-effectiveness [21]). *A weight function w_1 is said to be α-*effective in G*, if the following holds: if a subset of vertices is feasible, then it is also α-approximate with respect to w_1.*

Below we give a variation of the Local Ratio Lemma which uses the notion of α-effectiveness, and is the variation that will actually be used in the paper. Its proof is immediate from the Local-Ratio Lemma and the definition of α-effectiveness, and is left to the reader.

Lemma 2. *Let (G, w) be an instance of* GVC*, and let w_ε be a weight function which is α-effective in G. If S is a β-approximate solution for $(G, w - w_\varepsilon)$, then S is a* $\max\{\alpha, \beta\}$*-approximate solution for (G, w).*

3 The Main Proof

In this section we present the central result of this paper, namely the proof of Theorem 2. Our proof consists of two main steps: In the first, similar to the proof of the original Nemhauser&Trotter Theorem, we obtain an initial partition of the vertices of our graph G into three classes according to an optimal solution for an appropriate bipartite graph constructed from G. However, unlike the original proof, in our case we can have edges between all classes, and inside each class. We show that the only really problematic edges are those that are between two particular classes. These edges are taken care of in the second step by several applications of the Local-Ratio Lemma, at the end of which we obtain our desired partition of the vertices of G, and the desired weight function $\widetilde{w}$. Before describing both steps in actual detail, we start with the following lemma which will later be used in our proof, but is also of independent interest.

Lemma 3. GVC *is polynomial-time solvable in bipartite graphs.*

Proof. Let $B = (V, V', F)$ be a bipartite graph, and let $w : V \cup V' \cup F \to \mathbb{Q}^{\geq 0}$ be a weight function. Construct a flow-network N from B by adding to B a source s and a destination t, with s connected to all vertices in V, and vertices in V' connected to t. The capacities of the edges in N are: (i) $c(s, v) = w(v)$ for all $v \in V$, (ii) $c(u, v) = w(u, v)$ for all $\{u, v\} \in F$, and (iii) $c(v, t) = w(v)$ for all $v \in V'$. Observe that there is a one-to-one correspondence between edges in B and s,t-paths in N, and that the edges on an s,t-path in N correspond to the three ways of covering the corresponding edge in B: Either adding one of its endpoints to the generalized vertex cover, or not covering this edge at all. Specifically, given a subset $U \subseteq V \cup V'$ the corresponding s,t-cut is (S, T), where $S = \{s\} \cup (V \setminus U) \cup (V' \cap U)$ and $T = \{t\} \cup (V \cup V') \setminus S$, and conversely, given an s,t-cut (S, T) the corresponding cover is $U = (S \cap V') \cup (T \cap V)$. Hence, there is a one-to-one correspondence between generalized vertex covers in B and s,t-cuts in N. Moreover, by our selection of capacities, each generalized vertex cover corresponds to an s,t-cut whose capacity is equal to the cost of the cover. Since one can compute minimum s,t-cuts by standard flow techniques, the lemma is proven. □

Step I. Given an instance (G, w) for GVC, with $G = (V, E)$ and $w : V \cup E \to \mathbb{Q}^{\geq 0}$, we construct a bipartite graph $B = (V, V', F)$ along with a weight function $w_B : V \cup V' \cup F \to \mathbb{Q}^{\geq 0}$ for B, as follows: The set V' contains a *duplicate vertex* for each vertex in V, and is defined by $V' = \{v' \mid v \in V\}$. The set F of edges in B includes the pair of edges $\{u, v'\}$ and $\{u', v\}$, for each edge $\{u, v\} \in E$. We define w_B by $w_B(v), w_B(v') = w(v)$ for all $v, v' \in V \cup V'$, and $w_B(u', v), w_B(u, v') = w(u, v)$ for all $\{u, v'\}, \{u', v\} \in F$. Here, and throughout the remainder of this section, we denote by S' the set of duplicates of some subset $S \subseteq V$. That is, $S' = \{v' \mid v \in S\}$.

We compute an optimal solution S_B^* in B using the algorithm implied by Lemma 3. According to the computed solution S_B^*, we partition V into the following three subsets: $U_1 = \{v \mid v, v' \in S_B^*\}$, $U_0 = \{v \mid v, v' \notin S_B^*\}$ and $U_{1/2} = V \setminus (U_1 \cup U_0)$. We show that we may assume that $S_B^* \cap U'_{1/2} = \emptyset$.

Claim 1. $\text{cost}_B(U_1 \cup U'_1 \cup U_{1/2}) \leq \text{cost}_B(S_B^*)$.

Proof. By a simple manipulation of the cost of S_B^* in (B, w_B), we get:

$$\begin{aligned}\text{cost}_B(S_B^*) &= w_B(S_B^*) + w_B(U_0 \cup U_{1/2} \setminus S_B^*, U'_0 \cup U'_{1/2} \setminus S_B^*) \\ &\geq w_B(S_B^*) + w_B(U_0, U'_0) + w_B(U_0, U'_{1/2} \setminus S_B^*) + w_B(U'_0, U_{1/2} \setminus S_B^*) \ .\end{aligned}$$

Thus, since $w_B(U_0, U'_{1/2} \setminus S_B^*) + w_B(U'_0, U_{1/2} \setminus S_B^*) = w_B(U_0, U'_{1/2})$, we have $\text{cost}_B(S_B^*) \geq w_B(S_B^*) + w_B(U_0, U'_0) + w_B(U_0, U'_{1/2}) = \text{cost}_B(U_1 \cup U'_1 \cup U_{1/2})$, and the claim is proven. □

Next, using Claim 1, we show that we are on the right direction with our initial partition of the vertices of G, since there is an optimal solution which includes all vertices of U_1 and no vertex of U_0.

Claim 2. There is an optimal solution S for (G, w) with $U_1 \subseteq S$ and $U_0 \cap S = \emptyset$.

Proof. Let S be any subset of vertices in G, and let $S_z = U_z \cap S$ for $z \in \{1, 1/2, 0\}$. Also, let $T = V \setminus S$ and $T_z = U_z \setminus S_z$ for $z \in \{1, 1/2, 0\}$. To prove the claim, we argue that the solution $U_1 \cup S_{1/2}$ does not have greater cost than S. The claim follows by taking S to be optimal.

For this, consider the difference between the cost of S and the cost of $U_1 \cup S_{1/2}$. The only advantage the former has over the latter is that it does not pay for any vertex in T_1, nor for any edge between S_0 and $U_0 \cup T_{1/2}$, all elements which are paid for by $U_1 \cup S_{1/2}$. However, S has to pay for all vertices in S_0, and all edges between T_1 and T, while $U_1 \cup S_{1/2}$ does not. Since this is the only difference between the two solutions, we have $\text{cost}_G(S) - \text{cost}_G(U_1 \cup S_{1/2}) = w(S_0) + w(T_1, T) - w(T_1) - w(S_0, U_0 \cup T_{1/2})$.

Now, we construct a solution S_B for the bipartite graph $B = B(G)$ described above, defined by $S_B = (V \setminus T_0) \cup S'_1$, and we compare this solution to S_B^*. Claim 2 implies that there is no loss of generality in assuming that $S_B^* = U_1 \cup U_{1/2} \cup U'_1$, *i.e.* that S_B^* does not include any vertices of $U'_{1/2}$. Now S_B does not pay for any vertex in T'_1, while S_B^* does, nor does it pay for any edges between S_0 and $U'_0 \cup U'_{1/2}$, all of which are paid for by S_B^*. On the other hand, S_B^* does not pay for any vertex in S_0, nor for any edge between T'_1 and T_0. Noting that this is the exact difference between their costs, and that all weights are positive, we get

$$\begin{aligned}&\text{cost}_B(S_B) - \text{cost}_B(S_B^*) \\ &= w_B(S_0) + w_B(T'_1, T_0) - w_B(T'_1) - w_B(S_0, U'_0 \cup U'_{1/2}) \\ &= w(S_0) + w(T_1, T_0) - w(T_1) - w(S_0, U_0 \cup U_{1/2}) \\ &= w(S_0) + w(T_1, T) - w(T_1, T_1 \cup T_{1/2}) - w(T_1) - w(S_0, U_0 \cup T_{1/2}) - w(S_0, S_{1/2}) \\ &= \text{cost}_G(S) - \text{cost}_G(U_1 \cup S_{1/2}) - w(T_1, T_1 \cup T_{1/2}) - w(S_0, S_{1/2}) \\ &\leq \text{cost}_G(S) - \text{cost}_G(U_1 \cup S_{1/2}) \ .\end{aligned}$$

As S_B^* is optimal in B, we know that $\text{cost}_B(S_B) - \text{cost}_B(S_B^*) \geq 0$. Hence, $\text{cost}(S) - \text{cost}(U_1 \cup S_{1/2}) \geq 0$, and so the claim is proven. □

Claim 2 implies that we can safely restrict ourselves to solutions for (G, w) which include all vertices of U_1, and no vertex of U_0. Therefore, all edges which have at least one endpoint in U_1 are redundant to us in this sense. Also, edges with both endpoints in U_0 are redundant, since we can safely leave these uncovered. The same is not true for edges between U_0 and $U_{1/2}$, as these still might need to be covered. We take care of these edges in the second step of our algorithm, but for now consider the graph H obtained by deleting all edges between vertices of U_0 in the induced subgraph $G[U_0 \cup U_{1/2}]$ of G. Define a weight function w_H for H which equals w on all edges of H and all vertices of $U_{1/2}$ and assigns ∞ to all vertices in U_0. In the following we argue that a good approximation for (H, w_H) gives a good approximation for (G, w).

Claim 3. If S is α-approximate for (H, w_H), then $U_1 \cup S$ is α-approximate for (G, w).

Proof. Let S^*_H denote an optimal solution in (H, w_H), and assume w.l.o.g. that $\text{cost}_H(S^*_H) < \infty$. Then $\text{cost}_H(S) \leq \alpha \cdot \text{cost}_H(S^*_H) \leq \infty$. Now, according to Claim 2, there is an optimal solution for (G, w) which includes all vertices of U_1 and no vertex of U_0, so let S^* be such a solution, with $S^*_{1/2} = S^* \cap U_{1/2}$ and $T^*_{1/2} = U_{1/2} \setminus S^*_{1/2}$. Hence,

$$\begin{aligned}
\text{cost}_G(U_1 \cup S) &= w(U_1) + w(U_0, U_0) + \text{cost}_H(S) \\
&\leq w(U_1) + w(U_0, U_0) + \alpha \cdot \text{cost}_H(S^*_H) \\
&\leq \alpha \cdot (w(U_1) + w(U_0, U_0) + \text{cost}_H(S^*_{1/2})) \\
&= \alpha \cdot (w(U_1) + w(U_0, U_0) + w_H(S^*_{1/2}) + w_H(U_0 \cup T^*_{1/2}, U_0 \cup T^*_{1/2})) \\
&= \alpha \cdot (w(U_1) + w(U_0, U_0) + w(S^*_{1/2}) + w(U_0 \cup T^*_{1/2}, U_0 \cup T^*_{1/2})) \\
&= \alpha \cdot \text{cost}_G(S^*) \ ,
\end{aligned}$$

and the claim is proven. □

Furthermore, we show that the total weight of elements in H with finite weight is at most twice the cost of any solution of (H, w_H):

Claim 4. $w_H(U_{1/2}) + w_H(U_0, U_{1/2}) \leq 2 \cdot \text{cost}_H(S)$ for every $S \subseteq U_0 \cup U_{1/2}$.

Proof. If $S \nsubseteq U_{1/2}$, then $\text{cost}_H(S) = \infty$, and the claim is trivial. Assume therefore that $S \subseteq U_{1/2}$, and denote $T = U_{1/2} \setminus S$. Consider the solution $S_B = U_1 \cup U'_1 \cup S \cup S'$ for the bipartite graph B constructed above. The cost of this solution is:

$$\begin{aligned}
\text{cost}_B(S_B) &= w_B(U_1 \cup U'_1) + w_B(S \cup S') + w_B(U_0 \cup T, U'_0 \cup T') \\
&= 2 \cdot w(U_1) + 2 \cdot w(S) + w(U_0, U_0) + w(T, T) + 2 \cdot w(U_0, T) \\
&= 2 \cdot w(U_1) + 2 \cdot \text{cost}_H(S) + w(U_0, U_0) - w(T, T) \ .
\end{aligned}$$

Now let us compare this solution to S^*_B, the optimal solution of B. Recall that we can assume that $U_{1/2} \subseteq S^*_B$ and $S^*_B \cap U'_{1/2} = \emptyset$. The cost of S^*_B equals the total weight of its vertices plus the total weight of all edges between U_0 and $U'_0 \cup U'_{1/2}$. We have, $\text{cost}_B(S^*_B) = 2 \cdot w(U_1) + w(U_{1/2}) + w(U_0, U_0) + w(U_0, U_{1/2})$. The claim can now be easily proven by combining the two equalities above with the fact that $\text{cost}_B(S^*_B) \leq \text{cost}_B(S_B)$. □

Step II. Note that while the instance (H, w_H) is close to what we aimed to achieve, it is not quite there. One reason is that H is not an induced subgraph of G, it contains vertices of U_0 without the edges between them. Another reason is that the U_0 vertices have w_H-weight equal to ∞, and therefore any feasible solution for (H, w_H) will not satisfy the second condition of Theorem 2. We cannot simply discard these U_0 vertices, since some of them might be connected to vertices in $U_{1/2}$. For this reason, we apply the Local-Ratio lemma to eliminate edges between U_0 and $U_{1/2}$. This is done by applying the following procedure that iteratively subtracts 1-effective weight functions from w_H, in order to obtain the weight function $\widetilde{w}$ promised by Theorem 2:

```
While there is an edge $e_0 = \{u, v\}$ in $H$ with $u \in U_0$, $v \in U_{1/2}$, and
$w_H(e_0), w_H(v) > 0$, do:
a. Let $\varepsilon = \min\{w_H(e_0), w_H(v)\}$.
b. Define the weight function $w_\varepsilon$ for $H$ by:
   – $w_\varepsilon(v), w_\varepsilon(e_0) = \varepsilon$, and
   – $w_\varepsilon(x), w_\varepsilon(e) = 0$ for all $x \neq v$ and $e \neq e_0$.
c. $w_H = w_H - w_\varepsilon$.
```

The above procedure terminates in polynomial time, since at each iteration, either a vertex or an edge get their w_H-weight reduced to zero. The weight function $\widetilde{w}$ is defined to be w_H at the end of the procedure. We define the partition of the vertices in G which is promised in Theorem 2 using $\widetilde{w}$: $V_1 = U_1 \cup \{v \in U_{1/2} \mid \widetilde{w}(v) = 0\}$, $V_0 = U_0$ and $V_{1/2} = V \setminus (V_1 \cup V_0)$.

To complete the proof, we argue that a good approximation for $(G[V_{1/2}], \widetilde{w})$ gives a good approximation for (H, w_H), and that $w(V_{1/2})$ is at most twice the cost of any solution for $(G[V_{1/2}], \widetilde{w})$.

Claim 5. If S is α-approximate for $(G[V_{1/2}], \widetilde{w})$ then $S \cup \{v \mid \widetilde{w}(v) = 0\}$ is α-approximate for (H, w_H).

Proof. We prove the claim using induction on the number of steps applied in this procedure. According to Lemma 2, it suffices to show that any weight function w_ε subtracted in the procedure above is 1-effective. But this is immediate since any solution with cost less than ∞ in (H, w_ε) has cost exactly ε: It either pays for not covering the edge e_0 or for its endpoint in $U_{1/2}$, but never for both. Finally, since $G[V_{1/2}]$ is obtained by removing vertices from H with either 0 or ∞ $\widetilde{w}$-weights, an α-approximation for $(G[V_{1/2}], \widetilde{w})$ implies an α-approximation for $(H, \widetilde{w})$. □

Claim 6. $\widetilde{w}(V_{1/2}) \leq 2 \cdot \text{cost}_{G[V_{1/2}]}(S)$ for every $S \subseteq V_{1/2}$.

Proof. By Claim 4, we have $w_H(U_{1/2}) + w_H(U_0, U_{1/2}) \leq 2\text{cost}_H(S)$ for any $S \subseteq U_0 \cup U_{1/2}$. Since at each iteration in the procedure above, we subtract exactly 2ε from each side of this inequality, at the end of which $G[V_{1/2}]$ includes all positive weighted vertices of H, we get that $\widetilde{w}(V_{1/2}) \leq \widetilde{w}(U_{1/2}) + \widetilde{w}(U_0, U_{1/2}) \leq 2\text{cost}_{G[V_{1/2}]}(S)$. □

The partition V_1, V_0 and $V_{1/2}$, along with the weight function $\widetilde{w}$, satisfy both conditions of Theorem 2. Combining Claim 5 with Claim 3 proves the first condition, while the second condition follows directly from Claim 6.

4 Applications

As mentioned in Section 1, the Nemhauser&Trotter Theorem has several applications in designing approximation algorithms for Vertex Cover. We show that several of these extend to GVC using Theorem 2. The section is divided into four parts, with each part giving a different corollary of Theorem 2. We start with a $(2 - \frac{2}{d})$-approximation algorithm for graphs of bounded degree d, then continue to show a PTAS for planar graphs, and a $(2 - \frac{\lg \lg n}{2 \lg n})$-approximation algorithm for general graphs. Finally, we show that Theorem 2 gives a linear kernel for parameterized GVC.

4.1 Bounded Degree Graphs

Our first application for Theorem 2 is a $(2-\frac{2}{d})$-approximation algorithm for GVC in graphs of bounded degree d. This is an analogous result to an algorithm of Hochbaum [6] that applies the original Nemhauser&Trotter Theorem to obtain the same approximation ratio for Vertex Cover in graphs of bounded degree d. In her algorithm, Hochbaum uses a classical graph-theoretic result by Brooks [22] which states that any graph of bounded degree d which is not complete nor an odd cycle can be properly colored in d colors. (That is, its vertex set can be partitioned into d classes, with no edges between any pair of vertices in the same class.) Together with the Nemhauser&Trotter Theorem, this is basically all that is necessary for Hochbaum's algorithm. Indeed, in our case it is also all that is necessary, due to the following lemma:

Lemma 4. GVC *is polynomial-time solvable in cycles.*

Corollary 1. d-GVC *is approximable within* $2 - \frac{2}{d}$, *for any* $d > 1$.

4.2 Planar Graphs

We next use the technique of Baker [9] together with Theorem 2 to obtain a PTAS for GVC in planar graphs. The main idea is to first use the algorithm of Theorem 2, and then to break the planar subgraph $G[V_{1/2}]$ into a set of k-outerplanar graphs, by removing a set of vertices from G whose weight is at most $\widetilde{w}(V_{1/2})/k$. Since k-outerplanar graphs have treewidth depending only on k, we can compute an optimal solution for each graph in the set of remaining k-outerplanar graphs. (See [23] for the definition of tree width.) Furthermore, since $\widetilde{w}(V_{1/2})$ is at most twice the cost of the optimum solution in $(G[V_{1/2}], \widetilde{w})$ by Theorem 2, this removed set of vertices together with optimal solutions of the k-outerplanar graphs constitute a $(1 + \frac{2}{k})$-approximate generalized vertex cover.

We solve GVC in graphs with bounded treewidth using a standard bottom-up dynamic programming approach.

Lemma 5. GVC *can be solved in* $2^{O(w)} \cdot n$ *time in graphs of treewidth at most* w.

Corollary 2. GVC *in planar graphs has a PTAS.*

4.3 General Graphs

We now show that the $(2-\frac{\lg\lg n}{2\lg n})$-approximation algorithm of [7] can be extended to GVC, due to Theorem 2. The central component in the algorithm of [7] is given in the following lemma:

Lemma 6 ([7]). *There is a polynomial-time algorithm that given a graph* $G = (V, E)$*, a weight function* $w : V \to \mathbb{Q}^{\geq 0}$*, and an integer* k*, such that (i)* $|V| \geq (2k-1)^k$*, and (ii)* G *does not contain an odd cycle of length at most* $2k-1$*, computes a vertex cover* C *of* G *with* $w(C) \leq (1-\frac{1}{2k})w(V)$*.*

Another component we use is given due to the local-ratio technique: We can remove odd cycles in a given instance (G, w) of GVC at a relatively small cost to our approximation guarantee. We have the following lemma:

Lemma 7. *Let* (G, w) *be an instance of* GVC*, and let* C *be an odd cycle in* G *of size* $2t-1$*, where* $t \leq k$*. Then, the weight function* w_ϵ *which assigns* ϵ *to all vertices and edges in* C*, and 0 to all other vertices and edges* $(2-\frac{1}{k})$*-effective.*

Corollary 3. GVC *is approximable within* $2-\frac{\lg\lg n}{2\lg n}$.

4.4 Fixed-Parameter Tractability

The Nemhauser&Trotter Theorem has applications outside the world of approximation algorithms, most notably in the world of parameterized complexity. Chen *et al.* [10] observed that this theorem gives a $2k$ kernel for unweighted parameterized Vertex Cover problem, when the parameter is the total weight of the required vertex cover. We next note that, using Theorem 2, this straightforwardly extends to GVC parameterized by the cost of the optimal solution.

Parameterized complexity deals with parameterized problems, whose instances are given together with a numeric *parameter* k that encodes various structural properties of the input, *e.g.* solution size, maximum degree, and so forth. This allows a refined definition of tractable problems, where a tractable problem is now one with an algorithm running in $f(k)poly(n)$ time, where n is the instance size and f is any computable function. FPT is the class of all parameterized problems with an $f(k)poly(n)$ algorithm. A *kernelization algorithm*, or simply a *kernel*, is a commonly used technique for showing that a parameterized problem is in FPT. Formally, a kernel is a polynomial-time algorithm that transforms an instance (I, k) to an instance (I', k'), with $|I'| + k' \leq f(k)$ for some computable function f, and such that (I, k) is a "yes"-instance if and only if (I', k') is a "yes"-instance. It is easy to see that Theorem 2 gives exactly this, when the parameter is taken as the cost of the solution.

Corollary 4. GVC *parameterized by the cost* k *of the optimal solution has a* $2k$ *kernel.*

References

1. König, D.: Graphok és matrixok (Hungarian; Graphs and matrices). Matematikai és Fizikai Lapok 38, 116–119 (1931)
2. Karp, R.M.: Reducibility among combinatorial problems. Complexity of Computer Computations, 85–103 (1972)
3. Nemhauser, G.L., Trotter Jr., L.E.: Properties of vertex packing and independence system polyhedra. Math. Prog. 6, 48–61 (1974)
4. Hochbaum, D.S.: Approximation algorithms for the set covering and vertex cover problems. SIAM J. Comp. 11(3), 555–556 (1982)
5. Nemhauser, G.L., Trotter Jr., L.E.: Vertex packings: Structural properties and algorithms. Math. Prog. 8(2), 232–248 (1975)
6. Hochbaum, D.S.: Efficient bounds for the stable set, vertex cover and set packing problems. Discrete Applied Mathematics 6, 243–254 (1983)
7. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. Annals of Discrete Mathematics 25, 27–46 (1985)
8. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Applied Math. 36(2), 177–189 (1979)
9. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. ACM 41(1), 153–180 (1994)
10. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: Further observations and further improvements. J. Alg. 41(2), 280–301 (2001)
11. Chlebík, M., Chlebíková, J.: Improvement of Nemhauser-Trotter Theorem and its applications in parametrized complexity. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 174–186. Springer, Heidelberg (2004)
12. Hochbaum, D.S.: Solving integer programs over monotone inequalities in three variables: a framework of half integrality and good approximations. European Journal of Operational Research 140(2), 291–321 (2002)
13. Bar-Yehuda, R., Rawitz, D.: On the equivalence between the primal-dual schema and the local ratio technique. SIAM J. Disc. Math. 19(3), 762–797 (2005)
14. Hassin, R., Levin, A.: The minimum generalized vertex cover problem. ACM Trans. Alg. 2(1), 66–78 (2006)
15. Hassin, R., Tamir, A.: Improved complexity bounds for location problems on the real line. Operations Research Letters 10, 395–402 (1991)
16. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM J. Comp. 24(2), 296–317 (1995)
17. Charikar, M., Khuller, S., Mount, D.M., Narasimhan, G.: Algorithms for facility location problems with outliers. In: 12th SODA, pp. 642–651 (2001)
18. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual-fitting with factor-revealing LP. J. ACM 50(6), 795–824 (2003)
19. Dinur, I., Safra, S.: The importance of being biased. In: 34th ACM Symposium on the Theory of Computing, pp. 33–42 (2002)
20. Könemann, J., Parekh, O., Segev, D.: A unified approach to approximating partial covering problems. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 468–479. Springer, Heidelberg (2006)
21. Bar-Yehuda, R.: One for the price of two: A unified approach for approximating covering problems. Algorithmica 27(2), 131–144 (2000)
22. Brooks, R.L.: On colouring the nodes of a network. Mathematical Proceedings of the Cambridge Philosophical Society 37, 194–197 (1941)
23. Robertson, N., Seymour, P.D.: Graph minors. ii. algorithmic aspects of tree-width. J. Alg. 7(3), 309–322 (1986)

Price Fluctuations: To Buy or to Rent[⋆]

Marcin Bienkowski

Institute of Computer Science, University of Wroclaw, Poland

Abstract. We extend the classic online ski rental problem, so that the rental price may change over time. We consider several models which differ in the knowledge given to the algorithm: whereas the price development is unknown, an algorithm may have full, partial or no knowledge about the duration of the game. We construct algorithms whose competitive ratios are up to constant or logarithmic factors optimal.

Keywords: Online algorithms, competitive analysis, ski rental problem, average-case competitiveness.

1 Introduction

In the classic ski rental problem, a skier may rent skis for p dollars per day or buy them for $s \cdot p$ dollars, where s is an integer greater than one. At the end of any day, the skier may break his legs along with the skis, or in some other way irrevocably finish skiing. The goal is to develop an online strategy minimizing the cost of skiing; this cost is compared to the cost of an optimal offline strategy for the same input. The worst-case ratio between these two amounts is called competitive ratio.

A well-known result [11,12] states that the best deterministic online strategy is to rent skis for $s - 1$ day and then to buy them on day s. It is easy to check that such a strategy achieves a competitive ratio of $(2 - 1/s)$.

In this paper, we extend this model, so that the rental price p may evolve with time. Therefore, the instance of the problem includes not only the duration of the process in days, but also a sequence of prices in consecutive days. However, if we do not impose any constraints on the way the price changes, no algorithm may achieve a competitive ratio better than $\Omega(s)$ (even if the process is guaranteed to last infinitely). To see this, assume that $p = 1$ at day one. If the skier buys skis at the first day, then all the future prices are set to $1/s$, otherwise they are set to s. The optimal solution in these cases is to buy at the second day or at the first one, respectively, and the competitive ratio is $\Omega(s)$. Note also that the ratio $\mathcal{O}(s)$ is achieved by a trivial algorithm which buys at the first day. We view the whole process as a game between an algorithm (the skier) and an adversary (breaking-legs and fixing-prices reality).

Hence, we assume that the rate of price changes is bounded, i.e., the price for renting skis at any day is at least 1 and the prices in two consecutive days differ

[⋆] Supported by MNISW grants number N N206 1723 33, 2007–2010 and N N206 257335, 2008–2011.

at most by 1. Apparently, if we use the standard competitive analysis [1], the cost of any online algorithm, which knows neither the future prices nor the game duration, is rather high in comparison to the optimal offline solution. Therefore, we investigate other scenarios, in which the algorithm has full or partial knowledge about the game duration. We want to emphasize that in all cases, the algorithm does not know the future prices. In particular, we consider a hybrid scenario in which the duration of the game is a random variable, corresponding to the following natural model: each day a skier quits with probability $1/\Gamma$ and continues skiing with probability $1 - 1/\Gamma$. In effect, an input sequence in such a scenario is generated partially by an adversary and partially by a stochastic process, and we measure the performance of an algorithm by a mixture of the average-case and competitive analysis.

In this paper, we consider a continuous variant of the problem. Namely, the prices are given as a continuous curve satisfying the Lipschitz condition and the skier is renting skis up to time T when he buys them (T is possibly a non-integer). This hardly changes the problem; the calculations in the continuous model are simpler though. Moreover, we show that our algorithms can be modified to work in the discrete model without increasing their competitive ratios.

1.1 Problem Formulation and Our Results

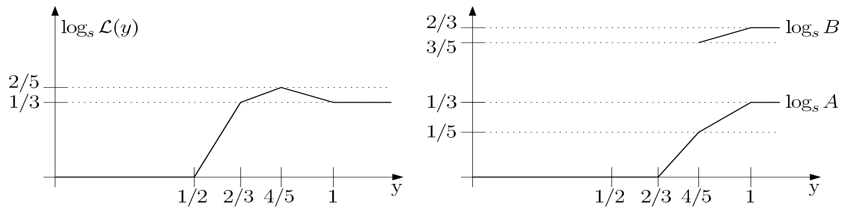
A parameter of the problem is a real number $s > 1$, representing the ratio between the cost of buying skis and renting them.

We assume that the input instance is a pair of an infinite *curve of prices* and a *game duration*. The former is a continuous function $p_t \geq 1$ satisfying the Lipschitz condition, i.e., for any two times t_0 and t_1, it holds that $|p_{t_1} - p_{t_0}| \leq |t_1 - t_0|$. The latter is a real positive number γ.

In online analysis, the input is revealed to an algorithm element by element: in the discrete counterpart of the problem, the algorithm would be given prices in consecutive days and may react after reading any of them. We would like to formulate algorithms in a similar manner, i.e., by saying, for example, that "the algorithm waits for the price p to reach a given threshold". While we use such phrases in the paper, we note that they can be justified without having the algorithm to process infinite number of input elements. For example, the algorithm may choose a small *probing frequency* $\varepsilon > 0$, read the price curve at times $0, \varepsilon, 2\varepsilon, 3\varepsilon \ldots$ and, at these times, make a decision whether to buy skis. The last value presented to the algorithm is then the price at time $\lfloor \frac{\gamma}{\varepsilon} \rfloor \cdot \varepsilon$. As the price curve satisfies the Lipschitz condition, for a small value of ε this read model gives the algorithm essentially the same power as the model in which it is possible to buy skis at any time.

Fix any input curve p_t and the game duration γ. For any T, let $\mathcal{F}(T) = \int_{t=0}^{T} p_t \, dt$ and let $\textsc{Buy}(T)$ be an algorithm which buys at time T. In particular, $\textsc{Buy}(\infty)$ is an algorithm which always rents. Then its cost is equal to

$$C_{\mathrm{BUY(T)}} = \begin{cases} \mathcal{F}(T) + s \cdot p_T & \text{if } T \leq \gamma \ , \\ \mathcal{F}(\gamma) & \text{if } T > \gamma \ . \end{cases}$$



y	$(-\infty, 1/2)$	$[1/2, 2/3)$	$[2/3, 4/5)$	$[4/5, 1)$	$[1, \infty)$
$\mathcal{L}(y)$	1	s^{2y-1}	$s^{y/2}$	$s^{(2-y)/3}$	$s^{1/3}$

Fig. 1. Function $\mathcal{L}(y)$ used in the description of competitive ratios (left), thresholds A and B used by the algorithm PROT_{s^y} (right)

The cost of the optimal offline solution, OPT, is $C_{\text{OPT}} = \min_{0 \le T \le \infty} C_{\text{BUY(T)}}$.

For any deterministic algorithm ALG, we say that ALG is $\mathcal{R}$-competitive if for any input it holds that $C_{\text{ALG}}/C_{\text{OPT}} \le \mathcal{R}$. If the algorithm is randomized, we replace C_{ALG} by its expected value. For randomized algorithms, we assume oblivious adversaries, which do not see random bits used by the algorithm.

We consider several scenarios, which differ in the knowledge given to the algorithm.

Scenario A: Unknown game end. This is the most straightforward case, in which the adversary dictates both price curve p_t and game duration γ, and an algorithm learns γ at the end of the game. For this scenario, in Sect. 2, we give a deterministic $\mathcal{O}(\sqrt{s})$-competitive algorithm. This ratio is asymptotically optimal, as we present an up to a constant factor matching lower bound, which holds even for randomized algorithms.

Scenario B: Known game end. In this scenario, the adversary defines both a price curve and duration γ, but the algorithm learns γ at the very beginning and can adjust its behavior accordingly. For describing our results, we define a function $\mathcal{L}$ as in Fig. 1. In Sect. 3, we construct an algorithm PROT which is $\mathcal{O}(\mathcal{L}(\log_s \gamma))$-competitive for Scenario B. This means that for the worst possible choice of γ, we achieve a competitive ratio of $\mathcal{O}(s^{2/5})$. We provide a matching lower bound for any value of γ and any randomized algorithm in Sect. 4.

Scenario C: Stochastic game end. In this scenario, the adversary defines a price curve and a parameter $\Gamma \ge 1$, which is revealed to the algorithm at the very beginning. Then the game duration γ is chosen randomly according to the exponential distribution with parameter Γ. The algorithm learns the game end at time γ. As mentioned above, this is a continuous counterpart of the following natural model: the probability of breaking a leg is $1/\Gamma$ each day, i.e., the expected duration is Γ. Such modeling of the input builds another bridge between the average-case and competitive analysis. In Sect. 3.3, we show that if we run an appropriately parameterized version of PROT, the expected value of its competitive ratio is $\mathcal{O}(\mathcal{L}(\log_s \Gamma) \cdot (\log s)^{7/9})$. We note that asymptotically the same results hold if we consider uniform distribution over interval $[0, 2\Gamma]$, i.e., with the same expected duration Γ.

Finally, we show that our algorithms for the continuous model can be adjusted to work in the discrete model without increasing their competitive ratios. Due to lack of space, some proofs will appear in the full version of the paper.

1.2 Related Work

For the classic ski rental problem (SRP), the competitive ratio of the best deterministic online algorithm is $2 - 1/s$ [11,12]. If we allow randomization, then this ratio can be reduced to $\mathrm{e}/(\mathrm{e}-1) \approx 1.58$ [10]. Similar constructions achieving optimal ratios (2 for the deterministic case and $\mathrm{e}/(\mathrm{e}-1)$ for the randomized one) were shown for related rent-or-buy problems like the Bahncard problem [6,9], the spin-block problem [10], or the TCP acknowledgement problem [3,9].

In the *multi-slope* SRP [2,13], the buyer may switch between many lease options (r_i, w_i), where r_i is the startup cost for using this option and w_i is the fee paid each day. In these terms, the classic SRP means a binary choice between the pure rental option $(0, p)$ and the pure purchase option $(s \cdot p, 0)$.

The SRP was also analyzed in an average-case: Fujiwara and Iwama [7] considered the case where the game duration is determined by a stochastic process and the goal is to minimize the expected value of the competitive ratio. In particular, they proved that if the duration is an exponentially distributed (with parameter Γ) random variable, the optimal strategies are either to rent forever if $\Gamma \leq s$ or to buy skis at day s^2/Γ if $\Gamma > s$.

For other successful applications of the competitive analysis where an input is generated randomly, see, e.g., [8] and the references therein. However, in contrast to these papers, we consider a model in which the input is not chosen purely randomly, but partially randomly and partially by an adversary. Our performance metric for the third scenario (the expected value of the competitive ratio) is thus similar in flavor to the smoothed competitive ratio [14].

To our best knowledge, the SRP was not analyzed in a model in which prices may change over time. This problem is loosely related to various versions of currency trading (see, e.g., [5,4]). Due to different focus and assumptions, algorithmic ideas developed there do not seem to apply to our problem.

2 Unknown Game End

In this section, we present a simple deterministic algorithm Thresh for Scenario A. In period $[0, \sqrt{s})$, Thresh always rents. Then, it buys skis at the first time $k \geq \sqrt{s}$, such that $\mathcal{F}(k) \geq s$.

Theorem 1. Thresh *is $\mathcal{O}(\sqrt{s})$-competitive for Scenario A.*

Proof. Let γ be the duration of the game. Let k be the time of skis purchase or $k = \gamma$ if Thresh does not buy skis. To show the theorem, we relate the costs of Thresh and Opt to $\mathcal{F}(k)$, proving the following two relations.

1. $C_{\text{THRESH}} = \mathcal{O}(\sqrt{s}) \cdot \mathcal{F}(k)$.
   If THRESH does not buy skis, then $k = \gamma$ and the relation holds trivially. Otherwise, $C_{\text{THRESH}} = \mathcal{F}(k) + s \cdot p_k$. By the definition of the algorithm, $\mathcal{F}(k) \geq s$ and $k \geq \sqrt{s}$. If $p_k \leq 2 \cdot \sqrt{s}$, then $\mathcal{F}(k) \geq \sqrt{s} \cdot p_k/2$. If $p_k > 2 \cdot \sqrt{s}$, then in period $[k - \sqrt{s}, k]$ the price is at least $p_k - \sqrt{s} > p_k/2$, and thus $\mathcal{F}(k) > \sqrt{s} \cdot p_k/2$ as well.
2. $C_{\text{OPT}} \geq \mathcal{F}(k)/2$.
   This relation clearly holds if OPT rents forever or buys skis at time k or later. Thus, we assume that OPT buys skis at time $\ell < k$, paying at least $s \cdot p_\ell$. We consider two cases.
   (a) $k > \sqrt{s}$. By the construction of THRESH, $\mathcal{F}(\sqrt{s}) < s$, and thus $\mathcal{F}(k) \leq s \leq C_{\text{OPT}}$. (The first inequality is strict only in case $k = \gamma$.)
   (b) $k \leq \sqrt{s}$. As any two prices in period $[0, k]$ can differ at most by k, we obtain $\mathcal{F}(k) = \int_0^k p_t \, dt \leq \int_0^k (p_\ell + k) \, dt = k \cdot p_\ell + k^2 \leq 2 \cdot s \cdot p_\ell \leq 2 \cdot C_{\text{OPT}}$. □

It appears that the achieved competitive ratio is asymptotically optimal, even for randomized algorithms.

Theorem 2. *In Scenario A, the competitive ratio of any randomized algorithm is $\Omega(\sqrt{s})$.*

3 Algorithm PROT

In this section, we construct an algorithm PROT_λ which we further analyze in Scenarios B and C; λ is a parameter of the algorithm. The algorithm performs well if $\lambda = \gamma$. This can be guaranteed in Scenario B; in Scenario C, we know only that $\mathbf{E}[\gamma] = \Gamma$, and hence we choose λ close to Γ.

For any values of λ and γ, let $\mathcal{R}(\lambda, \gamma)$ be the competitive ratio of PROT_λ run on a sequence of length γ. Then, the actual competitive ratio of PROT in Scenario B is $\mathcal{R}(\gamma, \gamma)$ and the expected value of the competitive ratio in Scenario C is $\mathbf{E}_\gamma[\mathcal{R}(\lambda, \gamma)]$.

Description of the Algorithm. The behavior of algorithm PROT_λ depends on the parameter λ. We think of λ as the algorithm's estimate of the game duration γ. As in the classic ski rental problem, we want to amortize the cost of skis purchase against the cost of their rental, e.g., to buy if we already rented for time $\Theta(s)$. For example, if λ is small, it makes little sense to buy skis. However, the fluctuation of the prices may trigger the purchase also at some other points of the time.

We give an informal rationale for algorithm PROT_s run on an input of length s. The algorithm has to protect itself against two different types of inputs. An input may contain a time point with a low price, giving OPT the possibility to buy for this price. To cope with such inputs, PROT_s chooses a threshold A and buys skis when the price falls below A. The second threshold is less intuitive. Assume that the price grows for the whole input. In this case, the optimal solution is to buy as early as possible. On the other hand, the total cost of renting skis grows quadratically with time (at least from a certain point). To protect against such inputs, $\textsc{Prot}_s$ uses the second threshold B; it buys skis when the price goes above B.

Although it is not necessary in Scenario B, the algorithm never buys skis (even if thresholds are reached) in period $[0, 1)$. The reason for this stems from Scenario C: in this case, if we just used thresholds A and B, the adversary could trigger the skis purchase of $\textsc{Prot}$ at time 0, i.e., $C_{\textsc{Prot}} = s \cdot p_0$. For any Γ, it can happen (with non-negligible probability) that the sequence ends at time $\varepsilon \ll 1$.[1] In this case, $C_{\textsc{Opt}} \approx \varepsilon \cdot p_0$, and the competitive ratio is very high.

Taking these intuitions into account, we formally define the algorithm. $\textsc{Prot}_\lambda$ uses two thresholds: A and B, which are depicted in Fig. 1 and defined as follows.

$\lambda = s^y$	$[0, s^{2/3})$	$[s^{2/3}, s^{4/5})$	$[s^{4/5}, s)$	$[s, \infty)$
A	$1/2$	$s^{(3/2)y-1}$	$s^{(2y-1)/3}$	$s^{1/3}$
B	∞	∞	$s^{(y+1)/3}$	$s^{2/3}$

If at time $t \geq 1$ price falls outside range (A, B), then $\textsc{Prot}_\lambda$ buys at this price. Otherwise, if the price remains in range (A, B) for the period $[1, s]$, then $\textsc{Prot}_\lambda$ buys at time s.

Note that as all the prices are at least 1, for $\lambda \leq s^{2/3}$, the strategy of $\textsc{Prot}_\lambda$ is just to rent for the whole period $[0, s)$ and to buy at time s (if the game lasts till this time).

Observation 1. *If the game ends at time $\gamma < 1$, then $\mathcal{R}(\lambda, \gamma) = \mathcal{O}(1)$.*

Therefore, in the remaining part of this section, we assume that $\gamma \geq 1$. For making our arguments concise, we consider a restricted version of $\textsc{Opt}$ which is not allowed to buy in period $[0, 1)$. Such a restriction increases its cost at most by a constant factor, and can be neglected as we are interested in the asymptotic performance.

3.1 Types of Sequences

We start with a classification of input curves and we characterize the behavior of $\textsc{Prot}$ on them.

1. An input is of type (a, b)-middle if all prices in period $[1, s]$ are in range (a, b).
2. An input is of type (a, b)-low with parameter $k < s$ if all prices in period $[1, k)$ are within range (a, b) and $p_k \leq a$.
3. An input is of type (a, b)-high with parameter $k < s$ if all prices in period $[1, k)$ are within range (a, b) and $p_k \geq b$.

Note that for any a and b, each input is either (a, b)-low, (a, b)-middle, or (a, b)-high. Moreover, by the continuity of the price curve, if an input is (a, b)-low

[1] Note that such situation cannot occur in the discrete variant of the problem.

with parameter k, either $k = 1$ and $p_1 \leq a$, or $k \geq 1$ and $p_k = a$. An analogous relation holds for (a, b)-high sequences.

Throughout this section, we assume that $\gamma \geq 1$ is the duration of the game, and $A < B$ are the thresholds used by PROT_λ. We present three lemmas describing the algorithm behavior on the corresponding three types of input sequences. For succinctness, we define

$$M_{A,B,\gamma} = \min \left\{ \frac{B}{A},\ 1 + \frac{\gamma^2}{s \cdot A} \right\} \ . \tag{1}$$

In the following three lemmas, we characterize the behavior of PROT on different types of sequences. Proofs of the second and third ones are relatively similar, and hence we omit one of them.

Lemma 1. *On an (A, B)-middle input, it holds that $\mathcal{R}(\lambda, \gamma) = \mathcal{O}(M_{A,B,\gamma})$.*

Proof. We note that if $\gamma > s$, we may set $\gamma = s$, as the behavior of the algorithm remains unchanged and cost of OPT may only decrease. Thus, we assume $\gamma \leq s$.

If $\gamma = s$, then PROT buys skis at time s, paying at most $s \cdot B$ for renting and $s \cdot B$ for purchase. Further, OPT $\geq s \cdot A$, as OPT either buys or rents for the entire period. Hence, $C_{\text{PROT}}/C_{\text{OPT}} \leq 2B/A$.

If $\gamma < s$, PROT always rents, paying $\mathcal{F}(\gamma)$. On the other hand, $C_{\text{OPT}} \geq \min\{\mathcal{F}(\gamma), s \cdot A\}$. Thus,

$$\frac{C_{\text{PROT}}}{C_{\text{OPT}}} \leq 1 + \frac{\mathcal{F}(\gamma)}{s \cdot A} \leq 1 + \frac{\gamma \cdot A + \gamma^2/2}{s \cdot A} = \mathcal{O}\left(1 + \frac{\gamma^2}{s \cdot A}\right) \ .$$

□

Lemma 2. *On an (A, B)-low input with parameter k, it holds that $\mathcal{R}(\lambda, \gamma) = \mathcal{O}(A + s \cdot A/\gamma + M_{A,B,\gamma})$.*

Lemma 3. *On an (A, B)-high input with parameter k, it holds that $\mathcal{R}(\lambda, \gamma) = \mathcal{O}\left(B/A + s/B + s/\gamma\right)$.*

Proof. If $\gamma < k$, then the prefix of the input seen by the algorithm is (A, B)-middle, and thus the competitive ratio is at most $\mathcal{O}(M_{A,B,\gamma})$.

Below, we assume that $\gamma \geq k$. By the definition, $C_{\text{PROT}} = \mathcal{F}(k) + s \cdot p_k$. The main complication is that p_k is not necessarily equal to B, as the input sequence may simply start with a very high price.

1. If $k \leq B/2$, then $p_k \geq B$ and during period $[0, k + B/2]$ the price remains in range $[p_k - B/2, p_k + B/2]$. We consider a possibly shorter period $[0, \min\{k + B/2, \gamma\}]$, which actually appears in the game. OPT either rents skis in this period paying at least $\min\{k + B/2, \gamma\} \cdot (p_k - B/2) = \Omega(\min\{B, \gamma\} \cdot p_k)$, or buys skis in this period paying at least $s \cdot (p_k - B/2) = \Omega(s \cdot p_k)$. Thus, the competitive ratio in this case is

$$\frac{C_{\text{PROT}}}{C_{\text{OPT}}} = \mathcal{O}\left(\frac{s \cdot p_k}{\min\{B, \gamma, s\} \cdot p_k}\right) = \mathcal{O}\left(1 + \frac{s}{\gamma} + \frac{s}{B}\right) \ . \tag{2}$$

2. If $k > B/2$, then $p_k = B$. We consider two subcases.
   (a) Opt buys skis in the period $[1, k]$. Then $C_{\text{OPT}} \geq s \cdot A$, $C_{\text{PROT}} \leq k \cdot B + s \cdot B \leq 2 \cdot s \cdot B$, and thus
$$\frac{C_{\text{PROT}}}{C_{\text{OPT}}} = \mathcal{O}(B/A) \ . \tag{3}$$
   (b) Opt rents skis in the period $[1, k]$. Note that during period $[k - B/2, k]$ the price remains above $B/2$. Thus, $C_{\text{OPT}} \geq \mathcal{F}(k)$ and $C_{\text{OPT}} = \Omega(B^2)$. The competitive ratio is then bounded by
$$\frac{C_{\text{PROT}}}{C_{\text{OPT}}} = \mathcal{O}\left(\frac{\mathcal{F}(k) + s \cdot B}{\mathcal{F}(k) + B^2}\right) = \mathcal{O}(s/B) \ . \tag{4}$$

By combining (2), (3), and (4), we obtain the lemma. □

3.2 Algorithm Analysis for Known-End Scenario

If we look at thresholds A and B of Prot_λ, it appears that for particular choices of parameter λ, we can restrict the input types that may appear during Prot_λ execution. For example, if $s^{2/3} \leq \lambda < s^{4/5}$, $B = \infty$, and thus the input can be only (A, B)-low or (A, B)-middle. Applying the results of Lemmas 1, 2, and 3, we immediately get the following theorem.

Theorem 3. *Fix any $\lambda, \gamma \geq 0$, and let $y = \log_s \lambda$. Then*
$$\mathcal{R}(\lambda, \gamma) = \mathcal{O}(1) \cdot \begin{cases} 1 + \gamma^2/s & \text{if } y < 2/3 \ , \\ s^{\frac{3}{2}y-1} + s^{\frac{3}{2}y}/\gamma + \gamma^2/s^{\frac{3}{2}y} & \text{if } 2/3 \leq y < 4/5 \ , \\ (1 + s^y/\gamma) \cdot s^{(2-y)/3} & \text{if } 4/5 \leq y < 1 \ , \\ (1 + s/\gamma) \cdot s^{1/3} & \text{if } y \geq 1 \ . \end{cases}$$

In Scenario B, the algorithm knows the game duration γ and by running Prot_γ, we may achieve the desired competitive ratio.

Corollary 1. *The competitive ratio of Prot_γ on an input of length γ is at most $\mathcal{O}(\mathcal{L}(\log_s \gamma))$.*

3.3 Algorithm Analysis for Stochastic-End Scenario

For Scenario C, we analyze the performance of Prot run on an input, whose length is an exponentially distributed random variable $\gamma \sim \text{Exp}(\Gamma)$. The density function of this distribution is $f(x) = \frac{1}{\Gamma} \cdot e^{-x/\Gamma}$ for $x \geq 0$. The following properties hold.

Observation 2. *If $f(x)$ is the probability density function of the exponential distribution with parameter $\Gamma \geq 1$, then $\int_0^\infty x \cdot f(x)\, dx = \Gamma$, $\int_0^\infty x^2 \cdot f(x)\, dx = \mathcal{O}(\Gamma^2)$, and $\int_1^\infty \frac{1}{x} \cdot f(x)\, dx = \mathcal{O}(\frac{\log \Gamma}{\Gamma})$.*

As mentioned earlier, Prot performs well if λ corresponds to the game duration γ. In Scenario C, this can be fulfilled only in expectation, e.g., by setting $\lambda = \Gamma = \mathbf{E}[\gamma]$.

Theorem 4. *For any $\Gamma \geq 1$, the expected value of competitive ratio of* $\textsc{Prot}_\Gamma$ *run on a sequence of length $\gamma \sim \mathrm{Exp}(\Gamma)$ is $\mathcal{O}(\mathcal{L}(\log_s \Gamma) \cdot \log \Gamma)$.*

Proof. Let f be the density function of the exponential distribution. We want to upper-bound the value of

$$\mathbf{E}_\gamma[\mathcal{R}(\Gamma, \gamma)] = \int_0^\infty \mathcal{R}(\Gamma, \gamma) \cdot f(\gamma)\, d\gamma = \mathcal{O}(1) + \int_1^\infty \mathcal{R}(\Gamma, \gamma) \cdot f(\gamma)\, d\gamma \ ,$$

where the second equality follows by Observation 1. Recall, that by Theorem 3, $\mathcal{R}(\Gamma, \gamma)$ is a sum of terms, where each term is a linear function of γ^c for $c \in \{-1, 1, 2\}$. Therefore, by Observation 2, we obtain

$$\int_1^\infty \mathcal{R}(\Gamma, \gamma) \cdot f(\gamma)\, d\gamma = \mathcal{O}(\mathcal{R}(\Gamma, \Gamma) \cdot \log \Gamma) \ .$$

As $\mathcal{R}(\Gamma, \Gamma) = \mathcal{O}(\mathcal{L}(\log_s \Gamma))$, this finishes the proof. □

The theorem above means that we pay an additional factor of $\log \Gamma$ in the competitive ratio for not knowing the exact end of the game, but only its expected value. In fact, the algorithm pays more because there is a quite big probability that $\gamma \ll \mathbf{E}[\gamma]$. For such cases, the thresholds chosen by Prot are too large, which leads to poorer performance. On the other hand, the effects of underestimating γ are more benign. Hence, it appears that the expected value of the competitive ratio can be improved if we choose slightly smaller value of parameter λ of the algorithm Prot.

Theorem 5. *Fix any $\Gamma \geq 1$ and let $\lambda = \Gamma/(\log s)^{1/3}$. Then the expected value of the competitive ratio of* $\textsc{Prot}_\lambda$ *run on a sequence of length $\gamma \sim \mathrm{Exp}(\Gamma)$ is $\mathcal{O}(\mathcal{L}(\log_s \Gamma) \cdot (\log s)^{7/9})$ for $\Gamma \in (s^{2/3}, s \cdot \log s)$ and $\mathcal{O}(\mathcal{L}(\log_s \Gamma))$ for remaining values of Γ.*

Finally, we observe that since in the proofs of Theorems 4 and 5, we base our reasoning entirely on Observation 2, the same results hold also for all probability distributions for which this observation holds, e.g., for a uniform distribution over the interval $[0, 2\Gamma]$.

4 Lower Bounds for Known-End Scenario

In this section, we show that $\textsc{Prot}_\gamma$ achieves an asymptotically optimal competitive ratio on inputs of length γ. Moreover, we show show that this lower bound holds even for randomized algorithms against an oblivious adversary.

Theorem 6. *Fix any γ. In Scenario B, the competitive ratio of any randomized algorithm (knowing γ) is at least $\Omega(\mathcal{L}(\log_s \gamma))$.*

Proof. Let $y = \log_s \gamma$, i.e., $\gamma = s^y$. For $y < 1/2$, the lower bound of $\Omega(\mathcal{L}(y)) = \Omega(1)$ follows trivially, thus we assume that $y \geq 1/2$.

To show the theorem, we use Yao min-max principle [15]. Let $f = \min\{y, 1\}$. Let a be a non-negative number satisfying $a \leq f/2$ and $a \leq 2f - 1$; we will specify the exact value of a later.

We call an input curve α-*peaky* if the price starts at s^a, increases till it achieves level α, then drops again to level 1 and remains there. Formally,

$$p_t = \begin{cases} s^a + t & \text{if } t \leq \alpha - s^a \ , \\ \alpha - (t - \alpha + s^a) & \text{if } t > \alpha - s^a \ . \end{cases}$$

We will consider only curves for which $\alpha \leq \gamma$. This guarantees that an α-peaky curve of length γ achieves its peak; however for large α the decreasing part of the curve may be shortened or not occur.

Note that on an α-peaky curve, the algorithm which always rents pays $\mathcal{O}(\alpha^2 + s^f)$ and the algorithm which buys at the very beginning pays $s \cdot s^a$. Thus, on an α-peaky curve, $C_{\text{OPT}} = \mathcal{O}(\min\{\alpha^2 + s^f, s^{1+a}\})$. On the other hand, on online algorithm does not know α, and cannot choose optimally between these two strategies.

We construct the following probability distribution π over input curves. With probability $1/3$, the input presented to the algorithm is $\frac{1}{2}s^{f/2}$-peaky and with probability $1/3$, it is s^f-peaky. The remaining probability $1/3$ is distributed uniformly over α-peaky inputs for $\alpha \in [\frac{1}{2}s^{f/2}, s^{(1+a)/2}]$. This means that the probability density function is $f(\alpha) = \frac{1}{3} \cdot \left(s^{(1+a)/2} - \frac{1}{2}s^{f/2}\right)^{-1} = \Theta(s^{-(1+a)/2})$ for $\alpha \in \left[\frac{1}{2}s^{f/2}, s^{(1+a)/2}\right]$ and 0 otherwise.

We want to analyze the best deterministic algorithm DET and its performance on an input chosen randomly according to the probability distribution π. Let $C_{\text{DET}}(\alpha)$ and $C_{\text{OPT}}(\alpha)$ be the costs of DET and OPT, respectively, on an α-peaky input.

$$\mathbf{E}_\pi \left[\frac{C_{\text{DET}}}{C_{\text{OPT}}}\right] = \frac{1}{3} \cdot \frac{C_{\text{DET}}(\frac{1}{2}s^{f/2})}{C_{\text{OPT}}(\frac{1}{2}s^{f/2})} + \frac{1}{3} \cdot \frac{C_{\text{DET}}(s^f)}{C_{\text{OPT}}(s^f)} + \int_{\frac{1}{2}s^{f/2}}^{s^{(1+a)/2}} \frac{C_{\text{DET}}(\alpha)}{C_{\text{OPT}}(\alpha)} f(\alpha)\, d\alpha \quad (5)$$

Then, by the Yao min-max principle, the competitive ratio $\mathcal{R}$ of any randomized algorithm against an oblivious adversary is at least $\mathbf{E}_\pi[C_{\text{DET}}/C_{\text{OPT}}]$.

Essentially, DET has the following options. It may opt for the always-rent strategy or it may choose a parameter $r \in [s^a, s^f]$ and buy if the price achieves r. If the price starts to drop before it hits level r, it does not make sense for DET to buy skis in the remaining period, as this cost would be greater then renting skis till time γ. We analyze this behavior in four cases below.

1. DET chooses $r \in [s^a, \frac{1}{2}s^{f/2})$. With probability $1/3$, the input curve is $\frac{1}{2}s^{f/2}$-peaky. For such an input, $C_{\text{DET}} \geq s \cdot s^a$ and $C_{\text{OPT}} = \mathcal{O}((\frac{1}{2}s^{f/2})^2 + s^f) = \mathcal{O}(s^f)$. As all the terms occurring in (5) are non-negative, we may omit some of them, obtaining

$$\mathcal{R} \geq \frac{1}{3} \cdot \frac{C_{\text{DET}}(\frac{1}{2}s^{f/2})}{C_{\text{OPT}}(\frac{1}{2}s^{f/2})} = \Omega\left(s^{1-f+a}\right) \ . \quad (6)$$

2. DET chooses $r \in [\frac{1}{2}s^{f/2}, \frac{1}{2}s^{(1+a)/2})$. For α-peaky inputs with $r \leq \alpha \leq s^{(1+a)/2}$, $C_{\mathrm{DET}}(\alpha) \geq s \cdot r$ and $C_{\mathrm{OPT}}(\alpha) = \mathcal{O}(\alpha^2 + s^f) = \mathcal{O}(\alpha^2)$. Then,

$$\begin{aligned}\mathcal{R} &\geq \int_r^{s^{(1+a)/2}} \frac{C_{\mathrm{DET}}(\alpha)}{C_{\mathrm{OPT}}(\alpha)} f(\alpha)\, d\alpha \\ &= \Theta\left(s^{-(1+a)/2}\right) \cdot \int_r^{s^{(1+a)/2}} \frac{s \cdot r}{\alpha^2}\, d\alpha \\ &= \Theta\left(s^{-(1+a)/2}\right) \cdot \left(\frac{s \cdot r}{r} - \frac{s \cdot r}{s^{(1+a)/2}}\right) \\ &= \Theta(s^{-(1+a)/2}) \cdot (s - s/2) \\ &= \Theta(s^{(1-a)/2}) \ .\end{aligned} \tag{7}$$

3. DET chooses $r \in [\frac{1}{2}s^{(1+a)/2}, s^f]$. With probability 1/3, the input is s^f-peaky. For such an input, $C_{\mathrm{DET}} \geq s \cdot \frac{1}{2}s^{(1+a)/2}$ and $C_{\mathrm{OPT}} = s \cdot s^a$. In this case,

$$\mathcal{R} \geq \frac{1}{3} \cdot \frac{C_{\mathrm{DET}}(s^f)}{C_{\mathrm{OPT}}(s^f)} = \Omega\left(s^{(1-a)/2}\right) \ . \tag{8}$$

4. DET never buys skis. On an s^f-peaky input, $C_{\mathrm{DET}} = \Omega((s^f)^2)$, and therefore

$$\mathcal{R} \geq \frac{1}{3} \cdot \frac{C_{\mathrm{DET}}(s^f)}{C_{\mathrm{OPT}}(s^f)} = \Omega\left(s^{2f-1-a}\right) \ . \tag{9}$$

For any fixed y and a, DET may freely choose one of the strategies above. Thus, by (6), (7), (8), and (9),

$$\mathcal{R} = \Omega\left(\min\left\{s^{1-f+a}, s^{(1-a)/2}, s^{2f-1-a}\right\}\right) \ .$$

Finally, the adversary, may choose a to maximize the ratio $\mathcal{R}$.

1. For $y \in [1/2, 2/3)$, we choose $a = 0$. In this case $\mathcal{R} = \Omega(s^{2y-1})$.
2. For $y \in [2/3, 4/5)$, we choose $a = (3/2)y - 1$. In this case $\mathcal{R} = \Omega(s^{y/2})$.
3. For $y \in [4/5, 1)$, we choose $a = (2y-1)/3$. In this case $\mathcal{R} = \Omega(s^{(2-y)/3})$.
4. For $y \geq 1$, we choose $a = 1/3$. In this case $\mathcal{R} = \Omega(s^{1/3})$.

Therefore, in all cases $\mathcal{R} = \Omega(\mathcal{L}(y))$, which concludes the proof. □

5 Final Remarks

This paper extends the classic ski rental problem in a way where the rental price may change each day. Although the ski rental may be perceived as a toy example, the underlying rent-or-buy structure can be found in many other problems. We believe that our preliminary work can contribute to developing solutions for these problems when they are analyzed in a dynamic setting. A natural example would be the TCP acknowledgement problem [3], where the varying traffic on the link is modeled by changes in the cost associated with sending acknowledgements. Moreover, if our algorithms can be adapted for such network problems, their simplicity ensures that they could be used in constrained devices.

References

1. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
2. Damaschke, P.: Nearly optimal strategies for special cases of on-line capital investment. Theoretical Computer Science 302(1–3), 35–44 (2003)
3. Dooly, D.R., Goldman, S.A., Scott, S.D.: On-line analysis of the TCP acknowledgment delay problem. Journal of the ACM 48(2), 243–273 (2001); Also appeared as TCP Dynamic Acknowledgment Delay: Theory and Practice. In: Proc. of the 30th STOC, pp. 389–398 (1998)
4. El-Yaniv, R.: Competitive solutions for online financial problems. ACM Computing Surveys 30(1), 28–69 (1998)
5. El-Yaniv, R., Fiat, A., Karp, R.M., Turpin, G.: Competitive analysis of financial games. In: Proc. of the 33rd IEEE Symp. on Foundations of Computer Science (FOCS), pp. 327–333 (1992)
6. Fleischer, R.: On the bahncard problem. In: Hsu, W.-L., Kao, M.-Y. (eds.) COCOON 1998. LNCS, vol. 1449, pp. 12–14. Springer, Heidelberg (1998)
7. Fujiwara, H., Iwama, K.: Average-case competitive analyses for ski-rental problems. In: Bose, P., Morin, P. (eds.) ISAAC 2002. LNCS, vol. 2518, pp. 476–488. Springer, Heidelberg (2002)
8. Garg, N., Gupta, A., Leonardi, S., Sankowski, P.: Stochastic analyses for online combinatorial optimization problems. In: Proc. of the 19th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 942–951 (2008)
9. Karlin, A.R., Kenyon, C., Randall, D.: Dynamic TCP acknowledgement and other stories about e/(e - 1). Algorithmica 36, 209–224 (2003); Also appeared in: Proc. of the 33rd STOC, pp. 502–509 (2001)
10. Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.: Competitive randomized algorithms for non-uniform problems. Algorithmica 11, 542–571 (1994); Also appeared in: Proc. of the 1st SODA, pp. 301–309 (1990)
11. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica 3(1), 77–119 (1988); Also appeared in: Proc. of the 27th FOCS, pp. 244–254 (1986)
12. Karp, R.M.: On-line algorithms versus off-line algorithms: How much is it worth to know the future? In: Proc. of the IFIP 12th World Computer Congress, pp. 416–429 (1992)
13. Lotker, Z., Patt-Shamir, B., Rawitz, D.: Rent, lease or buy: Randomized algorithms for multislope ski rental. In: Proc. of the 25th Symp. on Theoretical Aspects of Computer Science (STACS), pp. 503–514 (2008)
14. Schäfer, G., Sivadasan, N.: Topology matters: Smoothed competitiveness of metrical task systems. Theoretical Computer Science 341(1–3), 216–246 (2005); Also appeared in: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 489–500. Springer, Heidelberg (2004)
15. Yao, A.C.-C.: Probabilistic computation: towards a uniform measure of complexity. In: Proc. of the 18th IEEE Symp. on Foundations of Computer Science (FOCS), pp. 222–227 (1977)

Approximation Algorithms for Multiple Strip Packing*

Marin Bougeret[1], Pierre Francois Dutot[1], Klaus Jansen[2], Christina Otte[2], and Denis Trystram[1]

[1] LIG, Grenoble University, France
{bougeret,dutot,trystram}@imag.fr
[2] Department of Computer Science, Christian-Albrechts-University Kiel, Christian-Albrechts-Platz 4, 24098 Kiel, Germany
{kj,cot}@informatik.uni-kiel.de

Abstract. In this paper we study the Multiple Strip Packing (MSP) problem, a generalization of the well-known Strip Packing problem. For a given set of rectangles, $r_1, \ldots, r_n$, with heights and widths ≤ 1, the goal is to find a non-overlapping orthogonal packing without rotations into $k \in \mathbb{N}$ strips $[0,1] \times [0,\infty)$, minimizing the maximum of the heights. We present an approximation algorithm with absolute ratio 2, which is the best possible, unless $\mathcal{P} = \mathcal{NP}$, and an improvement of the previous best result with ratio $2+\varepsilon$. Furthermore we present simple shelf-based algorithms with short running-time and an AFPTAS for MSP. Since MSP is strongly $\mathcal{NP}$-hard, an FPTAS is ruled out and an AFPTAS is also the best possible result in the sense of approximation theory.

Keywords: Strip Packing, Scheduling in grids.

1 Introduction

In this paper we study the Multiple Strip Packing (MSP) problem, a generalization of the well-known Strip Packing (SP) problem. For a given set of rectangles, $r_1, \ldots, r_n$, with heights and widths ≤ 1, the goal is to find a non-overlapping orthogonal packing without rotations into $k \in \mathbb{N}$ strips $[0,1] \times [0,\infty)$, minimizing the maximum of the heights. As much as Strip Packing, its generalization Multiple Strip Packing is not only of theoretical interest, but also has many applications to real-world problems as in computer grids, server consolidation and in cutting problems. In computer grids for example, MSP is related to the problem of finding a schedule for parallel tasks into different clusters of processors with minimum makespan [12]. Consider an instance $L = \{r_1, \ldots, , r_n\}$ of MSP. The value k always denotes the number of strips $S_1, \ldots, S_k$. For $i \in \{1, \ldots, k\}$ the

* Work supported by EU project "'AEOLUS: Algorithmic Principles for Building Efficient Overlay Computers"', EU contract number 015964, and DFG project JA612/12-1, "'Design and analysis of approximation algorithms for two- and three-dimensional packing problems"', and DGA-CNRS.

value h_i denotes the height of a feasible packing in strip S_i. For an algorithm A for MSP let $A(L)$ be the output of the algorithm, in this case the maximum height of the packing generated, i.e. $\max_{i\in\{1,\dots,k\}} h_i$. The optimal value is denoted with $OPT(L)$, in this case the minimal height that can be achieved. The quality of an approximation algorithm is measured by its performance ratio. For a minimization problem as MSP we say that A has *absolute ratio* α, if $\sup_L {}^{A(L)}/_{OPT(L)} \leq \alpha$, and *asymptotic ratio* α, if $\alpha \geq \limsup_{OPT(L)\to\infty} {}^{A(L)}/_{OPT(L)}$, respectively. A minimization problem admits an *(asymptotic) polynomial-time approximation scheme* ((A)PTAS), if there exists a family of polynomial-time approximation algorithms $\{A_\varepsilon | \varepsilon > 0\}$ of (asymptotic) $(1+\varepsilon)$-approximations. We call an approximation scheme *fully polynomial* ((A)FPTAS), if the running-time of every algorithm A_ε is bounded by a polynomial in n and $\frac{1}{\varepsilon}$. Zhuk showed in [16] that there is no approximation algorithm for MSP with absolute ratio less than 2. Since MSP can be reduced to 3-Partition, it is also strongly $\mathcal{NP}$-hard. Therefore a PTAS and an FPTAS are ruled out and an AFPTAS is asymptotically the best possible.

A related problem is 3D Strip Packing (3SP), which also is a generalization of Strip Packing. Here the goal is to find a packing of a given list of cuboids with side lengths bounded by one into a 3-dimensional strip $[0,1]\times[0,1]\times[0,\infty)$, minmizing the height of the packing. Multiple Strip Packing with k strips can be reduced to 3SP by introducing a cuboid with depth $^1/_k$ for each rectangle packing the strips next to each other.

Parallel Job Scheduling in Grids with identical machines is also a related problem. In the offline case we have m machines M_i with ℓ processors and jobs $j \in J$ with processing time p_j, and a size $size_j$. The jobs must be executed on parallel processors within one machine M_i, but not necessary on consecutive processors. The machines can be seen as strips with width l and the jobs as vertically scissile rectangles with width $size_j$ and height p_j. In Multiple Strip Packing we have just the additional constraint that a job must be scheduled on consecutive processors. Unfortunately this is the reason why approximation algorithms for Parallel Job Scheduling cannot be applied to MSP maintaining their ratio.

Known Results. Multiple Strip Packing was first considered by Zhuk [16], who showed that there is no approximation algorithm with abolute ratio better than 2, and later by Ye et. al. [15]. Both concentrated on the online case. Additonally an approximation algorithm for the offline case with ratio $2+\varepsilon$ was achieved in [15]. For Strip Packing Coffman et al. gave in [8] an overview about performance bounds for shelf-orientated algorithms as $NFDH$ (Next Fit Decreasing Height) and $FFDH$ (First Fit Decreasing Height). Those adopt an absolute ratio of 3, and 2.7, respectively. Schiermeyer [11] and Steinberg [13] presented independently an algorithm for SP with absolute ratio 2. A further important result is an AFPTAS for SP with additive constant $\mathcal{O}(^1/_{\varepsilon^2})$ of Kenyon and Rémila [9]. This constant was improved by Jansen and Solis-Oba, who presented in [7] an APTAS with additive constant 1. For 3SP Jansen and Solis-Oba obtained an algorithm with ratio $2+\varepsilon$ in [6] as an improvement of the formerly known result by Miyazawa and Wakabayashi [10], who presented an algorithm with asymptotic ratio at most 2.64. Bansal et al. presented in [2] an algorithm for $3SP$ with a ratio of $T_\infty \approx 1.69$, which is the best known result. Schwiegelshohn et al. [12] achieved ratio 3 for a version of Parallel Job Scheduling in Grids without release times, and ratio 5 with release times. Tchernykh et al. presented in [14] an algorithm with absolute ratio 10 for the case of machines with different numbers of processors and without release times. However, this algorithm cannot be applied directly to MSP because of the non-contiguity.

Our Results. In this paper we present an approximation algorithm with absolute ratio 2, which is an improvement of the former result of $2+\varepsilon$ by Ye et al. [15] and best possible, unless $\mathcal{P} = \mathcal{NP}$. We also introduce an AFPTAS for Mutiple Strip Packing, which is a generalization of the algorithm of Kenyon and Rémila [9]. Our algorithm achieves an additive constant of $\mathcal{O}(1)$, if the number of strips is sufficient large, otherwise an additive constant of $\mathcal{O}(1/\varepsilon^2)$. Furthermore we show how to use the simple shelf-based heuristics $NFDH$ and $FFDH$ to obtain approximation algorithms for MSP with the same asymptotic ratio as for SP.

Organisation of the Paper. In the next section we introduce two shelf-based algorithms, using Next Fit and First Fit policies. In Section 3 we present a 2-approximation for MSP. Here we distinguish between different sizes for k. For $k=1$ we use the 2-approximation of Steinberg [13] or Schiermeyer [11]. If $k=2$ or bounded by a specified constant c we make use of a result by Bansal et al. [1] for Rectangle Packing with Area Maximization (RPA). For $k \geq c$ we use an approximation algorithm for 2D bin packing with asymptotic ratio 1.69 of Caprara [3]. In the last section we present an AFPTAS for MSP. Here we generalize the algorithm by Kenyon and Rémila [9]. Interestingly, the additive constant in our AFPTAS can be reduced from $\mathcal{O}(1/\varepsilon^2)$ to $\mathcal{O}(1)$, if the number k of strips is large enough.

2 Shelf-Based Algorithms

In this section we modify the shelf-based heuristics NFDH and FFDH. [8]. A shelf is a row of items placed next to each other left-justified. The baseline of a shelf is either the bottom of the bin or the extended upper edge of the tallest item packed in the shelf below. NFDH generates for a given list of rectangles $L = \{r_1, \ldots, r_n\}$ a packing into a strip with height at most $2OPT_{SP}(L)+h_{\max}$, FFDH produces a packing of height at most $1.7OPT_{SP}(L)+h_{\max}$, where $OPT_{SP}(L)$ is the optimum value of Strip Packing for the instance L and $h_{\max}$ is the height of the tallest item in L. Via this modification we obtain approximation algorithms for Multiple Strip Packing with the same asymptotic ratios. Furthermore, we present another algorithm, that computes for rectangles with widths bounded by $\varepsilon < 1$ a packing of height at most $1/(1-\varepsilon) OPT(L) + 2h_{\max}$.

Theorem 1. *Let A be one of the shelf-based Strip Packing algorithms NFDH or FFDH with asymptotic ratio $\alpha > 1$, that creates for an instance L a packing of height less than $\alpha OPT_{SP}(L) + h_{\max}$. For any $k \in \mathbb{N}$ there exists an algorithm A_k that packs a list of rectangles L into k strips with $A_k(L) \leq \alpha OPT(L) + h_{\max}$.*

Proof. For any instance L of MSP we define the algorithm A_k as follows

1 Pack the sorted rectangles with A into one strip S. (In particular the rectangles are first sorted by non-increasing height.) Let $A(L)$ denote the height of S.
2 Cut out the first shelf and pack it into the first strip S_1.
3 Divide the residual strip S into k parts:
  3.1 For each $\ell \in \{0, 1, \ldots, k\}$ draw a horizontal line across S at height $\ell(A(L) - h_{\max})/k$.
  3.2 For $\ell \in \{0, 1, \ldots, k-1\}$ pack all items intersecting the ℓth line and all items between the ℓth and $(\ell+1)$th line into strip $S_{\ell+1}$.

We show now that for any instance L of MSP the output of A_k is less than $\alpha\mathrm{OPT}(L) + h_{\max}$. Let $t \in \mathbb{N}$ be the number of shelves produced by A in Step 1 and H_j, $j \in \{1, \ldots, t\}$, the height of the jth shelf. Since there are no items intersecting the 0th line (see Fig 1), the height h_1 of the first strip S_1 is bounded by $h_{\max} + \frac{1}{k}\left(\sum_{j=1}^{t} H_j - h_{\max}\right)$ after the last step of the algorithm. For a strip S_i, $i \in \{2, \ldots, k\}$, containing the items between the $(i-1)$th and the ith line and the ones intersecting the $(i-1)$th line, we have $h_i \leq \frac{\sum_{j=1}^{t} H_j - h_{\max}}{k} + h_{\max} = \frac{A(L) - h_{\max}}{k} + h_{\max}$. We conclude

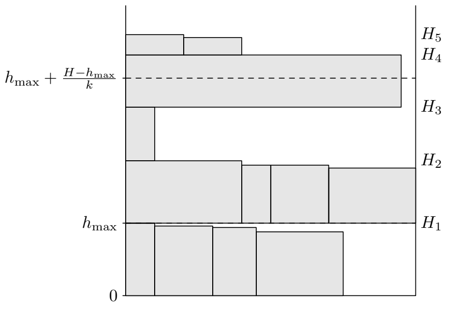


Fig. 1. Dividing strip S

$$A_k(L) = \max_{i \in \{1,\ldots,k\}} h_i \leq \frac{A(L) - h_{\max}}{k} + h_{\max}$$
$$\leq \frac{\alpha\mathrm{OPT}_{SP}(L) + h_{\max} - h_{\max}}{k} + h_{\max} = \frac{\alpha\mathrm{OPT}_{SP}(L)}{k} + h_{\max}.$$

Since $1/k\,\mathrm{OPT}_{SP}(L)$ is a lower bound for $\mathrm{OPT}(L)$ the proof is complete.

The running-time of the above algorithm is $\mathcal{O}(n \log n)$.

Corollary 1. *Let L be an instance of MSP. In a packing generated by the above algorithm A_k we have $\max_{i\in\{1,\ldots,k\}} |h_i - A_k(L)| \leq 2h_{\max}$, where h_i denotes the height of strip S_i.*

Another way to pack a set of rectangles with a modified version of the $NFDH$ heuristic into k strips is the following:

Algorithm 2

```
1 Sort the rectangles by non-increasing height.
2 For each $i \in \{1,\ldots,k\}$ pack one shelf according to the $NFDH$ heuristic into
  strip $S_i$, that means starting in the lower left corner pack the rectangles next
  to each other on the baseline of strip $S_i$, until the next rectangle does not fit.
  Draw a new baseline at the top edge of the tallest rectangle (that clearly is
  the first one).
3 Take the strip $S^-$ with the current lowest height $h^-$ and pack one shelf
  according to the $NFDH$ heuristic on top of the shelves.
4 Repeat Step 3 until all rectangles are packed.
```

The packing generated by the above algorithm is very smooth, in the sense that the heights of the strips only differ by $h_{\max}$.

Lemma 1. *For a set of rectangles $L = \{r_1,\ldots,r_n\}$ Algorithm 2 with output $A(L)$ generates a packing into k strips, so that $\max_{i\in\{1,\ldots,k\}} |A(L) - h_i| \leq h_{\max}$.*

This leads to a further result about rectangles with bounded width. Coffman et al. showed in [8] that $FFDH$ applied to an instance L of rectangles with widths bounded by $1/m$ for some integer m generates a packing into a strip of height at most $(1 + \frac{1}{m})OPT_{SP}(L) + h_{\max}$. Our result for packing into k strips is the following:

Theorem 3. *For a set of rectangles $L = \{r_1,\ldots,r_n\}$ with widths bounded by $\varepsilon > 0$ we obtain by the Algorithm 2 with output $A(L)$ a packing into k strips with height less than $\frac{1}{1-\varepsilon}OPT(L) + 2h_{\max}$.*

For $\varepsilon = \frac{1}{m}$ this is equal to $A(L) \leq \left(1 + \frac{1}{m-1}\right)\mathrm{OPT}(L) + 2h_{max}$.

3 A Two-Approximation for MSP

In this section we construct a polynomial-time approximation algorithm for MSP with absolute ratio 2. Since there is no approximation algorithm for MSP with ratio smaller than 2 (unless P=NP), this is the best possible result. To handle different sizes of k we use, besides the well-known algorithms of Steinberg [13] or Schiermeyer [11], a result of Bansal et al. [1] for Rectangle Packing with Area Maximization (RPA) and a of Caprara [3].

3.1 One or Two Strips

The case $k = 1$ is trivial, because we can use the algorithm of Steinberg [13] or Schiermeyer [11] with absolute performance bound 2.

Theorem 4 (Steinberg [13]). *Let $L = \{r_1, \ldots, r_n\}$ be a set of rectangles with heights h_i and widths w_i and Q be a rectangle with width u and height v. Let $h := \max_{i\in\{1,\ldots,n\}} h_i$ and $w := \max_{i\in\{1,\ldots,k\}} w_i$. If the following inequalities hold,*

$$w \leq u, \quad h \leq v, \quad 2SIZE(L) \leq uv - (2w - u)_+(2h - v)_+ \tag{1}$$

then it is possible to pack L into the rectangle Q.(As usual, $x_+ = \max(x, 0)$.)

Therefore let us first consider the case for $k = 2$. Here we use the PTAS found by Bansal et al. [1] for RPA. In RPA we are given a set of rectangles $L = \{r_1, \ldots, r_n\}$ with widths w_i and heights h_i and a bin of unit size. The goal is to find a feasible packing of a subset $L' \subset L$ of the rectangles and to maximize the area of the rectangles in L'.

Algorithm 5

1 *Guess the height of an optimal solution for MSP and denote it with v.*
2 *Scale the heights of the rectangles in L by $1/v$ so that the corresponding packing fits into one bin of height and width one.*
3 *The set of resulting rectangles L_v is now considered as an instance of RPA with $OPT_{RPA}(L) = SIZE(L_v)$, where $SIZE(L_v)$ is the total area of all rectangles in L_v. Apply the algorithm in [1] with accuracy $\varepsilon = 1/2$ and find a packing of a subset $L'_v \subset L_v$ with total area at least $(1-\varepsilon)SIZE(L_v)$. By rescaling the rectangles of L'_v get a packing for the first strip with height at most v.*
4 *Since $SIZE(L_v) \leq 2$ the remaining items in $L_v \backslash L'_v$ have total area $SIZE(L_v \backslash L'_v) \leq \varepsilon SIZE(L_v) \leq 1$. Therefore we can pack them with Steinberg's algorithm into a strip of height at most 2. Rescaling gives us a second strip of height at most $2v$.*

The running-time of the algorithm is polynomial in n:
In the first step we can assume that the heights of the rectangles are rational, so by multiplying with a common denominator they become integer values. Then the optimum height v of MSP is also integer and equals a sum of heights of the rectangles in L, so we have $h_{\max} \leq v \leq nh_{\max}$. Thus Binary Search takes at most $\log(nh_{\max})$ iterations to find the value v. Step 3 is also polynomial, since we apply the algorithm in [1] for a fixed accuracy $\varepsilon = 1/2$.

3.2 A Bounded Number of Strips

In the case of a constant number of strips we can use an extended version of the PTAS for RPA in [1] called $kRPA$. Another helpful tool is the next lemma. The proof can be obtained applying Steinberg's algorithm for $h, w, u = 1$ and $v = k/2$ in equation 1.

Lemma 2. *If L is an instance of 2DBP with total area $SIZE(L) \leq {}^{k}/_{4}$ and $k \geq 3$, then there exists a packing of L into k bins.*

Algorithm 6

1 *Guess an optimal height for MSP and denote it with v.*
2 *Scale the heights of the rectangles in L by ${}^{1}/_{v}$ so that the corresponding packing fits into k bins of height and width one.*
3 *The set of resulting rectangles L_v is now considered as an instance of RPA with $OPT_{RPA}(L) = SIZE(L_v)$. Apply kRPA to k bins of unit size and find for an accuracy $\varepsilon \leq {}^{1}/_{4}$ a packing for a subset $L'_v \subset L_v$ with total area $(1-\varepsilon)SIZE(L_v)$. By rescaling the rectangles of L'_v we get k bins of height v.*
4 *For the total area of the remaining rectangles in $L_v \backslash L'_v$ we have $SIZE(L_v \backslash L'_v) = \varepsilon SIZE(L_v) \leq {}^{k}/_{4}$. Pack those rectangles according to Lemma 2 into k bins and rescale the rectangles. This results again in k bins of height at most v.*
5 *Stack every two bins on top of each other and get a solution with k bins of height at most $2v$.*

3.3 A Large Number of Strips

Caprara presented in [3] a shelf algorithm for $2DBP$ that produces a solution whose asymptotic ratio can be made arbitrarily close to $T_\infty = 1.69...$. Clearly if the number of strips is large enough ($\approx 10^4$) applying this algorithm we get a two-approximation for MSP stacking every two bins on each other. Alternatively, we can use the recently published two-approximation for 2DBP by Jansen et al. [5] to achieve this result. Along with the previous sections we have the following:

Theorem 7. *For any $k \in \mathbb{N}$ there is a polynomial-time algorithm for MSP with absolute ratio two.*

4 An AFPTAS for MSP

In this section we present an AFPTAS for MSP. The algorithm is a generalization of an AFPTAS found by Kenyon and Rémila [9] for Strip Packing. For an instance L of Strip Packing and an accuracy $\varepsilon > 0$ their algorithm generates a packing with height $(1+\varepsilon)OPT_{SP}(L) + \mathcal{O}({}^{1}/_{\varepsilon^2})h_{\max}$. Our algorithm achieves the same ratio for Multiple Strip Packing. For instances with k sufficient large, namely $k \in \Omega({}^{1}/_{\varepsilon^3})$, our algorithm adopts an improved additive constant of $\mathcal{O}(1)$. More precisely for an accuracy ε and $k \geq \lceil {}^{128}/_{\varepsilon^3} \rceil$ we get an approximation ratio of $(1+\varepsilon)\mathrm{OPT}(L) + 6h_{\max}$.

4.1 The Regular Case

As in Section 2 we divide a packing into one strip into k parts of nearly the same height and distribute them to k strips.

Theorem 8 (Kenyon & Rémila [9]). *For a list $L = \{r_1, \ldots, r_n\}$ of rectangles and an accuracy $\varepsilon > 0$ the algorithm A_ε^{KR} in [9] generates a packing into one strip with height at most $(1+\varepsilon)OPT_{SP}(L) + (4(\frac{2+\varepsilon}{\varepsilon})^2 + 1)h_{\max}$.*

Our result is the following:

Theorem 9. *For a list $L = \{r_1, \ldots, r_n\}$ of rectangles with widths and heights ≤ 1 and an accuracy $\varepsilon > 0$ there exits an algorithm A_ε that generates a packing into k strips, so that $A_\varepsilon(L) \leq (1+\varepsilon)OPT(L) + (2(\frac{2+\varepsilon}{\varepsilon})^2 + 2)h_{\max}$.*

4.2 Instances with a Large Number of Strips

In this section we consider the case $k \geq \lceil {}^{128}/_{\varepsilon^3} \rceil$. In this case it is possible to improve the additive constant to $\mathcal{O}(1)h_{\max}$ by balancing the configurations.

Rounding. We choose $\varepsilon' = {}^{\varepsilon}/_{4}$ (w.l.o.g. ${}^{1}/_{\varepsilon'}$ integral) and divide the list of rectangles L into a list of narrow rectangles $L_{narrow} := \{r_i \in L | w(r_i) \leq \varepsilon'\}$ and a list of wide rectangles $L_{wide} := \{r_i \in L | w(r_i) > \varepsilon'\}$. Then we round L_{wide} to an instance L_{sup} with only $M := ({}^{1}/_{\varepsilon'})^2$ different widths. For the rounding step we put the wide rectangles sorted by non-increasing widths left-aligned on a stack. Let $STACK(L)$ denote the total area of the plane covered by this stack and let H denote its height. Moreover, for arbitrary lists L'', L' we define a relation $\leq_g$, so that $L'' \leq_g L'$, if and only if $STACK(L'') \subseteq STACK(L')$. We draw $M-1$ horizontal lines through $STACK(L)$ with distance ${}^{H}/_{M}$ starting at the bottom. Therefore we get M so-called *threshold* rectangles. A rectangle is a threshold rectangle if it either with its interior or with its lower edge intersects a line at height ${}^{iH}/_{M}$, $i \in \{1, \ldots, M-1\}$. For $i \in \{1, \ldots, M-1\}$ we round up the width of each rectangle between the lines ${}^{iH}/_{M}$ and ${}^{(i+1)H}/_{M}$ to the width of the ith threshold rectangle. The widths of the rectangles below the first line are rounded up to the width of the undermost rectangle in the stack. So we get at most M groups of different widths (see Fig 2). Furthermore, we get a list L_{sup} of rectangles with widths larger than ε' and only M different widths, in particular we have $L_{wide} \leq_g L_{sup}$.

Fractional Packing. Our first objective is to create a fractional packing for the wide rectangles into k strips. To do this we introduce *configurations.* A configuration is a non-empty multiset of widths, which sum up to less than one. Denote with q the number of different configurations C_j with height x_j. Let α_{ij} be the number of occurence of width w_i in configuration C_j and let β_i be the total height of all rectangles of width w_i. Based on the solution of the following Linear Program

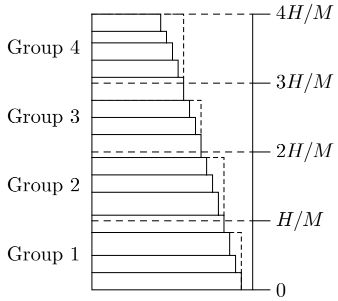


Fig. 2. Rounding the rectangles in L_{sup}

$$\begin{aligned} &\min \frac{\sum_{j=1}^{q} x_j}{k} \\ &\text{s.t.} \sum_{j=1}^{q} \alpha_{ij} x_j \geq \beta_i \text{ for all } i \in \{1, \ldots, M\} \\ &x_j \geq 0 \text{ for all } j \in \{1, \ldots, q\}, \end{aligned} \qquad (\text{LP}(L_{sup}))$$

by distributing the configurations to k strips we get the requested fractional packing for the rectangles in L_{sup}. Note that $\text{rank}(\alpha_{ij})_{ij} \leq M$ and hence a basic solution x of $LP(L_{sup})$ has at most M nonzero entries. In the next section we show how to transform a fractional packing into a feasible packing for L_{sup}. Later the rectangles in L_{narrow} are packed into the idle space in a Greedy manner. For a list L of rectangles let $LIN(L)$ denote the height of an optimum fractional packing for L. Let $h_0 := LIN(L_{sup})$ and note that $h_0 \leq OPT(L)$.

Lemma 3. *Let $x = (x_1, \ldots, x_q)$ be a solution of $LP(L_{sup})$ with at most $m \leq M$ nonzero entries $x_1, \ldots, x_m$. For $k \geq \lceil 128/\varepsilon^3 \rceil$ we get a fractional packing into k strips with height at most $(1 + \varepsilon')h_0$ and at most $m' \leq 2M$ different configurations.*

Proof. First we fractionally pack the rectangles into the configurations. Imagine each configuration C_j as a bin with height x_j and width c_j and divide it into α_{ij} columns of widths w_i and height x_j. Pack the rectangles in L_{sup} of width w_i in a Greedy manner fractionally into the columns of width w_i until exactly height x_j, starting with $j = 1$. In this way each column contains a sequence of rectangles, which completely fits inside the column, and possibly the top part of a rectangle, that started in a previous column, and the bottom part of a rectangle, that is too tall to fit into this column. Since $\sum_{j=1}^{m} \alpha_{ij} x_j \geq \beta_i$, there will be maybe more than enough space for the rectangles of width w_i in the configurations. In this case we distribute the rectangles among the columns and delete the additional space. So we split a configuration C_j into two parts, one of the old type where the columns of width w_i are completely filled and one without columns of width

w_i. This case may happen only M times. So we have in total $m' = m + M \leq 2M$ configurations $C_1, \dots C_{m'}$ with nonzero heights $x_1, \dots, x_{m'}$.

Notice that there exist configurations with height larger or equal h_0, since if not we conclude $\sum_{j=1}^{m'} x_j < m'h_0 \leq 2Mh_0 \overset{\varepsilon'=\varepsilon/4}{=} \frac{32h_0}{\varepsilon^2} < kh_0$, which is a contradiction. Consider a configuration C_j, $j \in \{1, \dots, m'\}$. If $x_j \geq h_0$ we allocate $\lfloor x_j/h_0 \rfloor$ empty strips with height h_0 for C_j. If then $x_j/h_0 - \lfloor x_j/h_0 \rfloor \leq \varepsilon' h_0$, we assign to C_j additional space with height $(x_j/h_0 - \lfloor x_j/h_0 \rfloor)$ in a strip, that has already height h_0. If $x_j/h_0 - \lfloor x_j/h_0 \rfloor > \varepsilon' h_0$, we divide $(x_j/h_0 - \lfloor x_j/h_0 \rfloor)$ into at most $1/\varepsilon'$ stripes with height less or equal $\varepsilon' h_0$. So assign to C_j additional space of height $\varepsilon' h_0$ in no more than $1/\varepsilon'$ strips, which are already occupied until height h_0. In the same way as the remaining stripes we handle configurations of height less than h_0. Since there are at most $2M$ configurations with nonzero height, we get at most $2M/\varepsilon' = 2/\varepsilon'^3 \leq 2 \cdot 4^3/\varepsilon^3 \leq k$ additional assignments of height $\varepsilon' h_0$, which can be distributed to k strips. Thus by this assignment policy, where the configurations are balanced, each strip has allocated area of height at most $(1+\varepsilon')h_0$ for at most 2 different configurations.

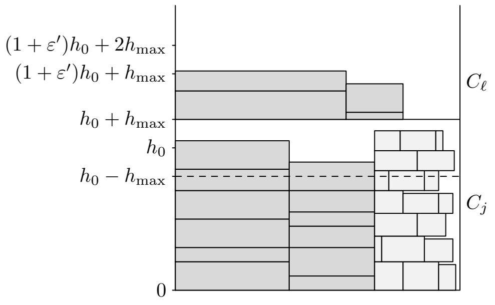

$(1+\varepsilon')h_0 + 2h_{\max}$
$(1+\varepsilon')h_0 + h_{\max}$
$h_0 + h_{\max}$
h_0
$h_0 - h_{\max}$
0
C_ℓ
C_j

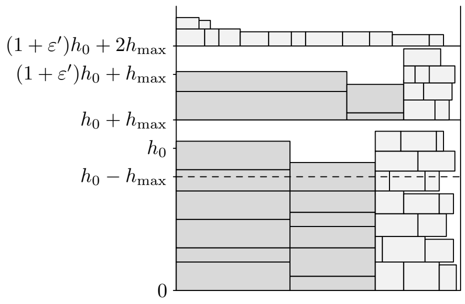
Fig. 3. S_i with C_j and C_ℓ

Integral Packing. The next Lemma shows how to get from a fractional packing to a feasible integral packing. A proof is given in the full paper.

Lemma 4. *Let $x = (x_1, \dots, x_q)$ be a solution of $LP(L_{sup})$ with at most $m' \leq 2M$ nonzero entries $x_1, \dots, x_{m'}$. For $k \geq \lceil 128/\varepsilon^3 \rceil$ we can convert x to a feasible packing for the wide rectangles with height at most $(1+\varepsilon')h_0 + 2h_{\max}$ and at most 2 different configurations per strip.*

Since we can guarantee that there are at most 2 different configurations per strip, the additive constant will be improved, while the running-time is still polynomial in n and $1/\varepsilon$.

Our last step is to pack the narrow rectangles. We use a modified version of the NFDH algorithm: For strip S_i as above we pack narrow rectangles with NFDH into the empty space next to the configurations until the total height is at most $(1+\varepsilon')h_0 + 2h_{\max}$. After that we repeat the process for strip S_{i+1}. When all



$(1+\varepsilon')h_0 + 2h_{\max}$
$(1+\varepsilon')h_0 + h_{\max}$
$h_0 + h_{\max}$
h_0
$h_0 - h_{\max}$
0

Fig. 4. S_i after packing the narrow rectangles

strips are filled in this way, we draw a horizontal line at height $(1+\varepsilon')h_0 + 2h_{\max}$ in each strip and pack the remaining narrow rectangles with Algorithm 2 on top (see Fig 3 and 4). Thus we can ensure by Lemma 1 that the maximum difference of the heights of two arbitrary strips is at most $h_{\max}$ (see Fig 4). Let h_{final} denote the height of the packing after packing the narrow rectangles.

Lemma 5. *Let $k \geq \lceil 128/\varepsilon^3 \rceil$. If $h_{final} \geq (1+\varepsilon')h_0 + 2h_{\max}$, then we have $h_{final} \leq \frac{SIZE(L_{sup} \cup L_{narrow})}{k(1-\varepsilon')} + 6h_{\max} + \varepsilon' h_0$.*

For details we refer to the full paper. The next lemma is shown in [9] for the Linear Program corresponding to Strip Packing, but obviously also holds for our linear program $LP(L_{sup})$.

Lemma 6. *[9] For the rounded instance L_{sup} and L_{wide} the inequalities $LIN(L_{sup}) \leq LIN(L_{wide})\left(1+\frac{1}{M\varepsilon'}\right)$ and $SIZE(L_{sup}) \leq SIZE(L_{wide})\left(1+\frac{1}{M\varepsilon'}\right)$ hold.*

The entire algorithm is now defined as follows:

Algorithm 10

```
1 Set $\varepsilon' := \varepsilon/4$ and $M := (1/\varepsilon')^2$.
2 Partition $L$ into $L_{wide}$ and $L_{narrow}$.
3 Construct $L_{sup}$, so that $L_{wide} \leq_g L_{sup}$ and there are only $M$ different widths
  in $L_{sup}$.
4 Solve the linear program $LP(L)$.
5 Construct a feasible solution for $L_{sup}$ by balancing the configurations.
6 Use modified NFDH to pack the rectangles in $L_{narrow}$ into the remaining
  space and on top of the strips.
```

Theorem 11. *If $k \geq \lceil 128/\varepsilon^3 \rceil$ the Algorithm 10 generates for an instance L of MSP a packing of height at most $(1+\varepsilon)OPT(L) + \mathcal{O}(1)h_{\max}$.*

References

1. Bansal, N., Caprara, A., Jansen, K., Prädel, L., Sviridenko, M.: How to maximize the total area of rectangle packed into a rectangle. To appear in The 20th International Symposium on Algorithms and Computation, ISAAC 2009 (2009)
2. Bansal, N., Han, X., Iwama, K., Sviridenko, M., Zhang, G.: Harmonic algorithm for 3-dimensional strip packing problem. In: Proceedings of the eighteenth ACM-SIAM symposium on Discrete algorithm (SODA 2007), pp. 1197–1206 (2007)
3. Caprara, A.: Packing 2-dimensional bins in harmony. In: Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS 2002), pp. 490–499 (2002)
4. Caprara, A., Lodi, A., Monaci, M.: An approximation scheme for the two-stage, two-dimensional bin packing problem. In: Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization, pp. 315–328 (2002)
5. Jansen, K., Prädel, L., Schwarz, U.M.: Two for one: Tight approximation of 2d bin packing. In: Algorithms and Data Structures Symposium, WADS 2009 (2009)
6. Jansen, K., Solis-Oba, R.: An asymptotic approximation algorithm for 3d-strip packing. In: Proceedings of the seventeenth ACM-SIAM symposium on Discrete algorithm (SODA 2006), pp. 143–152 (2006)
7. Jansen, K., Solis-Oba, R.: New approximability results for 2-dimensional packing problems. Journal of Discrete Optimization 6, 310–323 (2009)
8. Coffman Jr., E.G., Garey, M.R., Johnson, D.S., Tarjan, R.E.: Performance bounds for level-oriented two-dimensional packing algorithms. SIAM Journal of Computing 9(4), 808–826 (1980)
9. Kenyon, C., Rémila, E.: A near optimal solution to a two-dimensional cutting stock problem. Mathematics of Operations Research 25, 645–656 (2000)
10. Miyazawa, F., Wakabayashi, Y.: Cube packing. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 58–67. Springer, Heidelberg (2000)
11. Schiermeyer, I.: Reverse-fit: A 2-optimal algorithm for packing rectangles. In: Proceedings of the Second European Symposium on Algorithms, pp. 290–299. Springer, Heidelberg (1994)
12. Schwiegelshohn, U., Tchernykh, A., Yahyapour, R.: Online scheduling in grids. In: IEEE Internation pp. 1–10 (2008)
13. Steinberg, A.: A strip-packing algorithm with absolute performance bound 2. SIAM Journal of Computing 26(2), 401–409 (1997)
14. Tchernykh, A., Ramírez, J., Avetisyan, A., Kuzjurin, N., Grushin, D., Zhuk, S.: Two level job-scheduling strategies for a computational grid. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 774–781. Springer, Heidelberg (2006)
15. Ye, D., Han, X., Zhang, G.: On-line multiple-strip packing. In: The 3rd Annual International Conference on Combinatorial Optimization and Applications COCOA 2009 (2009)
16. Zhuk, S.N.: Approximate algorithms to pack rectangles into several strips. Discrete Mathematics and Applications 16(1), 73–85 (2006)

Approximating Frequent Items in Asynchronous Data Stream over a Sliding Window

Ho-Leung Chan, Tak-Wah Lam*, Lap-Kei Lee, and Hing-Fung Ting**

Department of Computer Science, University of Hong Kong
{hlchan,twlam,lklee,hfting}@cs.hku.hk

Abstract. In an asynchronous data stream, the data items may be out of order with respect to their original timestamps. This paper gives a space-efficient data structure to maintain such a data stream so that it can approximate the frequent item set over a sliding time window with sufficient accuracy. Prior to our work, Cormode *et al.* [3] have the best solution, with space complexity $O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W}) \min\{\log W, \frac{1}{\varepsilon}\} \log U)$, where ε is the given error bound, W and B are parameters of the sliding window, and U is the number of all possible item names. Our solution reduces the space to $O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W}))$. We also unify the study of synchronous and asynchronous data stream by quantifying the delay of the data items. When the delay is zero, our solution matches the space complexity of the best solution to the synchronous data streams [8].

1 Introduction

Identifying frequent items in a massive data stream has many applications in data mining and network monitoring, and the problem has been studied extensively [8,7,9,6,5,1]. Recent interest has been shifted from the statistics of the whole data stream to that of a sliding window of recent data [1,8]. In most applications, the amount of data in a window is gigantic when compared with the amount of memory in the processing units, and it is impossible to store all the data and to find the exact frequent items. Existing research has focused on finding space-efficient data structures to support finding approximate frequent items. The key concern is how to minimize the space so as to achieve a given level of accuracy.

Asynchronous data stream. Most of the previous work on data streams assume that items in a data stream are synchronous in the sense that the order of their arrivals is the same as the order of their creations. This synchronous model is however not suitable to applications that are distributed in nature. For example, in a sensor network, the sink collects data transmitted from sensors over a large area, and the data transmitted from different sensors would suffer different delay. It is possible that an item created at time t at a certain

* T.W. Lam is partially supported by GRF Grant HKU 713909E.
** H.F. Ting is partially supported by GRF Grant HKU 716307E.

sensor may arrive at the sink later than an item created after t at another sensor. From the sink's viewpoint, items in the data stream are out of order with respect to their creation times. Yet the statistics to be computed are usually based on the creation times. More specifically, an *asynchronous data stream* (or equivalently, *out-of-order data stream*) [11,2,3] can be considered as a sequence $\langle (a_1, d_1), (a_2, d_2), (a_3, d_3), \ldots \rangle$, where a_i is the name of a data item chosen from a fixed universe Σ, and d_i is an integer timestamp recording the creation time of this item. Items arriving at the data stream are in arbitrary order regarding their timestamps. Furthermore, it is possible that more than one data item has the same timestamp.

Previous work on approximating frequent items. Consider a data stream and, in particular, those data items whose timestamps fall into the last W time units (W is the size of the sliding window). An item (precisely, an item name) is said to be a frequent item if its count (i.e., the number of occurrences) exceeds a certain required threshold of the total item count. Arasu and Manku [1] were the first to study the data structures for computing approximate frequent items over a sliding window under the synchronous model (in which data items arrive in non-decreasing order of timestamps). The space complexity is $O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon}\log(\varepsilon B))$, where ε is a user-specified error bound and B is the maximum number of items with timestamps falling into the same sliding window. Their work was later improved by Lee and Ting [8] to $O(\frac{1}{\varepsilon}\log(\varepsilon B))$ space. Recently, Cormode *et al.* [3] has initiated the study of frequent items under the asynchronous model, and gave a solution with space complexity $O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W})$ $\min\{\log W, \frac{1}{\varepsilon}\}\log U)$, where $U = |\Sigma|$ is the number of possible item names.

The earlier work on asynchronous data stream focused on a relatively simpler problem called basic counting [11,2].[1] In the same paper, Cormode *et al.* [3] improved the space complexity of basic counting to $O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W}))$. Notice that under the synchronous model, the best data structure requires $O(\frac{1}{\varepsilon}\log(\varepsilon B))$ space [4]. It is believed that there is roughly a gap of $\log W$ between the synchronous model to the asynchronous model. Yet, for frequent items, the asynchronous result of Cormode *et al.* [3] requires space way bigger than that of the best synchronous result (which is $O(\frac{1}{\varepsilon}\log(\varepsilon B))$) [8]. This motivates us to study better solutions for approximating frequent items in the asynchronous model.

Formal definition of approximate frequent item set. Consider an asynchronous data stream $\langle (a_1, d_1), (a_2, d_2), \ldots \rangle$. The current time is defined to be the maximum timestamp over all items received thus far. If the current time is t, a sliding window of size W covers the time interval $[t-W+1, t]$. For any time interval I and any data item a, let $f_a(I)$ denote the frequency of item a in interval I, i.e., the number of items named a in the data stream with timestamps falling into I. Let Σ be the set of all possible item names. Define $f_*(I) = \sum_{a\in\Sigma} f_a(I)$ to be the total number of all items in the stream with timestamps within I.

[1] It is worth-mentioning that with respect to a sliding time window, the problem of approximating the basic counting can be reduced to the problem of approximating the frequent item set problem.

Table 1. The space and time complexity for finding ε-approximate frequent item set in a sliding time window. Results from this paper are marked with [†].

	Space (words)	Update time	Query time
Synchronous [8]	$O(\frac{1}{\varepsilon}\log(\varepsilon B))$	$O(\frac{1}{\varepsilon}\log(\varepsilon B))$	$O(\frac{1}{\varepsilon})$
Asynchronous [3]	$O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W}) \cdot \min\{\log W, \frac{1}{\varepsilon}\}\log U)$	$O(\log(\frac{\varepsilon B}{\log W})\log W \log\log U)$	linear to space
Asynchronous [†]	$O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W}))$	$O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log\log W))$	$O(\frac{1}{\varepsilon}\log W + \log\log(\varepsilon B))$
Asynchronous with tardiness τ [†]	$O(\frac{1}{\varepsilon}\log \tau \log(\frac{\varepsilon B}{\log \tau}))$	$O(\log(\frac{\varepsilon B}{\log \tau})(\frac{1}{\varepsilon} + \log\log W))$	$O(\frac{1}{\varepsilon}\log \tau + \log\log(\varepsilon B))$

Given a user-specified error bound ε and a window size W, we want to maintain a data structure to answer any ε-approximate frequent item set query in the form (ϕ, W'), where $\phi \geq \varepsilon$ and $W' \leq W$. The answer to such a query is a set S of item names defined as follows. Let t be the current time, and let $I = [t - W' + 1, t]$ be the current window.

(i) S contains every item a whose frequency in interval I is at least $\phi f_*(I)$ (i.e., $f_a(I) \geq \phi f_*(I)$); and
(ii) For any item a in S, its frequency in interval I is at least $(\phi - \varepsilon) f_*(I)$ (i.e., $f_a(I) \geq (\phi - \varepsilon) f_*(I)$).

For example, assume $\varepsilon = 1\%$, then the query $(10\%, 10000)$ would return all items whose frequencies are each at least 10% of the total item count in the last 10000 time units, plus possibly some items with frequency at least 9% of the total count. The set S is also called the ε-approximate ϕ-frequent item set.

Our contribution. Our main result is a space-efficient data structure for computing an ε-approximate frequent item set; it uses $O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W}))$ words, where B is the maximum number of items with timestamp in the same window of W time units. Our memory usage is much smaller than that of the algorithm in [3], i.e., $O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W}) \min\{\log W, \frac{1}{\varepsilon}\}\log U)$ words. We have also improved the time complexity of the update and query operations. See Table 1 for a comparison. Interestingly, the space complexity for finding approximate frequent items can now match the best space requirement for the relatively simpler problem of approximating basic counting (i.e., $O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W})))$ [3].

In the asynchronous model, if a data item has a delay more than W time units, it can be immediately discarded when it arrives at the data stream. On the other hand, in many applications, the delay is relatively small when compared with the window size. This motivates us to extend the asynchronous model to consider data items that have a bounded delay. We say that an asynchronous data stream has tardiness τ if a data item created at time d must arrive at the data stream no later than a data item created at time $d + \tau$. If we set $\tau = 0$, the model becomes the synchronous model. If we consider $\tau = W$, this is the asynchronous model studied above. In general, for any $\tau \in [0, W]$, we adapt our approximate data structure for the frequent item set to occupy $O(\frac{1}{\varepsilon}\log\tau\log(\frac{\varepsilon B}{\log\tau}))$ space. Note that our work implies the same space complexity as the best result under the synchronous model [8], but the underlying data structures and algorithms are more complicated than that in [8].

Technical digest. Our improved solution to the frequent item set problem stems from two new ideas, namely, a more versatile data structure called λ-counter for bookkeeping individual data items, and a technique of exploiting multi-resolution to optimize the space for organizing the λ-counters.

The λ-counter is a generalization of the data structure proposed by Lee and Ting [8], which estimates the number of an item within a sliding window under the synchronous model. Roughly speaking, the idea in [8] is to keep a sample of every λ item a's received, for some constant λ. In an out-of-order stream, the order of timestamps is arbitrary, and it does not make sense to sample every λ item a's in the stream. Our idea is to use an *interval splitting* technique, which dynamically (and erroneously) splits the current window into disjoint intervals of varying sizes but with similar number of items. An interesting point is that the splitting will inevitably introduce error, but we are able to keep the error in control. In Cormode *et al.*'s solution [3], they make use of the data structure q-digest (first proposed in [10]) for a similar purpose. Both q-digest and λ-counter allow efficient insertion of a new item. But an obvious advantage of λ-counter is that it also allows the deletion of the latest item in $O(1)$ time. The special deletion operation allows us to have a better and simpler way to organize the λ-counters for approximating frequent item set.

Based on the new λ-counters, it is not difficult to adapt Lee and Ting's synchronous data structure for approximating frequent item set [8] to the asynchronous model. The space complexity would be $O(\frac{1}{\varepsilon}(\log W)B)$. The extra factor B is due to the fact that the approximation has to be within an absolute error $\varepsilon f_*(I)$, where $f_*(I)$ can be as small as one and as big as B. An useful observation here is that there is a more error-sensitive way to exploit the λ-counters such that if $f_*(I)$ is restricted to be in the range $[\ell, r]$, then the space complexity for approximating frequent item set would be $O(\frac{1}{\varepsilon}(\log W)\frac{r}{\ell})$. We call such a structure the (Y,δ)-collection (see the details in Section 3). Then we can adopt the "multi-resolution" idea of [1] to keep $O(\log(\frac{\varepsilon B}{\log W}))$ data structures, each could estimate the frequent item set with sufficient accuracy when $f_*(I)$ is in a particular range, and each requires only $O(\frac{1}{\varepsilon}\log W)$ space.

Organization of the paper. Sections 2 and 3 present the data structures λ-counter and (Y,δ)-collection, respectively; the latter gives good estimates on the frequency of any item, when the total frequency of all items in the window is bounded by some fixed value Y. Section 4 uses the multi-resolution technique to remove the restriction on total frequency, and Section 5 shows how this data structure can identify ε-approximate frequent item set using $O(\frac{1}{\varepsilon}\log W\log(\frac{\varepsilon B}{\log W}))$ words of space. Finally, Section 6 extends the results for out-of-order stream with tardiness.

2 -Counter: Estimating the Frequency of a Fixed Item

This section presents a new data structure λ-counter, where $\lambda \geq 1$ is a parameter. Let W be the window size. Each λ-counter counts only one certain item, and it maintains an estimation for the number of this item within a sliding window of W' time units, for any $W' \leq W$. The absolute error in the estimation is at most $2\lambda \log W$. We first define a λ-counter and then present the analysis.

2.1 Definition of a -Counter

A λ-counter C_a for an item a is simply a list of intervals

$$\langle [p_1, q_1], [p_2, q_2], [p_3, q_3], \ldots, [p_k, q_k] \rangle, \tag{1}$$

where $p_i \leq q_i$ for $i = 1, \ldots, k$. We will maintain that $p_1 < q_k$, and the intervals in the list form a partition of $[p_1, q_k]$ (i.e., $p_{i+1} = q_i + 1$ for any $1 \leq i < k$); we say that C_a *monitors* the interval $[p_1, q_k]$. Each interval $[p_i, q_i]$ in the list is associated with a number $e_a([p_i, q_i])$, which is an estimate of $f_a([p_i, q_i])$, i.e., the number of item a in $[p_i, q_i]$. Initially, the list has only one interval $[1, W]$, and $e_a([1, W]) = 0$. A λ-counter is updated and answers queries as follows.

Updating a λ-counter. A λ-counter C_a is updated only when an item a arrives. Suppose a new item (a, t) arrives. Let the list of C_a be the one shown in (1) for some $k \geq 1$. Then we update the list according to the three cases below.

Case 1: $t < p_1$**.** We ignore the new item in this case.
Case 2: $p_1 \leq t \leq q_k$**.** In this case, let $[p_i, q_i]$ be the unique interval in the list with $t \in [p_i, q_i]$. We increase its estimate $e_a([p_i, q_i])$ by 1.
Then, if $p_i \neq q_i$ and $e_a([p_i, q_i]) = 2\lambda$, we do the following extra step. Let $m = \lfloor (p_i + q_i)/2 \rfloor$. We replace the interval $[p_i, q_i]$ by the two intervals $[p_i, m]$ and $[m + 1, q_i]$. Then we set both $e_a([p_i, m])$ and $e_a([m + 1, q_i])$ to λ.
Case 3: $q_k < t$**.** In this case, we have two subcases.

1. If $q_k < t \leq q_k + W$, then append the interval $[q_k + 1, q_k + W]$ to the list and set its estimate to 1.
2. If $q_k + W < t$, then find the unique integer ℓ such that $t \in [(\ell - 1)W + 1, \ell W]$, and append to the list the two intervals $[(\ell - 2)W + 1, (\ell - 1)W]$ and $[(\ell - 1)W + 1, \ell W]$ with estimates 0 and 1, respectively.

After the above updates, we remove all expired intervals, i.e., intervals $[p, q]$ in the list with $q \leq t_{\mathtt{cur}} - W$, where $t_{\mathtt{cur}}$ is the current time, i.e., the largest timestamp of items received so far.

Estimation by a λ-counter. Let $t_{\mathtt{cur}}$ be the current time and consider any time $t_{\mathtt{cur}} - W + 1 \leq t \leq t_{\mathtt{cur}}$. We estimate the number of item a in the interval $[t, t_{\mathtt{cur}}]$ by summing up the estimates of all intervals $[p_i, q_i]$ in the list of C_a such that $t \leq q_i$. That is, the estimate is

$$E_a(t) = \sum \{ e_a([p_i, q_i]) \mid [p_i, q_i] \text{ is in list of } C_a \text{ and } t \leq q_i \} \tag{2}$$

2.2 Analysis of the -Counter

Without loss of generality, we assume that the first item or query arrives no earlier than time W. We first state a fact which can be proved easily by induction.

Fact 1. *At any time $t_{\mathtt{cur}} \geq W$, suppose C_a contains a list as shown in* (1) *for some $k \geq 1$. Then, $p_1 \leq t_{\mathtt{cur}} - W + 1 \leq t_{\mathtt{cur}} \leq q_k$, and q_k is a multiple of W.*

It is not hard to see the memory requirement of a λ-counter C_a (Lemma 1), since $[p_1, q_k]$ has length $O(W)$ and the total number of item a in this interval is $O(B)$, where B is the maximum number of items with timestamp in a window.

Lemma 1. *The memory usage of a λ-counter C_a is $O(\frac{B}{\lambda})$ words.*

It is straightforward to see that each update takes $O(\log(\frac{B}{\lambda}))$ time (by binary search on the list) and each query takes $O(\frac{B}{\lambda})$ time. It remains to show that C_a gives estimates within the stated error bound. Suppose C_a contains a list as shown in (1) for some $k \geq 1$. We focus on some time $t \in [t_{\mathtt{cur}} - W + 1, t_{\mathtt{cur}}] \subseteq [p_1, q_k]$. For such t, there is a unique interval $I_c = [p_c, q_c]$ in the list of C_a such that $t \in I_c$; we call I_c the *critical interval* for t. We can check that Equation (2) is equivalent to

$$E_a(t) = \sum_{i=c,\ldots,k} e_a([p_i, q_i])$$

The following technical lemma analyzes how well $E_a(t)$ estimates $f_a([t, q_k])$. We say that an update is *critical* to t if it splits the critical interval for t (i.e., Case 2, and the extra step is executed on the critical interval for t). The proof of the lemma is a tedious verification of Inequality (3) and will be given in full paper.

Lemma 2. *Let $I = [p_1, q_k]$ be the interval monitored by C_a. Consider any time $t \in [t_{\mathtt{cur}} - W + 1, t_{\mathtt{cur}}]$. Let I_c be the critical interval for t and let n_c be the total number of updates made to C_a so far that are critical to t. Then,*

$$E_a(t) - \min\{e_a(I_c), 2\lambda\} - \lambda n_c \leq f_a([t, q_k]) \leq E_a(t) + 2\lambda n_c. \qquad (3)$$

We can apply Lemma 2 to obtain our main theorem, as follows. Note that if $W < 4$, we can use three counters to maintain the exact count of an item a.

Theorem 1. *For $W \geq 4$ and any $W' \leq W$, the estimate $E_a(t_{\mathtt{cur}} - W' + 1)$ returned by C_a has absolute error at most $2\lambda \log W$.*

Proof. Observe that no item with a timestamp larger than $t_{\mathtt{cur}}$ arrives yet, so $f_a([t_{\mathtt{cur}} - W' + 1, t_{\mathtt{cur}}]) = f_a([t_{\mathtt{cur}} - W' + 1, q_k])$. Then by Lemma 2, we have

$$E_a(t_{\mathtt{cur}} - W' + 1) - 2\lambda - \lambda n_c \leq f_a([t_{\mathtt{cur}} - W' + 1, t_{\mathtt{cur}}]) \leq E_a(t_{\mathtt{cur}} - W' + 1) + 2\lambda n_c$$

where n_c is the number of updates critical to $t_{\mathtt{cur}} - W' + 1$. Since $W \geq 4$, $2\lambda \leq \lambda \log W$. Note that $n_c \leq \log W$ because $t_{\mathtt{cur}} - W' + 1$ is initially in an interval of size at most W and each critical update reduces the size of the critical interval by half. Since the interval length is at least 1, the theorem follows. $\square$

3 $(Y,\ )$-Collection: Estimating the Frequency of Every Item When Total Frequency Is at Most Y

Let Y be a fixed constant and δ be an error bound. This section presents a data structure (Y, δ)-collection, which maintains an estimation of the frequency of any item within a sliding window of W time units. We will define a (Y, δ)-collection and show that the absolute error of an estimation is at most δY, when the total frequency of all items within the window at most Y. In the next section, we will extend the data structure to remove this restriction.

A (Y, δ)-collection is a collection of at most $\frac{24}{\delta}$ λ-counters where $\lambda = \frac{\delta Y}{10 \log W}$. Below we assume that $W \geq 4$.[2] Initially, the collection has no counter.

3.1 Updating a $(Y,\ )$-Collection

When an item (a, t) arrives, we update as follows.

Case 1. If the collection has a λ-counter C_a for a, we update C_a by (a, t). If after the update, we have either

- **(i)** the total number of intervals of all λ-counters in the collection is greater than $(\frac{82}{\delta}) \log W$, or
- **(ii)** the total estimate of all intervals of all λ-counters in the collection is greater than $\frac{58}{10} Y$,

then we find, from the lists of intervals of all λ-counters, the interval $[p, q]$ with the smallest q, and discard it from the collection.

Case 2. Suppose the collection does not have a λ-counter for a. If the total number of λ-counters is smaller than $\frac{24}{\delta}$, we create a new λ-counter for a, and update it by (a, t). Otherwise, we do nothing about (a, t). Instead, we perform a "decrement" operation to every λ-counter C_x in the collection:

Decrement of C_x. We find the last interval $[p, q]$ in C_x (i.e., the one with the largest q) with a positive estimate $e_x([p, q])$. Then, we decrease this estimate $e_x([p, q])$ by 1. If the estimate becomes zero, and q is not a multiple of W, and there is another interval $[q + 1, r]$ with estimate zero, then we combine the two intervals $[p, q]$ and $[q + 1, r]$ into one interval $[p, r]$ with estimate $e_x([p, r]) = 0$.

For ease of reference, we call this step a batch-of-decrement step. After it, we remove all λ-counters which do not have any interval with positive estimate.

By the update operation, we have some simple fact and observation about a (Y, δ)-collection. Consider any λ-counter C_a in the (Y, δ)-collection. Suppose that C_a is monitoring the list of intervals $\langle [p_1, q_1], [p_2, q_2], \ldots, [p_{k-1}, q_{k-1}], [p_k, q_k] \rangle$. The following fact is easy to verify.

Fact 2. *With respect to the λ-counter C_a, there are at most two intervals in its list with estimates smaller than λ.*

[2] If $W < 4$, we can replace each λ-counter for some item a with W counters to maintain the exact count of a in the window. Such (Y, δ)-collection using exact counters can be analyzed similarly to that using λ-counters.

In Case 1 of the update operation, we discard some interval if (i) the total number of intervals is greater than $(\frac{82}{\delta})\log W$ or (ii) the total estimate of all intervals is greater than $\frac{58}{10}Y$. The following lemma asserts that no "important" interval is discarded if the total frequency of all items within the window is at most Y. Let $I_{\mathsf{cur}} = [t_{\mathsf{cur}} - W + 1, t_{\mathsf{cur}}]$. Note that $\varSigma$ is the set of possible items and $f_*(I_{\mathsf{cur}}) = \sum_{a\in\varSigma} f_a(I_{\mathsf{cur}})$.

Lemma 3. *Suppose that we have discarded at or before* t_{cur} *an interval* $[p, q]$ *from the* (Y, δ)*-collection with* $q \geq t_{\mathsf{cur}} - W + 1$*. Then* $f_*(I_{\mathsf{cur}}) > Y$*.*

Proof. Consider the moment of the removal. We claim that

$$\sum_{C_a \text{ in the collection}} E_a(t_{\mathsf{cur}} - W + 1) > \tfrac{58}{10}Y.$$

It is obviously true if the removal is triggered by Case 1(ii). Suppose the removal is triggered by Case 1(i). Then there are totally more than $(\frac{82}{\delta})\log W$ intervals in the collection. Since we remove the interval $[p, q]$ with the smallest q, and $q \geq t_{\mathsf{cur}} - W + 1$, we conclude that all these intervals $[x, y]$ have $y \geq t_{\mathsf{cur}} - W + 1$. Together with the fact that there are at most $\frac{24}{\delta}$ λ-counters and each of them has at most two intervals with estimates smaller than λ (Fact 2), we conclude

$$\sum_{C_a \text{ in the collection}} E_a(t_{\mathsf{cur}} - W + 1) > \lambda(\tfrac{82}{\delta}\log W - 2(\tfrac{24}{\delta})) \geq \lambda(\tfrac{82}{\delta} - \tfrac{24}{\delta})\log W = \tfrac{58}{10}Y$$

If there is no batch-of-decrement step, by Theorem 1, the total estimate is at most

$$\sum_{C_a \text{ in the collection}} (f_a(I_{\mathsf{cur}}) + 2\lambda\log W) \leq f_*(I_{\mathsf{cur}}) + \tfrac{24}{\delta}(2\lambda\log W).$$

Otherwise, suppose there is a batch-of-decrement step. We first claim that Theorem 1 still holds. To see this, consider the decrement of some λ-counter C_x, which essentially deletes the latest item x in the stream. If the decrement combines two intervals $[p, q]$ and $[q + 1, r]$ into an interval $[p, r]$, then the number of critical update to any time $t \in [p, r]$ can be reset to 0 (because $e_a([p, r]) = 0$ and any item x with timestamps in $[p, r]$ has been deleted). Furthermore, the condition that q is not a multiple of W guarantees $[p, r]$ has size at most W. Thus, we still have Theorem 1. The decrement operations will only make the estimates smaller and thus the above inequality still holds even when there are decrements. Combining the above two inequalities and recall that $\lambda = \frac{\delta Y}{10\log W}$, the lemma follows. $\square$

3.2 Estimation by a $(Y,\ )$-Counter

Given any $W' \leq W$, let $I_{W'} = [t_{\mathsf{cur}} - W' + 1, t_{\mathsf{cur}}]$. The (Y, δ)-collection gives an estimate $\mathrm{Est}_a(I_{W'})$ of $f_a(I_{W'})$ for every item a as follows.

- If the (Y, δ)-collection has a λ-counter C_a for item a, then $\mathrm{Est}_a(I_{W'})$ is equal to the estimate $E_a(t_{\mathsf{cur}} - W' + 1)$ of C_a;
- Otherwise, $\mathrm{Est}_a(I_{W'}) = 0$.

We can analyze the accuracy of the estimate given by a (Y, δ)-collection, as follows. Note that $I_{\mathtt{cur}} = [t_{\mathtt{cur}} - W + 1, t_{\mathtt{cur}}]$.

Lemma 4. *Suppose that $f_*(I_{\mathtt{cur}}) \leq Y$. Then, for any item a and any $W' \leq W$, we have*

$$f_a(I_{W'}) - \delta Y \leq Est_a(I_{W'}) \leq f_a(I_{W'}) + \delta Y.$$

Proof. We first derive an upper bound on the number of batch-of-decrement steps performed during $I_{\mathtt{cur}}$. Note that at time $t_{\mathtt{cur}} - W$, the total estimate of all intervals in the (Y, δ)-collection is at most $\frac{58}{10}Y$. Together with the fact that $f_*(I_{\mathtt{cur}}) \leq Y$ items arrived during $I_{\mathtt{cur}}$, the total units that can be decreased by the batch-of-decrement steps performed during $I_{\mathtt{cur}}$ is at most $\frac{68}{10}Y$. Since each batch-of-decrement step decreases $\frac{24}{\delta} + 1$ units (each for a distinct item), we conclude that the number of batch-of-decrement steps performed during $I_{\mathtt{cur}}$ is at most $(\frac{68}{10}Y)/(\frac{24}{\delta} + 1) < \frac{68}{240}\delta Y$.

We can now analyze the accuracy of the estimate $\mathrm{Est}_a(I_{W'})$.

– If there is no counter in the (Y, δ)-collection for a, then $\mathrm{Est}_a(I_{W'}) = 0$. Note that $f_a(I_{W'})$ is at most $\frac{68}{240}\delta Y$ because we need a batch-of-decrement step to take away a unit for item a and there are at most $\frac{68}{240}\delta Y$ such steps. Thus, $|\mathrm{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \frac{68}{240}\delta Y < \delta Y$.
– Suppose that there is a λ-counter C_a for a. Since $f_*(I_{\mathtt{cur}}) \leq Y$, Lemma 3 asserts that no interval $[p, q]$ with $q \geq t_{\mathtt{cur}} - W + 1$ is removed. Thus, we can apply Equation (2) in Section 2.1 to determine $E_a(t_{\mathtt{cur}} - W + 1)$. If there is no batch-of-decrement step, by Theorem 1, we have

$$|\mathrm{Est}_a(I_{W'}) - f_a(I_{W'})| = |E_a(t_{\mathtt{cur}} - W' + 1) - f_a(I_{W'})| \leq 2\lambda \log W.$$

The decrement operations will introduce an additional error of at most $\frac{68}{240}\delta Y$ and thus we have

$|\mathrm{Est}_a(I_{W'}) - f_a(I_{W'})| \leq 2\lambda \log W + \frac{68}{240}\delta Y = 2(\frac{\delta Y}{10 \log W}) \log W + \frac{68}{240}\delta Y < \delta Y.$ □

4 Estimating the Frequency of Every Item

In this section, we build a data structure D_ε using (Y, δ)-collections and show that D_ε can give good estimates without any restriction on the total frequency. We estimate the frequency of any item within a sliding window of the last recent W' time units, for any $W' \leq W$.

Data structure D_ε. Let k be the smallest integer such that $2^k \geq \frac{\log W}{\varepsilon}$. The data structure D_ε comprises the following:
– a $(Y, \frac{\varepsilon}{4})$-collection for each $Y = 2^k, 2^{k+1}, \ldots, 2^h$ where $h = \lceil \log 4B \rceil$;
– the 2^k items with the largest timestamps among all items received so far;
– the data structure by Cormode *et al.* [3] which uses $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$ words of space and allows us to find an estimate $\mathrm{Est}_*(I_{W'})$ of $f_*(I_{W'})$ for any $W' \leq W$ such that

$$|\mathrm{Est}_*(I_{W'}) - f_*(I_{W'})| \leq \frac{\varepsilon}{4} f_*(I_{W'}).$$

The following theorem analyzes the accuracy of the estimate given by D_ε as well as the memory usage, update time and query time of D_ε. Recall that for any $W' \leq W$, $I_{W'} = [t_{\mathtt{cur}} - W' + 1, t_{\mathtt{cur}}]$.

Theorem 2. *Given any $W' \leq W$ at the query time, the data structure D_ε can give an estimate $Est_a(I_{W'})$ for any item a such that*

$$|Est_a(I_{W'}) - f_a(I_{W'})| \leq \varepsilon f_*(I_{W'}),$$

using $O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W}))$ words of space. Furthermore, an update upon the arrival of a new item takes $O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log\log W))$ time, and a query takes $O(\frac{1}{\varepsilon}\log W + \log\log(\varepsilon B))$ time.

Proof. To find an estimate of $f_a(I_{W'})$ for any item a, we first get an estimate $\mathrm{Est}_*(I_{W'})$ using the data structure in [3]. Let $Y^* = 2^i$ be the unique integer with $\frac{\mathrm{Est}_*(I_{W'})}{1-\varepsilon/4} \leq Y^* < \frac{2\mathrm{Est}_*(I_{W'})}{1-\varepsilon/4}$. Note that $f_*(I_{W'}) \leq \frac{\mathrm{Est}_*(I_{W'})}{1-\varepsilon/4} \leq Y^* \leq \frac{2\mathrm{Est}_*(I_{W'})}{1-\varepsilon/4} \leq \frac{2(1+\varepsilon/4)f_*(I_{W'})}{1-\varepsilon/4} \leq 4f_*(I_{W'}) \leq 4B$.

If $Y^* \leq 2^k$, then $f_*(I_{W'}) \leq Y^* \leq 2^k$ and we can obtain the exact value of $f_*(I_{W'})$ from the sequence containing at most 2^k items received thus far with the largest timestamps. Therefore, we have $\mathrm{Est}_a(I_{W'}) = f_a(I_{W'})$ and hence $|\mathrm{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \varepsilon f_*(I_{W'})$.

Now, suppose $Y^* > 2^k$. Since $f_*(I_{W'}) \leq Y^*$, by Lemma 4, we can find an estimate $\mathrm{Est}_a(I_{W'})$ using the $(Y^*, \frac{\varepsilon}{4})$-collection such that $|\mathrm{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \frac{\varepsilon}{4}Y^*$. Since $Y^* \leq 4f_*(I_{W'})$, we have $|\mathrm{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \varepsilon f_*(I_{W'})$.

Memory usage. A $(Y, \frac{\varepsilon}{4})$-collection has $O(\frac{4\log W}{\varepsilon})$ intervals, each of which together with its estimate needs $O(1)$ words. Since the number of $(Y, \frac{\varepsilon}{4})$-collections are $O(h-k) = O(\log(\frac{\varepsilon B}{\log W}))$, the total space used by the collections are $O(\frac{1}{\varepsilon}\log W \cdot \log(\frac{\varepsilon B}{\log W}))$ words. Together with the space of the data structure in [3], the total space is $O(\frac{1}{\varepsilon}\log W \log(\frac{\varepsilon B}{\log W}))$ words.

Update and Query time. Consider an update upon the arrival of a new item. For each $(Y, \frac{\varepsilon}{4})$-collection, if the new item causes a batch-of-decrement step, it takes $O(\frac{1}{\varepsilon})$ time. Otherwise, we need to update a λ-counter in the $(Y, \frac{\varepsilon}{4})$-collection, in which we need to update an interval in the list of the λ-counter. We can locate the interval to be updated using binary search; it takes $O(\log(\frac{1}{\varepsilon}\log W)) = O(\log(\frac{1}{\varepsilon}) + \log\log W)$ time. Since there are $O(\log(\frac{\varepsilon B}{\log W}))$ $(Y, \frac{\varepsilon}{4})$-collections, the update time required by the collections are $O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log\log W))$ time. Furthermore, it takes $O(\log(\frac{1}{\varepsilon}\log W))$ time to update the list of the 2^k items. Together with the update time of the data structure in [3], which is $O(\log(\frac{\varepsilon B}{\log W}) \cdot \log\log W)$, the total update time is $O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log\log W))$. Upon a query, it takes $O(1)$ time to identify which of $(Y, \frac{\varepsilon}{4})$-collection or the list of the 2^k items to be used for estimation; in either case, it takes $O(\frac{1}{\varepsilon}\log W)$ time to obtain the estimate. Together with the query time of the data structure in [3], which is $O(\frac{1}{\varepsilon}\log W + \log\log(\varepsilon B))$, the theorem follows. $\square$

5 Finding ε-Approximate -Frequent Item Set

In this section, we apply the data structure in Section 4 to find ε-approximate ϕ-frequent item set S within a sliding window covering the last recent W' time units for any window size $W' \leq W$ given at the query time. Recall that $I_{W'} = [t_{\mathtt{cur}} - W' + 1, t_{\mathtt{cur}}]$. The set S needs to satisfy the following two requirements:

(i) S contains every item a with $f_a(I_{W'}) \geq \phi f_*(I_{W'})$, and
(ii) For any item $b \in S$, $f_b(I_{W'}) \geq (\phi - \varepsilon) f_*(I_{W'})$.

To find such set, we maintain the following:

- Our data structure $D_{\frac{\varepsilon}{4}}$, which enables us to find, for any item a, an estimate $\mathrm{Est}_a(I_{W'})$ of $f_a(I_{W'})$ such that $|\mathrm{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \frac{\varepsilon}{4} f_*(I_{W'})$.
- The data structure by Cormode *et al.* [3] that enables us to find an estimate $\mathrm{Est}_*(I_{W'})$ of $f_*(I_{W'})$ such that $|\mathrm{Est}_*(I_{W'}) - f_*(I_{W'})| \leq \frac{\varepsilon}{4} f_*(I_{W'})$.

The following theorem suggests how to find an ε-approximate ϕ-frequent item set, and states the space and time complexity of the data structure.

Theorem 3. *Let $\epsilon \in (0, 1)$ be the error bound. With respect to the above data structure, given any threshold $\phi \in [\varepsilon, 1]$ and any window size $W' \leq W$ at the query time, the set*

$$S = \{a \mid Est_a(I_{W'}) \geq (\phi - \tfrac{\varepsilon}{2}) Est_*(I_{W'})\}$$

is an ε-approximate ϕ-frequent item set. It uses space $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$ words, an update takes $O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log\log W))$ time, and a query takes $O(\frac{1}{\varepsilon} \log W + \log\log(\varepsilon B))$ time.

Proof. By the estimate guarantees of the two data structures and the fact that $\phi \leq 1$, it can be verified that any item a with $f_a(I_{W'}) \geq \phi f_*(I_{W'})$ is in S, and any item $a \in S$ has $f_a(I_{W'}) \geq (\phi - \varepsilon) f_*(I_{W'})$. Thus, S is an ε-approximate ϕ-frequent item set. The space and time complexity follow directly from Theorem 2. □

6 Extension for a Stream with Bounded Tardiness

Recall that in an out-of-order data stream with *tardiness* $\tau \in [0, W]$, any item (a, d) arriving at time $t_{\mathtt{cur}}$ satisfies $d \geq t_{\mathtt{cur}} - \tau$; intuitively, it guarantees that the delay of any item is at most τ. In this section, we sketch the idea for extending our data structure to take advantage of this small delay guarantee to reduce the space requirement.

Modification to λ-counter. We say that an interval $[p, q]$ is *short* if its length is no more than $\tau + 1$ (i.e., $p - q \leq \tau$); otherwise it is *long*. To take advantage of the small delay guarantee, we make some minor modification to the implementation of λ-counter. Consider any λ-counter C_a. When an item (a, d) arrives at time $t_{\mathtt{cur}}$, we update C_a in exactly same way as in the original implementation except for the case: d belongs to some *long* interval $[p, q]$ in C_a and $e_a([p, q]) = 2\lambda - 1$. Then, we perform the update differently as follows:

Clean-up step. If $q > t_{\texttt{cur}}$, replace $[p, q]$ by the two intervals $[p, t_{\texttt{cur}}]$ and $[t_{\texttt{cur}} + 1, q]$, and set their estimates to $2\lambda - 1$ and 0, respectively.

Divide step. Let $s = \min\{q, t_{\texttt{cur}}\}$. If $p < t_{\texttt{cur}} - \tau$, we replace $[p, s]$ by $[p, t_{\texttt{cur}} - \tau - 1]$ and $[t_{\texttt{cur}} - \tau, s]$; otherwise, we replace $[p, s]$ by $[p, m]$ and $[m + 1, s]$ where $m = \lfloor \frac{p+s}{2} \rfloor$. Then, we set the estimates of the two new intervals to λ.

Note that if we add $[t_{\texttt{cur}} + 1, q]$ to the list of C_a in the clean-up step, then before the clean-up, $[p, q]$ must be the last interval in the list. The clean-up will not increase the error; for any $t \in [p, q]$, if $t \leq t_{\texttt{cur}}$, then $f_a([t, q]) = f_a([t, t_{\texttt{cur}}])$ and in our construction, $E_a(t)$ does not change. Furthermore, note that for any $t \in [p, q]$ where $t > t_{\texttt{cur}}$, t will be in the new interval $[t_{\texttt{cur}} + 1, q]$ after the clean-up, and in such case, the number of critical updates to t is reset to 0 (because $E_a(t) = f_a([t, q]) = 0$ and the estimate for this interval is accurate).

Lemma 5. *Suppose that the data stream has tardiness $\tau \in [0, W]$. With respect to the λ-counter, for any $W' \leq W$, our new update operation guarantees that $|E_a(t_{\texttt{cur}} - W' + 1) - f_a(I_{W'})| \leq 2\lambda(\log(\tau + 1) + 2)$.*

Proof. It can be verified that the modified implementation still satisfies Lemma 2. To prove the error bound, it suffices to prove that for any t, the number of updates that are critical to t is at most $(\log(\tau + 1) + 1)$.

Suppose that initially, $t \in [p, q]$. If $[p, q]$ are short, there will be at most $\log(\tau + 1)$ critical updates to t. Suppose $[p, q]$ is long. If $t > t_{\texttt{cur}}$, the clean-up step will add the new interval $R = [t_{\texttt{cur}} + 1, q]$, and the number of critical updates to t is reset to 0. In such case, we can treat it as if t was initially in R without any update yet, and we can repeat the argument by replacing $[p, q]$ by R.

Suppose $t \leq t_{\texttt{cur}}$. A divide step can split $[p, s]$ into two new intervals and t is in one of them, say $[x, y]$. It can be verified that either (i) $[x, y]$ is short, and there will be at most $\log(\tau + 1)$ additional critical updates to t, or (ii) $y < t_{\texttt{cur}} - \tau$ and there will be no more update to $[x, y]$ because the tardiness is τ. Thus, the total number of critical updates to t is at most $\log(\tau + 1) + 1$. □

Finding frequent item set. By setting $\lambda = \frac{\delta Y}{10(\log(\tau+1)+2)}$ and using the same analysis in previous sections, we have the following theorem.

Theorem 4. *There is a data structure that finds ε-approximate ϕ-frequent item set for data streams with tardiness τ using $O(\frac{1}{\varepsilon} \log \tau \log(\frac{\varepsilon B}{\log \tau}))$ words of space. Furthermore, an update takes $O(\log(\frac{\varepsilon B}{\log \tau})(\frac{1}{\varepsilon} + \log \log \tau))$ time and a query takes $O(\frac{1}{\varepsilon} \log \tau + \log \log(\varepsilon B))$ time.*

References

1. Arasu, A., Manku, G.: Approximate counts and quantiles over sliding windows. In: Proc. PODS, pp. 286–296 (2004)
2. Busch, C., Tirthapua, S.: A deterministic algorithm for summarizing asynchronous streams over a sliding window. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 465–476. Springer, Heidelberg (2007)

3. Cormode, G., Korn, F., Tirthapura, S.: Time-decaying aggregates in out-of-order streams. In: Proc. PODS, pp. 89–98 (2008)
4. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. SIAM Journal on Computing 31(6), 1794–1813 (2002)
5. Demaine, E., Lopez-Ortiz, A., Munro, J.: Frequency estimation of internet packet streams with limited space. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 348–360. Springer, Heidelberg (2002)
6. Karp, R., Shenker, S., Papadimitriou, C.: A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Systems 28(1), 51–55 (2003)
7. Lee, L.K., Ting, H.F.: Maintaining significant stream statistics over sliding windows. In: Proc. SODA, pp. 724–732 (2006)
8. Lee, L.K., Ting, H.F.: A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: Proc. PODS, pp. 290–297 (2006)
9. Misra, J., Gries, D.: Finding repeated elements. Science of Computer Programming 2(2), 143–152 (1982)
10. Shrivastava, N., Buragohain, C., Agrawal, D., Suri, S.: Medians and beyond: new aggregation techniques for sensor networks. In: Proc. SenSys, pp. 239–249 (2004)
11. Tirthapura, S., Xu, B., Busch, C.: Sketching asynchronous streams over a sliding window. In: PODC, pp. 82–91 (2006)

Longest Wait First for Broadcast Scheduling [Extended Abstract]

Chandra Chekuri[⋆], Sungjin Im[⋆⋆], and Benjamin Moseley[⋆⋆⋆]

Dept. of Computer Science, University of Illinois, Urbana, IL 61801
{chekuri,im3,bmosele2}@cs.illinois.edu

Abstract. We consider *online* algorithms for broadcast scheduling. In the pull-based broadcast model there are n unit-sized pages of information at a server and requests arrive online for pages. When the server transmits a page p, all outstanding requests for that page are satisfied. There is a lower bound of $\Omega(n)$ on the competitiveness of online algorithms to minimize average flow-time; therefore we consider resource augmentation analysis in which the online algorithm is given extra speed over the adversary. The *longest-wait-first* (LWF) algorithm is a natural algorithm that has been shown to have good empirical performance [2]. Edmonds and Pruhs showed that LWF is 6-speed $O(1)$-competitive using a novel yet complex analysis; they also showed that LWF is not $O(1)$-competitive with less than 1.618-speed. In this paper we make two main contributions to the analysis of LWF and broadcast scheduling.

- We give an intuitive and easy to understand analysis of LWF which shows that it is $O(1/\epsilon^2)$-competitive for average flow-time with $(4+\epsilon)$ speed.
- We show that a natural extension of LWF is $O(1)$-speed $O(1)$-competitive for more general objective functions such as average delay-factor and L_k norms of delay-factor (for fixed k). These metrics generalize average flow-time and L_k norms of flow-time respectively and ours are the first non-trivial results for these objective functions in broadcast scheduling.

1 Introduction

We consider online algorithms for broadcast scheduling in the pull-based model. In this model there are n pages (representing some form of useful information) available at a server and clients request a page that they are interested in. The server *broadcasts* pages according to some online policy and *all* outstanding requests for a page are satisfied when that page is transmitted/broadcast. This is what distinguishes this model from the standard scheduling models where the server has to process each request separately. Broadcast scheduling is motivated by several applications. Example situations where the broadcast assumption is

[⋆] Partially supported by NSF grants CCF-0728782 and CNS-0721899.
[⋆⋆] Partially supported by NSF grant CNS-0721899 and Samsung Fellowship.
[⋆⋆⋆] Partially supported by NSF grant CNS-0721899.

natural include wireless and satellite networks, LAN based systems and even some multicast systems. See [22,1,2] for pointers to applications and systems that are based on this model. In addition to their practical interest, broadcast scheduling has been of much interest in recent years from a theoretical point of view. There is by now a good amount of literature in online and offline algorithms in this model [6,2,1,7].

It is fair to say that algorithmic development and analysis for broadcast scheduling have been challenging even in the simplest setting of unit-sized pages; so much so that a substantial amount of technical work has been devoted to the development of *offline* approximation algorithms [20,15,16,17,3,4]; many of these offline algorithms are non-trivial and are based on linear programming based methods. Further, most of these offline algorithms, with the exceptions of [3,4], are in the resource augmentation model of Kalyanasundaram and Pruhs [19] in which the analysis is done by giving the algorithm a machine with speed $s > 1$ when compared to a speed 1 machine for the adversary. In this paper we are interested in *online* algorithms in the worst-case competitive analysis framework. We consider the problem of minimizing average flow-time (or waiting time) of requests and other more stringent objective functions. It is easy to show an $\Omega(n)$ lower bound on the competitive ratio [20] of any deterministic algorithm and hence we also resort to resource augmentation analysis. For average flow-time three algorithms are known to be $O(1)$-competitive with $O(1)$-speed. The first is the natural *longest-wait-first* (**LWF**) algorithm/policy: at any time t that the server is free, schedule the page p for which the total waiting time of all outstanding requests for p is the largest. Edmonds and Pruhs [13], in a complex and original analysis, showed that **LWF** is a 6-speed $O(1)$-competitive algorithm and also that it is not $O(1)$-competitive with a speed less than $(1+\sqrt{5})/2$; they also conjectured that their lowerbound is tight. The same authors also gave a different algorithm called BEQUI in [12] and show that it is a $(4+\epsilon)$-speed $O(1)$-competitive algorithm; although the algorithm has intuitive appeal, the proof of its performance relies on an reduction to an algorithm for a non-clairvoyant scheduling problem [11]. The recent improved result in [14] for the non-clairvoyant problem when combined with the reduction mentioned above leads to a $(2+\epsilon)$-speed $O(1)$-competitive algorithm. The preemptive algorithms in [12,14] are also applicable when the page sizes are arbitrary; see [18] for empirical evaluation in this model.

At a technical level, a main difficulty in online analysis for broadcast scheduling is the fact shown in [20] that no online algorithm can be *locally*-competitive with an adversary[1].

We focus on the **LWF** algorithm in the setting of unit-sized pages. In addition to being a natural greedy policy, it has been shown to outperform other natural policies [2]. It is, therefore, of interest to better understand its performance. We are motivated by the following questions. Is there a simpler and more intuitive

[1] An algorithm is locally-competitive if at each time t, its queue size is comparable to that of the queue size of the adversary. Many results in standard scheduling are based on showing local-competitiveness.

analysis of **LWF** for broadcast scheduling than the analysis presented in [13]? Can we close the gap between the upper and lower bounds on the speed requirement of **LWF** to guarantee constant competitiveness? Can we obtain competitive algorithms for more stringent objective functions than average flow-time such as L_k norms of flow-time, average delay-factor[2] and L_k norms of delay-factor? We give positive answers to these questions.

Results: Our results are for unit-size pages. We make two contributions.

– We give a simple and intuitive analysis of **LWF** that already improves the speed bound in [13]; the analysis shows that **LWF** is $(4 + \epsilon)$-speed $O(1/\epsilon^2)$-competitive for average flow time.
– We show that a natural generalization of **LWF** that we call **LF** is $O(k)$-speed $O(k)$-competitive for minimizing the L_k norm of flow time — these bounds extend to average delay factor and L_k norms of delay factor. These are the first non-trivial results for L_k norms in broadcast scheduling for $k > 1$.

L_k norms for flow-time for some small $k > 1$ such as $k = 2, 3$ have been suggested as alternate and robust metrics of performance; see [5,21] for more on this. Our results show that **LWF**-like algorithms have reasonable theoretical performance even for these more difficult metrics. We derive these additional results in a unified fashion via a general framework that is made possible by our simpler analysis for **LWF**. We note that the algorithms in [12,14] that perform well for average flow time do not easily extend to the more general objective functions that we consider.

Our analysis of **LWF** borrows several key ideas from [13], however, we make some crucial simplifications. We outline the main differences in Section 1.1 where we give a brief overview of our approach.

Notation and Formal Definitions: We assume that the server has n distinct unit-sized pages of information. We use $J_{p,i}$ to denote i'th request for a page $p \in \{1, \ldots, n\}$. We let $a_{p,i}$ denote the arrival time of the request $J_{p,i}$. The finish time $f_{p,i}$ of a request $J_{p,i}$ under a given schedule/algorithm is defined to be the earliest time after $a_{p,i}$ when the page p is sequentially transmitted by the scheduler; to avoid notational overload we assume that the algorithm is clear from the context. Note that multiple requests for the same page can have the same finish time. The total flow time for an algorithm over a sequence of requests is now defined as $\sum_p \sum_i (f_{p,i} - a_{p,i})$. Delay-factor is a recently introduced metric in scheduling [9,8,10]. In the context of broadcast scheduling, each request $J_{p,i}$ has a soft deadline $d_{p,i}$ that is known upon its arrival. The slack of $J_{p,i}$ is $d_{p,i} - a_{p,i}$. The delay-factor of $J_{p,i}$ with finish time $f_{p,i}$ is defined to be $\max(1, \frac{f_{p,i} - a_{p,i}}{d_{p,i} - a_{p,i}})$; in other words it is the ratio of the waiting time of the request to its slack. It can be seen that delay-factor generalizes flow-time since we can set $d_{p,i} = a_{p,i} + 1$ for each (unit-sized) request $J_{p,i}$. Given a scheduling metric such as flow-time or delay-factor that, for each schedule assigns a value $m_{p,i}$ to a request $J_{p,i}$, one

[2] Delay-factor is a recently introduced metric and we describe it more formally later.

can define the L_k norm of this metric in the usual way as $\sqrt[k]{\sum_{(p,i)} m_{p,i}^k}$. Note that minimizing the sum of flow-times or delay-factors is simply the L_1 norm problem. In resource augmentation analysis, the online algorithm is given a faster machine than the optimal offline algorithm. For $s \geq 1$, an algorithm A is s-speed r-competitive if A when the given s-speed machine achieves a competitive ratio of r.

In this paper we assume, for simplicity, the discrete time model. In this model, at each integer time t, the following things happen exactly in the following order; the scheduler make a decision of which page p to broadcast; the page p is broadcast and all outstanding requests of page p are immediately satisfied, thus having finish time t; new requests arrive. Note that new pages which arrive at t are not satisfied by the broadcasting at the time t. It is important to keep it in mind that all these things happen only at integer times. See [13] for more discussion on discrete time versus continuous time models. For the most part, we assume for simplicity of exposition, that the algorithm is given an integer speed s which implies that the algorithm schedules (at most) s requests in each time slot. For this reason we present our analysis for 5-speed, which can be easily extended to analysis for $(4 + \epsilon)$-speed, if continuous time model is assumed.

1.1 Overview of Analysis

We give a high level overview of our analysis of **LWF**. Let OPT denote some fixed optimal 1-speed offline solution; we overload notation and use OPT also to denote the value of the optimal schedule. Recall that for simplicity of analysis, we assume the discrete-time model in which requests arrive at integer times. For the same reason we analyze **LWF** with an integer speed $s > 1$. We can assume that **LWF** is never idle. Thus, in each time step **LWF** broadcasts s pages and the optimal solution broadcasts 1 page. We also assume that requests arrive at integer times. At time t, a request is in the set $U(t)$ if it is unsatisfied by the scheduler at time t. In the broadcast setting **LWF** with speed s is defined as the following.

Algorithm: $\mathbf{LWF}_s$

- At any integer time t, broadcast the s pages with the largest waiting times, where the waiting time of page p is $\sum_{J_{p,i} \in U(t)} (t - a_{p,i})$.

Our analysis of **LWF** is inspired by that in [13]. Here we summarize our approach and indicate the main differences from the analysis in [13]. Given the schedule of $\mathbf{LWF}_s$ on a request sequence σ, the requests are partitioned into two disjoint sets S (self-chargeable requests) and N (non-self-chargeable requests). Let the total flow time accumulated by $\mathbf{LWF}_s$ for requests in S and N be denoted by $\mathbf{LWF}_s^S$ and $\mathbf{LWF}_s^N$ respectively. Likewise, let OPT^S and OPT^N be the flow-time OPT accumulates for requests in S and N, respectively. S is the set of requests whose flow-time is comparable to their flow-time in OPT. Hence one immediately obtains that $\mathbf{LWF}_s^S \leq \rho \mathrm{OPT}^S$ for some constant ρ. For requests in N, instead of charging them only to the optimal solution, these requests are charged to the total flow time accumulated by **LWF** *and* OPT. It will be shown that $\mathbf{LWF}_s^N \leq \delta\mathbf{LWF}_s + \rho\mathrm{OPT}^N$ for some $\delta < 1$; this is crux of the proof. It follows that $\mathbf{LWF}_s = \mathbf{LWF}_s^S + \mathbf{LWF}_s^N \leq \rho\mathrm{OPT}^S + \rho\mathrm{OPT}^N + \delta\mathbf{LWF}_s \leq \rho\mathrm{OPT} + \delta\mathbf{LWF}_s$. This shows that $\mathbf{LWF}_s \leq \frac{\rho}{1-\delta}\mathrm{OPT}$, which will complete our analysis. Perhaps the key idea in [13] is the idea of charging $\mathbf{LWF}_s^N$ to $\mathbf{LWF}_s$ with a $\delta < 1$; as shown in [20], no algorithm for any constant speed can be locally competitive with respect to all adversaries and hence previous approaches in the non-broadcast scheduling context that establish local competitiveness with respect to OPT cannot work.

In [13], the authors do not charge $\mathbf{LWF}_s^N$ directly to $\mathbf{LWF}_s$. Instead, they further split N into two types and do a much more involved analysis to bound the flow-time of the type 2 requests via the flow-time of type 1 requests. Moreover, they first transform the given instance to canonical instance in a complex way and prove the correctness of the transformation. Our simple proof shows that these complex arguments can be done away with. We also improve the speed bounds and generalize the proof to other objective functions.

1.2 Preliminaries

To show that $\mathbf{LWF}_s^N \leq \delta\mathbf{LWF}_s + \rho\mathrm{OPT}^N$, we will map the requests in N to other requests scheduled by $\mathbf{LWF}_s$ which have comparable flow time. An issue that can occur when using a charging scheme is that one has to be careful not to overcharge. In this setting, this means for a single request $J_{p,i}$ we must bound of the number of requests in N which are charged to $J_{p,i}$. To overcome the overcharging issue, we will appeal to a generalization of Hall's theorem. Here we will have a bipartite graph $G = (X \cup Y)$ where the vertices in X will correspond to requests in N. The vertices in Y will correspond to all requests scheduled by $\mathbf{LWF}_s$. A vertex $u \in X$ will be adjacent to a vertex $v \in Y$ if u and v have comparable flow time and v was satisfied while u was in our queue and unsatisfied; that is, u can be charged to v. We then use a simple generalization of Hall's theorem, which we call *Fractional Hall's Theorem.* Here a vertex of $u \in X$ is matched to a vertex of $v \in Y$ with weight $\ell_{u,v}$ where $\ell_{u,v}$ is not necessarily an integer. Note that a vertex can be matched to multiple vertices.

Definition 1 (c-covering). *Let $G = (X \cup Y, E)$ be a bipartite graph whose two parts are X and Y, and let $\ell : E \rightarrow [0, 1]$ be an edge-weight function. We say that ℓ is a c-covering if for each $u \in X$, $\sum_{(u,v)\in E} \ell_{u,v} = 1$ and for each $v \in Y$, $\sum_{(u,v)\in E} \ell_{u,v} \leq c$.*

The following lemma follows easily from either Hall's Theorem or the Max-Flow Min-Cut Theorem.

Lemma 1 (Fractional Hall's theorem). *Let $G = (V = X \cup Y, E)$ be a bipartite graph whose two parts are X and Y, respectively. For a subset S of X, let $N_G(S) = \{v \in Y | uv \in E,\ u \in S\}$, be the neighborhood of S. For every $S \subseteq X$, if $|N_G(S)| \geq \frac{1}{c}|S|$, then there exists a c-covering for X.*

Throughout this paper we will discuss time intervals and unless explicitly mentioned we will assume that they are closed intervals with integer end points. When considering some contiguous time interval $I = [s, t]$ we will say that $|I| = t - s + 1$ is the length of interval I; in other words it is the number of integers in I. For simplicity, we abuse this notation; when X is a set of closed intervals, we let $|X|$ denote the number of distinct integers in some interval of X. Note that $|X|$ also can be seen as the sum of the lengths of maximal contiguous sub-intervals if X is composed of non-overlapping intervals.

To be able to apply Lemma 1, we show another lemma which will be used throughout this paper. Lemma 2 says that the union of some fraction of time intervals is comparable to that of the whole time interval.

Lemma 2. *Let* $0 \leq \lambda \leq 1$ *be a constant. Let* $X = \{[s_1, t_1], \ldots, [s_k, t_k]\}$ *be a finite set of closed intervals and let* $X' = \{[s'_1, t_1], \ldots, [s'_k, t_k]\}$ *be an associated set of intervals such that for* $1 \leq i \leq k$, $s'_i \in [s_i, t_i]$ *and* $|[s'_i, t_i]| \geq \lambda |[s_i, t_i]|$. *Then* $|X'| \geq \lambda |X|$.

2 Minimizing Average Flow Time

We focus our attention to minimizing average flow time. A fair amount of notation is needed to clearly illustrate our ideas. Following [13], for each page, we will partition time into intervals via *events.* Events for page p are defined by $\mathbf{LWF}_s$'s broadcasts of page p. When $\mathbf{LWF}_s$ broadcasts page p a new event occurs. An event x for page p will be defined as $E_{p,x} = \langle b_{p,x}, e_{p,x} \rangle$ where $b_{p,x}$ is the beginning of the event and $e_{p,x}$ is the end. Here $\mathbf{LWF}_s$ broadcast page p at time $b_{p,x}$ and this is the xth broadcast of page p. Then $\mathbf{LWF}_s$ broadcast page p at time $e_{p,x}$ and this is the $(x+1)$st broadcast of page p. This starts a new event $E_{p,x+1}$. Therefore, the algorithm $\mathbf{LWF}_s$ does not broadcast p on the time interval $[b_{p,x} + 1, e_{p,x} - 1]$. Thus, it can be seen that for page p, $e_{p,x-1} = b_{p,x}$. It is important to note that the optimal offline solution may broadcast page p multiple (or zero) times during an event for page p. For each event $E_{p,x}$ we let $\mathcal{J}_{p,x} = \{(p, i) \mid a_{p,i} \in [b_{p,x}, e_{p,x} - 1]\}$ denote the set of requests for p that arrive in the interval $[b_{p,x}, e_{p,x} - 1]$ and are satisfied by $\mathbf{LWF}_s$ at $e_{p,x}$. We let $F_{p,x}$ denote the flow-time in $\mathbf{LWF}_s$ of all requests in $\mathcal{J}_{p,x}$. Similarly we define $F^*_{p,x}$ to be flow time in OPT for all requests in $\mathcal{J}_{p,x}$. Note that OPT may or may not satisfy requests in $\mathcal{J}_{p,x}$ during the interval $[b_{p,x}, e_{p,x}]$.

An event $E_{p,x}$ is said to be self-chargeable and in the set S if $F_{p,x} \leq F^*_{p,x}$ or $e_{p,x} - b_{p,x} < \rho$, where $\rho > 1$ is a constant which will be fixed later. Otherwise the event is non-self-chargeable and is in the set N. Implicitly we are classifying the requests as self-chargeable or non-self-chargeable, however it is easier to work with events rather than individual requests. As the names suggest, self-chargeable events can be easily charged to the flow-time of an optimal schedule. To help analyze the flow-time for non-chargeable events, we set up additional notation and further refine the requests in N.

Consider a non-self-chargeable event $E_{p,x}$. Note that since this event is non-self-chargeable, the optimal solution must broadcast page p during the interval

$[b_{p,x}+1, e_{p,x}-1]$; otherwise, $F_{p,x} \leq F^*_{p,x}$ and the event is self-chargeable. Let $o_{p,x}$ be the last broadcast of page p by the optimal solution during the interval $[b_{p,x}+1, e_{p,x}-1]$. We define $o'_{p,x}$ for a non-self-chargeable event $E_{p,x}$ as $\min\{o_{p,x}, e_{p,x}-\rho\}$. This ensures that the interval $[o'_{p,x}, e_{p,x}]$ is sufficiently long; this is for technical reasons and the reader should think of $o'_{p,x}$ as essentially the same as $o_{p,x}$.

Let $\mathbf{LWF}^S_s = \sum_{p,x:E_{p,x}\in S} F_{p,x}$ and $\mathbf{LWF}^N_s = \sum_{p,x:E_{p,x}\in N} F_{p,x}$ denote the the total flow time for self-chargeable and non self-chargeable events respectively. Similarly, let $\mathrm{OPT}^S = \sum_{p,x:E_{p,x}\in S} F^*_{p,x}$ and $\mathrm{OPT}^N = \sum_{p,x:E_{p,x}\in N} F^*_{p,x}$. For a non-chargeable event $E_{p,x}$ we divide $\mathcal{J}_{p,x}$ into early requests and late requests depending on whether the request arrives before $o'_{p,x}$ or not. Letting $F^e_{p,x}$ and $F^l_{p,x}$ denote the flow-time of early and late requests respectively, we have $F_{p,x} = F^e_{p,x} + F^l_{p,x}$. Let $\mathbf{LWF}^{N^e}_s$ and $\mathbf{LWF}^{N^l}_s$ denote the total flow time of early and late requests of non-self-chargeable events for **LWF**'s schedule, respectively.

The following two lemmas follow easily from the definitions.

Lemma 3. $\mathbf{LWF}^S_s \leq \rho\mathrm{OPT}^S$.

Lemma 4. $\mathbf{LWF}^{N^l}_s \leq \rho\mathrm{OPT}^N$.

Thus the main task is to bound $\mathbf{LWF}^{N^e}_s$. For a non-chargeable event $E_{p,x}$ we try to charge $F^e_{p,x}$ to events ending in the interval $[o'_{p,x}, e_{p,x}-1]$. The lemma below quantifies the relationship between $F^e_{p,x}$ and the flow-time of events ending in this interval.

Lemma 5. *For any* $0 \leq \lambda \leq 1$, *if* $e_{q,y} \in [\lceil o'_{p,x} + \lambda(e_{p,x} - o'_{p,x})\rceil, e_{p,x}-1]$ *then* $F_{q,y} \geq \lambda F^e_{p,x}$.

Proof. Let $F_{p,x}(t)$ be the total waiting time accumulated by **LWF** for page p on the time interval $[b_{p,x}, t]$. We divide $F_{p,x}(t)$ into two parts $F^e_{p,x}(t)$ and $F^l_{p,x}(t)$, which are the flow time due to early requests and to late requests, respectively. Note that $F_{p,x}(t) = F^e_{p,x}(t) + F^l_{p,x}(t)$. The early requests arrived before time $o'_{p,x}$, thus, for any $t' \geq \lceil o'_{p,x} + \lambda(e_{p,x} - o'_{p,x})\rceil$, $F^e_{p,x}(t') \geq \lambda F^e_{p,x}(e_{p,x}) = \lambda F^e_{p,x}$.

Since $\mathbf{LWF}_s$ chose to transmit q at $e_{q,y}$ when p was available to be transmitted, it must be the case that $F_{q,y} \geq F_{p,x}(e_{q,y}) \geq F^e_{p,x}(e_{q,y})$. Combining this with the fact that $F^e_{p,x}(e_{q,y}) \geq \lambda F^e_{p,x}$, the lemma follows. □

With the above setup in place, we now prove that $\mathbf{LWF}_s$ is $O(1)$ competitive for $s = 5$ via a clean and simple proof. This proof can be easily extended to non-integer speeds, showing that **LWF** is $(4+\epsilon)$-speed $O(1/\epsilon^2)$-competitive.

2.1 Analysis of 5-Speed

This section will be devoted to proving the following main lemma that bounds the flow-time of early requests of non self-chargeable events.

Lemma 6. *For* $\rho \geq 1$, $\mathbf{LWF}^{N^e}_5 \leq \frac{4\rho}{5(\rho-1)}\mathbf{LWF}_5$.

Assuming the lemma, $\mathbf{LWF}_5$ is $O(1)$-competitive, using the argument outlined earlier in Section 1.1.

Theorem 1. $\mathbf{LWF}_5 \leq 90\mathrm{OPT}$.

Proof. By combining Lemma 3, 4 and 6, we have that $\mathbf{LWF}_5 = \mathbf{LWF}_5^S + \mathbf{LWF}_5^{N^l} + \mathbf{LWF}_5^{N^e} \leq \rho\mathrm{OPT}^S + \rho\mathrm{OPT}^N + \frac{4\rho}{5(\rho-1)}\mathbf{LWF}_5$. Setting $\rho = 10$ completes the proof. □

We now prove Lemma 6. In the analysis, we assume that **LWF** broadcasts 5 pages at each time; otherwise we can apply the same argument to maximal subintervals when **LWF** is fully busy, respectively. Let $E_{p,x} \in N$. We define two intervals $I_{p,x} = [o'_{p,x}, e_{p,x} - 1]$ and $I'_{p,x} = [o'_{p,x} + \lceil (e_{p,x} - o'_{p,x})/2 \rceil, e_{p,x} - 1]$. Since $\rho \leq e_{p,x} - o'_{p,x}$, it follows that $|I'_{p,x}| \geq \frac{\rho-1}{2\rho}|I_{p,x}|$. We wish to charge $F^e_{p,x}$ to events (could be in S or N) in the interval $I'_{p,x}$. By Lemma 5, each event $E_{q,y}$ that finishes in $I'_{p,x}$ satisfies the property that $F_{q,y} \geq F^e_{p,x}/2$. Moreover, there are $5(\lfloor e_{p,x} - o'_{p,x})/2 \rfloor$ such events to charge to since $\mathbf{LWF}_5$ transmits 5 pages in each time slot. Thus, locally for $E_{p,x}$ there are enough events to charge to if ρ is a sufficiently large constant. However, an event $E_{q,y}$ with $e_{q,y} \in I'_{p,x}$ may also be charged by many other events if we follow this simple local charging scheme. To overcome this overcharging, we resort to a global charging scheme by setting up a bipartite graph G and invoking the fractional Hall's theorem (see Lemma 1) on this graph.

The bipartite graph $G = (X \cup Y, E)$ is defined as follows. There is exactly one vertex $u_{p,x} \in X$ for each non-self-chargeable event $E_{p,x} \in N$ and there is exactly one vertex $v_{q,y} \in Y$ for each event $E_{q,y} \in A$, where A is the set of all events. Consider two vertices $u_{p,x} \in X$ and $v_{q,y} \in Y$. There is an edge $u_{p,x}v_{q,y} \in E$ if and only if $e_{q,y} \in I'_{p,x}$. By Lemma 5, if there is an edge between $u_{p,x} \in X$ and $v_{q,y} \in Y$ then $F_{q,y} \geq F^e_{p,x}/2$.

The goal is now to show that G has a $\frac{2\rho}{5(\rho-1)}$-covering. Consider any non-empty set $Z \subseteq X$ and a vertex $u_{p,x} \in Z$. Recall that the interval $I_{p,x}$ contains at least one broadcast by OPT of page p. Let $\mathcal{I} = \bigcup_{u_{p,x} \in Z} I_{p,x}$ be the union of the time intervals corresponding to events in Z. Similarly, define $\mathcal{I}' = \bigcup_{u_{p,x} \in Z} I'_{p,x}$.

We claim that $|Z| \leq |\mathcal{I}|$. This is because the optimal solution has 1-speed and it has to do a separate broadcast for each event in Z during $\mathcal{I}$. Now consider the neighborhood of Z, $N_G(Z)$. We note that $|N_G(Z)| = 5|\mathcal{I}'|$ since $\mathbf{LWF}_5$ broadcasts 5 pages for each time slot in $|\mathcal{I}'|$ and each such broadcast is adjacent to an event in Z from the definition of G. From Lemma 2, $|\mathcal{I}'| \geq \frac{\rho-1}{2\rho}|\mathcal{I}|$ as we had already observed that $|I'_{p,x}| \geq \frac{\rho-1}{2\rho}|I_{p,x}|$ for each $E_{p,x} \in N$. Thus we conclude that $|N_G(Z)| = 5|\mathcal{I}'| \geq 5\frac{\rho-1}{2\rho}|\mathcal{I}| \geq 5\frac{\rho-1}{2\rho}|Z|$. Since this holds for $\forall Z \subseteq X$, by Lemma 1, there must exist a $\frac{2\rho}{5(\rho-1)}$-covering. Let ℓ be such a covering. Finally, we prove that the covering implies the desired bound on $\mathbf{LWF}_5^{N^e}$.

$$\begin{aligned}
\mathbf{LWF}_5^{N^e} &= \sum_{u_{p,x}\in X} F^e_{p,x} \text{ [By Definition]} \\
&= \sum_{u_{p,x}v_{q,y}\in E} \ell_{u_{p,x},v_{q,y}} F^e_{p,x} \text{ [By Def. 1, i.e. for } \forall u_{p,x}\in X,\ \textstyle\sum_{v_{q,y}\in Y} \ell_{u_{p,x},v_{q,y}} = 1] \\
&\le \sum_{u_{p,x}v_{q,y}\in E} \ell_{u_{p,x},v_{q,y}} 2F_{q,y} \text{ [By Lemma 5]} \\
&\le \frac{4\rho}{5(\rho-1)} \sum_{v_{q,y}\in Y} F_{q,y} \text{ [Change order of } \textstyle\sum \text{ and } \ell \text{ is a } \frac{2\rho}{5(\rho-1)}\text{-covering]} \\
&\le \frac{4\rho}{5(\rho-1)} \mathbf{LWF}_5. \text{ [Since } Y \text{ includes all events]}
\end{aligned}$$

This finishes the proof of Lemma 6.

3 Generalization to Delay-Factor and L_k Norms

In this section, our proof techniques are extended to show that a generalization of **LWF** is $O(1)$-speed $O(1)$-competitive for minimizing the average delay-factor and minimizing the L_k-norm of the delay-factor. Recall that flow-time can be subsumed as a special case of delay-factor. Thus, these results will also apply to L_k norms of flow-time. Instead of focusing on specific objective functions, we develop a general framework and derive results for delay-factor and L_k norms as special cases. First, we set up some notation. We assume that for each request $J_{p,i}$ there is a non-decreasing function $m_{p,i}(t)$ that gives the cost/penalty of that $J_{p,i}$ accumulates if it has *waited* for a time of t units after its arrival. Thus the total cost/penalty incurred for a schedule that finishes $J_{p,i}$ at $f_{p,i}$ is $m_{p,i}(f_{p,i} - a_{p,i})$. For flow-time $m_{p,i}(t) = t$ while for delay-factor it is $\max(1, \frac{t-a_{p,i}}{d_{p,i}-a_{p,i}})$. For L_k norms of delay-factor we set $m_{p,i}(t) = \max(1, \frac{t-a_{p,i}}{d_{p,i}-a_{p,i}})^k$. Note that the L_k norm of delay-factor for a given sequence of requests is $\sqrt[k]{\sum_{p,i} m_{p,i}(f_{p,i} - a_{p,i})}$ but we can ignore the outer k'th root by focusing on the inner sum.

A natural generalization of **LWF** to more general metrics is described below; we refer to this (greedy) algorithm as **LF** for Longest First. We in fact describe $\mathbf{LF}_s$ which is given s speed over the adversary.

Algorithm: $\mathbf{LF}_s$

– At any integer time t, broadcast the s pages with the largest m-waiting times where the m-waiting time of page p at t is $\sum_{J_{p,i}\in U(t)} m_{p,i}(t - a_{p,i})$.

Remark 1. The algorithm and analysis do not assume that the functions $m_{p,i}$ are "uniform" over requests. In principle each request $J_{p,i}$ could have a different penalty function.

In order to analyze **LF**, we need a lower bound on the "growth" rate of the functions $m_{p,i}()$. In particular we assume that there is a function $h : [0, 1] \to \mathbb{R}^+$ such that $m_{p,i}(\lambda t) \geq h(\lambda)m_{p,i}(t)$ for all $\lambda \in [0, 1]$. It is not to difficult to see that for flow-time and delay-factor we can choose $h(\lambda) = \lambda$, and for L_k norms of flow-time and delay-factor, we can set $h(\lambda) = \lambda^k$. We also define a function $m : \mathbb{R}^+ \to \mathbb{R}^+$ as $m(x) = \max_{(p,i)} m_{p,i}(x)$. The rest of the analysis depends only on h and m.

In the following subsection we outline a generalization of the analysis from Section 2.1 that applies to $\mathbf{LF}_s$; the analysis bounds various quantities in terms of the functions $h()$ and $m()$. In Section 3.2, we derive the results for minimizing delay-factor and L_k norms of delay-factor.

3.1 Outline of Analysis

To bound the competitiveness of $\mathbf{LF}_s$, we use the same techniques we used for bounding the competitiveness of $\mathbf{LWF}_s$. Events are again defined in the same fashion; $E_{p,x}$ is the event defined by the x'th transmission of p by $\mathbf{LF}_s$. We again partition events into self-chargeable and non self-chargeable events and charge self-chargeable events to the optimal value and charge non-self-chargeable events to $\delta\mathbf{LF}_s + m(\rho)\mathrm{OPT}^N$ for some $\delta < 1$. For an event $E_{p,x}$, let $M_{p,x}(t) = \sum_{J_{p,i} \in \mathcal{J}_{p,x}} m_{p,i}(t - a_{p,i})$ denote the total m-cost of all requests for p that arrive in $[b_{p,x}, e_{p,x} - 1]$ that are satisfied at $e_{p,x}$. We let $M^*_{p,x}(t)$ be the m-cost of the same set of requests for the optimal solution. An event $E_{p,x}$ is self-chargeable if $M_{p,x} \leq m(\rho)M^*_{p,x}$ or $e_{p,x} - b_{p,x} \leq \rho$ for some constant ρ to be optimized later. The remaining events are non self-chargeable. Again, requests for non-self-chargeable events are divided into early requests and late requests based on whether they arrive before $o'_{p,x}$ or not where $o'_{p,x} = \min\{o_{p,x}, e_{p,x} - \rho\}$. Let $M^e_{p,x}$ and $M^l_{p,x}$ be the flow time accumulated for early and late requests of a non-self-chargeable event $E_{p,x}$, respectively. The values of $\mathbf{LF}^N_s$, $\mathbf{LF}^{N^l}_s$, $\mathbf{LF}^{N^e}_s$, and $\mathbf{LF}^S_s$ are defined in the same way as $\mathbf{LWF}^N_s$, $\mathbf{LWF}^{N^l}_s$, $\mathbf{LWF}^{N^e}_s$, and $\mathbf{LWF}^S_s$. Likewise for OPT. The following two lemmas are analogues of Lemmas 3 and 4 and follow from definitions.

Lemma 7. $\mathbf{LF}^S_s \leq m(\rho)\mathrm{OPT}^S$.

Lemma 8. $\mathbf{LF}^{N^l}_s \leq m(\rho)\mathrm{OPT}^N$.

We now show a generalization of Lemma 5 that states that any event $E_{q,y}$ such that $e_{q,y}$ is close to $e_{p,x}$ has m-waiting time comparable to the m-waiting time of early requests of $E_{p,x}$.

Lemma 9. *Suppose $E_{p,x}$ and $E_{q,y}$ are two events such that $e_{q,y} \in [\lceil o'_{p,x} + \lambda(e_{p,x} - o'_{p,x})\rceil, e_{p,x} - 1]$, $M_{q,y} \geq h(\lambda)M^e_{p,x}$.*

Proof (Sketch). Consider an early request $J_{p,i}$ in $\mathcal{J}_{p,x}$ and let $t \in [\lceil o'_{p,x} + \lambda(e_{p,x} - o'_{p,x})\rceil, e_{p,x} - 1]$. Since $a_{p,i} \leq o'_{p,x}$, it follows that $t \geq \lambda(e_{p,x} - a_{p,i}) + a_{p,i}$. Hence,

$m_{p,i}(t - a_{p,i}) \geq h(\lambda) m_{p,i}(e_{p,x} - a_{p,i})$. Summing over all early requests, it follows that $M^e_{p,x}(t) \geq h(\lambda) M^e_{p,x}$. Since $\mathbf{LF}_s$ chose to transmit q at $t = e_{q,y}$ instead of p, it follows that $M_{q,y} \geq M_{p,x}(e_{q,y}) \geq M^e_{p,x}(e_{q,y}) \geq h(\lambda) M^e_{p,x}$. □

As in Section 2.1, the key ingredient of the analysis is to bound the waiting time of early requests. We state the analogue of Lemma 6 below. Observe that we have an additional parameter β. In Lemma 6 we hard wire β to be 1/2 to simplify the exposition. In the more general setting, the parameter β needs to be tuned based on h.

Lemma 10. *For any* $0 < \beta < 1$, $\mathbf{LF}_s^{N^e} \leq \frac{\rho}{sh(\beta)(\rho(1-\beta)-1)} \mathbf{LF}_s$, *where h is some scaling function for m.*

The proof of the above lemma follows essentially the same lines as that of Lemma 6. The idea is to charge $M^e_{p,x}$ to events in the interval $[o'_{p,x} + \lceil \beta(e_{p,x} - o'_{p,x}) \rceil, e_{p,x} - 1]$. Using Lemma 9, each event in this interval is within a factor of $h(\lambda)$ of $M^e_{p,x}$. The length of this interval is at least $\frac{\rho(1-\beta)-1}{\rho}$ times the length of the interval $[o'_{p,x}, e_{p,x} - 1]$. To avoid overcharging we again resort to the global scheme using fractional Hall's theorem after we setup the bipartite graph. We can then prove the existence of a $\frac{\rho}{s(\rho(1-\beta)-1)}$-covering and since each event can pay to within a factor of $h(\beta)$, the lemma follows.

Putting the above lemmas together we derive the following theorem.

Theorem 2. *Let* $\beta \in (0, 1)$ *and* $\rho > 1$ *be given constants. If s is an integer such that* $\frac{\rho}{sh(\beta)(\rho(1-\beta)-1)} \leq \delta < 1$, *then algorithm* $\mathbf{LF}_s$ *is s-speed* $\frac{m(\rho)}{1-\delta}$*-competitive.*

3.2 Results for Delay-Factor and L_k Norms

We can apply Theorem 2 with appropriate choice of parameters to show that $\mathbf{LF}_s$ is $O(1)$-competitive with $O(1)$ speed.

For minimizing average delay-factor we have $h(\lambda) = \lambda$ and $m(x) \leq x$. For this reason, average delay-factor behaves essentially the same as average flow-time and we can carry over the results from flow-time.

Theorem 3. *The algorithm* **LF** *is* 5*-speed* $O(1)$ *competitive for minimizing the average delay-factor.*

We note that **LF** can be shown to be $(4 + \epsilon)$-speed $O(1/\epsilon^2)$-competitive in continuous time model.

For L_k norms of delay-factor we have $h(\lambda) = \lambda^k$ and $m(x) \leq x^k$. By choosing $\beta = \frac{k}{k+1}$, $\rho = 90(k + 1)$ and $s = 3(k + 1)$ in Theorem 2, we can show that the algorithm **LF** is $3(k+1)$-speed $O(\rho^k)$-competitive for minimizing $\sum_{p,i} m_{p,i}(f_{p,i} - a_{p,i})$. Thus for minimizing the L^k-norm delay factor, we obtain $\sqrt[k]{O(\rho^k)} = O(\rho)$ competitiveness, which shows the following.

Theorem 4. *For* $k \geq 1$, *the algorithm* **LF** *is* $O(k)$*-speed* $O(k)$*-competitive for minimizing* L_k*-norm of delay-factor.*

4 Conclusions

We gave a simpler analysis of **LWF** for minimizing average flow-time in broadcast scheduling. This not only helps improve the speed bound but also results in extending the algorithm and analysis to more general objective functions such as delay-factor and L_k norms of delay-factor. We hope that our analysis is useful in other scheduling contexts.

Using a more involved analysis, it can be shown that **LWF** is $O(1/\epsilon^3)$-competitive for average flow-time with $(3.4+\epsilon)$ speed. Likewise, **LF** for average delay factor. Recently we have shown that for any fixed $\epsilon > 0$, **LF** is not $O(1)$-competitive with speed $(k+1-\epsilon)$ for minimizing L_k norm of flow time. For $k = 1$ this implies that LWF is not $O(1)$-competitive with $(2-\epsilon)$-speed and disproves the conjecture from [13]. We believe that **LWF** is $O(1)$-competitive with 2-speed. It would be interesting to find an $O(1)$-speed $O(1)$-competitive algorithm for L_k norm of flow time where the speed and competitive ratio are independent of k.

Acknowledgments. We thank Kirk Pruhs for his helpful comments and encouragement.

References

1. Acharya, S., Franklin, M., Zdonik, S.: Dissemination-based data delivery using broadcast disks. IEEE Pers. Commun. 2(6), 50–60 (1995)
2. Aksoy, D., Franklin, M.J.: rxw: A scheduling approach for large-scale on-demand data broadcast. IEEE/ACM Trans. Netw. 7(6), 846–860 (1999)
3. Bansal, N., Charikar, M., Khanna, S., Naor, J.S.: Approximating the average response time in broadcast scheduling. In: SODA, pp. 215–221 (2005)
4. Bansal, N., Coppersmith, D., Sviridenko, M.: Improved approximation algorithms for broadcast scheduling. In: SODA, pp. 344–353 (2006)
5. Bansal, N., Pruhs, K.: Server scheduling in the l_p norm: a rising tide lifts all boat. In: STOC, pp. 242–250 (2003)
6. Bar-Noy, A., Bhatia, R., Naor, J.S., Schieber, B.: Minimizing service and operation costs of periodic scheduling. Math. Oper. Res. 27(3), 518–544 (2002)
7. Bartal, Y., Muthukrishnan, S.: Minimizing maximum response time in scheduling broadcasts. In: SODA, pp. 558–559 (2000)
8. Bender, M.A., Clifford, R., Tsichlas, K.: Scheduling algorithms for procrastinators. J. Scheduling 11(2), 95–104 (2008)
9. Chang, J., Erlebach, T., Gailis, R., Khuller, S.: Broadcast scheduling: algorithms and complexity. In: SODA, pp. 473–482 (2008)
10. Chekuri, C., Moseley, B.: Online scheduling to minimize the maximum delay factor. In: SODA, pp. 1116–1125 (2009)
11. Edmonds, J.: Scheduling in the dark. Theor. Comput. Sci. 235(1), 109–141 (2000)
12. Edmonds, J., Pruhs, K.: Multicast pull scheduling: When fairness is fine. Algorithmica 36(3), 315–330 (2003)
13. Edmonds, J., Pruhs, K.: A maiden analysis of longest wait first. ACM Trans. Algorithms 1(1), 14–32 (2005)

14. Edmonds, J., Pruhs, K.: Scalably scheduling processes with arbitrary speedup curves. In: SODA, pp. 685–692 (2009)
15. Erlebach, T., Hall, A.: Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In: SODA, pp. 194–202 (2002)
16. Gandhi, R., Khuller, S., Kim, Y.-A., Wan, Y.-C.J.: Algorithms for minimizing response time in broadcast scheduling. Algorithmica 38(4), 597–608 (2004)
17. Gandhi, R., Khuller, S., Parthasarathy, S., Srinivasan, A.: Dependent rounding and its applications to approximation algorithms. J. ACM 53(3), 324–360 (2006)
18. Hall, A., Täubig, H.: Comparing push- and pull-based broadcasting. or: Would "microsoft watches" profit from a transmitter? In: Jansen, K., Margraf, M., Mastrolli, M., Rolim, J.D.P. (eds.) WEA 2003. LNCS, vol. 2647, pp. 148–164. Springer, Heidelberg (2003)
19. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM 47(4), 617–643 (2000)
20. Kalyanasundaram, B., Pruhs, K., Velauthapillai, M.: Scheduling broadcasts in wireless networks. J. Scheduling 4(6), 339–354 (2000)
21. Pruhs, K.: Competitive online scheduling for server systems. SIGMETRICS Perform. Eval. Rev. 34(4), 52–58 (2007)
22. Wong, J.: Broadcast delivery. Proc. IEEE 76(12), 1566–1577 (1988)

The Routing Open Shop Problem: New Approximation Algorithms[*]

Ilya Chernykh[1,2], Nikita Dryuck[2], Alexander Kononov[1,2], and Sergey Sevastyanov[1,2]

[1] Sobolev Institute of Mathematics
Acad. Koptyug pr. 4, 630090 Novosibirsk, Russia
[2] Novosibirsk State University
Acad. Koptyug pr. 2, 630090 Novosibirsk, Russia

Abstract. We consider the routing open shop problem being a generalization of the open shop and the metric travelling salesman problems. The jobs are located at nodes of some transportation network, and the machines travel on the network to execute the jobs in the open shop environment. The machines are initially located at the same node (depot) and must return to the depot after completing all the jobs. It is required to find a non-preemptive schedule that minimizes the makespan. The problem is NP-hard even on a two-node network with two machines. We present new polynomial-time approximation algorithms with worst-case performance guarantees.

Keywords: Routing open shop, approximation algorithm, worst-case analysis.

1 Introduction

We consider the routing open shop problem being a generalization of the open shop and the metric travelling salesman problems. Both problems are strongly NP-hard (see [16] and [9], respectively).

Open shop problem ([11])
We have a set of n jobs $\boldsymbol{J} = \{J_1, \dots, J_n\}$ and a set of m machines $\boldsymbol{M} = \{M_1, \dots, M_m\}$. Each job J_j has to be processed by each machine M_i, and this operation takes $p_{ji} \in Z^+$ time units. Operations of each job can be processed in an arbitrary order. Preemption is not allowed. Different machines cannot work on the same job simultaneously, and a machine cannot work on more than one job at a time. The goal is to minimize the makespan. (For this problem it coincides with the maximum job completion time $C_{\max}$.)

Metric traveling salesman problem (metric TSP)
We have an undirected edge-weighted complete graph $G = \langle V, E \rangle$, the weight τ_{ij}

[*] This research is supported by RFBR grant N 08-01-00370. Research of the third author is partially supported by ADTP grant 2.1.1/3235.

of edge $e_{ij} = [v_i, v_j]$ is a nonnegative integer which represents a distance between nodes v_i and v_j. Distances satisfy the triangle inequality. The goal is to build a hamiltonian tour R in G with the minimum total weight $|R| \doteq \sum_{e_{ij} \in R} \tau_{ij}$.

Routing open shop problem
The input of this problem combines the inputs of two problems mentioned above. The jobs are located at nodes of an undirected transportation network G. The machines have to travel between the jobs (with unit speed). Thus not only the processing times of the operations, but also the travel times between jobs have to be taken into account.

It is assumed that all machines are initially located at the same node (depot), and have to return to the depot after the completion of all jobs. Any number of machines can travel through the same edge or node simultaneously in any direction. All machines use the shortest path between the nodes. Without loss of generality we associate node v_j with job J_j for $j = 1, \ldots, n$ and a special node v_0 with the depot. Thus, we have a complete graph $G = \langle V, E \rangle$, with the set of nodes $V = \{v_0, v_1, \ldots, v_n\}$ and the set of edges E, where all distances satisfy the triangle inequality.

The makespan of a feasible schedule is the interval between the instant when the machines start working or moving and the instant when the last machine returns to the depot after finishing all its operations. The goal is to minimize the makespan ($F_{\max}$).

According to the standard three-fold notation of scheduling problems [13], we will denote this problem as $\langle RO||F_{\max}\rangle$ (or $\langle ROm||F_{\max}\rangle$ for a fixed number m of machines).

In the last decade considerable amount of research has been devoted to routing scheduling problems. Most of research in this area focuses on single-stage scheduling environments (e.g., see [7,8,12] and the references therein). Multistage scheduling problems with machines traveling between jobs located at nodes of a transportation network have been studied in [1,2,3,4]. Examples of applications where machines have to travel between jobs either include the situation where the details are too big or too heavy to be moved between machines (e.g., casings of ships), or scheduling of robots that perform daily maintenance operations on immovable objects located at different places of a workshop [2].

Previous approximation results for $RO||F_{\max}$
The routing open shop problem was introduced by I. Averbakh, O. Berman, and I. Chernykh in [3], [4]. It is strongly NP-hard even for a single machine case as it contains the metric TSP as special case. Moreover, the routing open shop problem is NP-hard even on a 2-node network with only two machines [4].

For the latter case a 6/5-approximation polynomial time algorithm was presented in [3]. A 7/4-approximation algorithm for the general 2-machine case and a simple $(m + 4)/2$-approximation algorithm for the m-machine case were given in [4].

A similar problem is considered in [15]. In that problem, n jobs have to be scheduled in a two-machine open shop to minimize the makespan. It is assumed that there is a known (job-specific) time lag between the completion of an operation and the beginning of the next operation of the same job. These time lags are also interpreted as transportation times (of jobs between the immovable machines). At that, any job is available for any machine at the beginning of the schedule, so the time lags can be interpreted as cooling or heating times in this model. For this problem, a 1.5-approximate heuristic is presented in [15].

New results
Our main results are a ρ-approximation algorithm with $\rho = O(\sqrt{m})$ for $\langle RO||F_{\max}\rangle$ with m machines and a $\frac{13}{8}$-approximation algorithm for $\langle RO2||F_{\max}\rangle$. We also present a $\frac{3}{2}$-approximation algorithm for the 2-machines case with a transportation network that enables one to solve TSP in polynomial time ("easy TSP").

The remainder of the paper is organized as follows. Section 2 provides necessary definitions and preliminary results. Improved approximation algorithms for the m-machine and 2-machine cases are given in Sections 3 and 4 respectively. The 2-machine case with "easy TSP" is considered in Section 5.

2 Preliminary Results, Definitions and Notation

We define the *length* d_j of job $J_j \in \boldsymbol{J}$ as the total processing time of its operations, $d_j = \sum_{i=1}^{m} p_{ji}$, and denote $d_{\max} = \max_{J_j \in \boldsymbol{J}} d_j$, $\delta_{\max} = \max_{J_j \in \boldsymbol{J}}(d_j + 2\tau_{0j})$. The total processing time of the operations on machine M_i is denoted by ℓ_i and is called the *load* of machine M_i; $\ell_{\max} = \max_i \ell_i$ is the *maximum machine load* and $F_{\max}(\sigma)$ is the *makespan of schedule* σ; $F^*_{\max}$ stands for the optimum; T^* stands for the length of the optimal tour in G.

Since each machine has to execute operations of all jobs, it has to visit all nodes of G, and we get the following lower bound

$$\bar{F} = \max\{\ell_{\max} + T^*, \delta_{\max}\} \leqslant F^*_{\max} \ . \tag{1}$$

As was mentioned above, the metric TSP is strongly NP-hard. The best known approximation algorithm for this problem is due to Christofides [6] and Serdyukov [14].

Algorithm $\mathcal{A}_{CS}$ (Christofides-Serdyukov)
Input: A complete edge-weighted graph G.
1. Find a minimal spanning tree H in G.
2. Let X be the set of nodes having odd degrees in H, and let G_X be the induced graph of G. Find a minimum weight matching M in G_X.
3. Find an eulerian walk in $H \bigcup M$. The order in which nodes appear in this walk defines a near-optimal tour in G.
Output: Hamiltonian tour R_{CS} in G.

The total weight of edges in H does not exceed T^*, and the weight of M does not exceed $\frac{T^*}{2}$. The triangle inequality immediately implies that

$$|R_{CS}| \leqslant \frac{3}{2}T^* \ . \tag{2}$$

Note that algorithm $\mathcal{A}_{CS}$ runs in $O(n^3)$ time (the running time of Step 2).

As for the open shop problem, it is well known that the greedy algorithm provides 2-approximation [5]. More precisely, the length of any schedule obtained by the greedy algorithm does not exceed $\ell_{\max} + d_{\max}$.

We present a similar result for $\langle RO|\tau_{j'j''} = \tau|F_{\max}\rangle$ with equal distances between all nodes. To do that, we consider a more general problem.

Open shop with post-setup times
This problem differs from the ordinary open shop problem by the following additional requirement: each machine M_i after the completion of the operation of job J_j needs a post-setup time pst_{ij} before it can start processing the next job. The goal is to minimize the maximum completion time of the post-setup procedure over all operations, which is denoted as $F_{\max}$.

This problem will be referred to as $\langle O|pst_{ij}|F_{\max}\rangle$.

Let $\varGamma_i = \sum_j pst_{ij}$ stand for the *total post-setup time* of machine M_i, and $\gamma_{\max} = \max_{M_i \in \boldsymbol{M}}(\ell_i + \varGamma_i)$. Since each machine M_i needs at least $\ell_i + \varGamma_i$ time to process all its operations (including the post-setup times), the following lower bound on the optimum holds:

$$F^*_{\max} \geqslant \max\{\gamma_{\max}, d_{\max}\} \tag{3}$$

Note that in any schedule any machine at any time may have one of the following three statuses: *working* (processing some job), *resting* (undergoing the post-setup procedure) or *waiting* (none of the above). Any job can be either *busy* (while being processed on a machine) or *available*. We call a job to be *in queue for a machine*, if the job has not yet been processed by the machine.

Consider the following Greedy Algorithm for $\langle O|pst_{ij}|F_{\max}\rangle$. At any iteration of the algorithm a partial schedule is defined in the interval $[0, t]$, where t increases at discrete steps.

Greedy Algorithm

Initialization. All machines are initially *waiting*, and all jobs are *available* and are *in queue* for every machine (the queue is pre-specified for each machine as a sequence of jobs). Set $t_i := \infty$ for $i = 1, \ldots, m$. (The infinite value of t_i means that machine M_i is *waiting*; a finite value of t_i will specify the nearest time moment when machine M_i changes its status.)

For every machine M_i, $i = 1, \ldots, m$, take the first available job J_j in queue for this machine (if any) and schedule its operation at time 0 (machine M_i becomes *working*, job J_j becomes *busy*), and set $t_i := p_{ji}$.

Iteration. Set $t := \min_i t_i$, $i^* := \min\{i|t_i = t\}$. If $t = \infty$ then **STOP** (this means that all machines are waiting and there is no job in queue for any machine).

If $t < \infty$ and machine M_{i^*} is working, this means that at time t machine M_{i^*} is completing the operation of some job J_j; change the status of job J_j to *available*, the status of machine M_{i^*} to *resting*, $t_{i^*} := t + pst_{i^*j}$; remove J_j from the queue for M_{i^*}. If there is a waiting machine for which job J_j is in queue, schedule its operation at time t. **GOTO** next iteration.

In the case that M_{i^*} is *resting* (i.e., t is a completion time of the post-setup procedure), if there is an available job J_j in queue for M_{i^*} then schedule its operation at time t, and set $t_{i^*} := t + p_{ji^*}$, else set the status of machine M_{i^*} to *waiting*. **GOTO** next iteration.

Note that the Greedy Algorithm can be easily implemented to run in $O(mn \cdot \min\{m, n\})$ time.

Lemma 1. *The Greedy Algorithm for $\langle O|pst_{ij}|F_{\max}\rangle$ runs in $O(nm \cdot \min\{m, n\})$ time and constructs a schedule σ with makespan at most $d_{\max} + \gamma_{\max}$.*

Proof. Indeed, let schedule σ terminate with the post-setup procedure performed with machine M_i after the processing of job J_j. Then $F_{\max}(\sigma) = \gamma_i + W_i$, where $\gamma_i = \ell_i + \Gamma_i$ and W_i is the total waiting time of machine M_i in interval $[0, F_{\max}(\sigma)]$. Note that at any time when machine M_i is waiting, job J_j is in queue for M_i but not available (because of being processed by some other machine). Therefore, $W_i \leqslant d_j$, which implies the claim of the Lemma. □

Bound (3) implies the following

Corollary 1. *The Greedy Algorithm is a 2-approximation algorithm for $\langle O|pst_{ij}|F_{\max}\rangle$, as well as for its special case $\langle RO|\tau_{j'j''} = \tau|F_{\max}\rangle$.* □

3 The m-Machine Routing Open Shop Problem

In this section we present a polynomial time algorithm that for any instance of the general routing open shop problem delivers a schedule with makespan $F_{\max}(\sigma) \leqslant O(\sqrt{m})\bar{F}$. Let us briefly sketch the ideas of the algorithm.

The first step consists in finding a near-optimal tour R in graph G.

At step 2 we replace the given set of jobs by at most $O(\sqrt{m})$ new "aggregated" jobs. Each aggregated job combines several original jobs consecutively located on a segment of tour R. The processing time of a new operation of an aggregated job on a machine is equal to the total processing time of its components on that machine, plus the length of the path connecting those jobs in R. We also set the post-setup time of each machine to be equal to the maximum distance between the nodes in graph G.

Next we apply the Greedy Algorithm to the defined above instance of problem $\langle O|pst_{ij}|F_{\max}\rangle$. The obtained schedule can be converted to a feasible schedule for the original instance of the routing open shop problem. The makespan of the schedule does not exceed $c\sqrt{m}F^*_{\max}$, where c is a constant.

3.1 Approximation Algorithm for $RO||F_{\max}$ with m Machines

Each step of the algorithm described below is followed by a brief discussion and implementation details, if needed. The algorithm uses two positive parameters α, β (whose exact values will be defined later) and consists of five steps.

Algorithm ROS(α, β)

Step I. Use algorithm $\mathcal{A}_{CS}$ to find a near-optimal hamiltonian tour R_{CS} with length $T \leqslant \frac{3}{2}T^*$ in graph G. Without loss of generality we assume that R_{CS} walks through the nodes in the order $v_0, v_1, \ldots, v_n$. (The running time is $O(n^3)$.)

Step II. Partition the tour R_{CS} into disjoint paths $P_1, \ldots, P_k$, where the number of pathes k is specified at the completion of step II.

```
Put $i := 0$; $j := 0$.
**While** $j \leqslant n$ **do begin**
$i := i + 1$; $P_i :=$**null**;
**repeat** $\{P_i := P_i \oplus v_j;\ j := j + 1\}$ **until**
at least one of three conditions is satisfied:
(a) $\sum_{v_j \in P_i} d_j \geqslant \alpha\sqrt{m}\,\ell_{\max}$ (where $d_0 = 0$, for certainty),
(b) $\sum_{v_j \in P_i} \tau_{j,j+1} \geqslant \frac{\beta T}{\sqrt{m}}$ (where $\tau_{n,n+1} = \tau_{n0}$, for certainty),
(c) $j > n$.
**end** (while)
```

(The running time of step II does not exceed $O(nm)$.)

Step III. Set $\tau \doteq \max_{j',j''} \tau_{j'j''}$. We define an instance I of $\langle O|pst_{ij} = \tau|F_{\max}\rangle$ with $k+1$ jobs that have to be executed on the set of m machines. To that end, for each path P_j, $j = 1, \ldots, k$, we define job J'_j. The processing time of job J'_j on machine M_i is set to $p'_{ji} \doteq \sum_{v_h \in P_j} p_{hi} + \sum_{e_{hh'} \in P_j} \tau_{hh'}$. We also define a dummy job J'_0 with zero processing times of all operations. Let d'_j stand for the length of job J'_j. Each machine M_i requires the same post-setup time $pst_{ij} = \tau$ after the processing of each job J'_j. (The running time of the step is $O(nm)$.)

Step IV. Run the Greedy Algorithm on instance I to obtain a schedule σ in which all machines but one start with the dummy job. Machine M_1 starts with job J_1. (The running time is $O(km \cdot \min\{k, m\})$, according to Lemma 1.)

Step V. Convert schedule σ to a feasible solution of the original problem, treating each post-setup time of machine M_i as its travel time from one aggregated job (or base) to another (or base). At that, the post-setup time on machine M_i after processing the dummy job enables the machine to start with an aggregated job different from J'_1. (The running time is $O(nm)$.)

3.2 Performance Ratio of Algorithm ROS

Note that for each path P_i, $i \neq k$, either $\sum_{v_j \in P_i} d_j \geqslant \alpha\sqrt{m}\,\ell_{\max}$ or $\sum_{v_j \in P_i} \tau_{j,j+1} \geqslant \frac{\beta T}{\sqrt{m}}$ holds. Since the total load of all machines does not exceed $m\ell_{\max}$, there are at most $\lfloor\frac{\sqrt{m}}{\alpha}\rfloor$ paths with $\sum_{v_j \in P_i} d_j \geqslant \alpha\sqrt{m}\,\ell_{\max}$. Observe that $\sum_{i=1}^{k}\sum_{v_j \in P_i} \tau_{j,j+1} = T$. It follows that there are at most $\lfloor\frac{\sqrt{m}}{\beta}\rfloor$ paths with $\sum_{v_j \in P_i} \tau_{j,j+1} \geqslant \frac{\beta T}{\sqrt{m}}$. Thus, $k \leqslant \lfloor\frac{\sqrt{m}}{\alpha}\rfloor + \lfloor\frac{\sqrt{m}}{\beta}\rfloor + 1$.

Let $d'_{\max}$ be the maximum job length over all jobs and $\ell'_{\max}$ be the maximum machine load in instance I obtained at step III of Algorithm ROS. Conditions

a) and b) at step II ensure that $d'_j = \sum_{i=1}^m p'_{ji} = \sum_{v_h \in P_j} d_h + m \sum_{e_{hh'} \in P_j} \tau_{hh'} \leqslant \alpha\sqrt{m}\,\ell_{\max} + d_{\max} + m\frac{\beta T}{\sqrt{m}}$, therefore

$$d'_{\max} \leqslant \alpha\sqrt{m}\,\ell_{\max} + d_{\max} + \beta\sqrt{m}\,T\ .$$

By definition, the load of machine M_i meets

$$\ell'_i = \sum_{j=1}^k p'_{ij} = \sum_{j=1}^k \left(\sum_{v_h \in P_j} p_{ih} + \sum_{e_{hh'} \in P_j} \tau_{hh'} \right) \leqslant \ell_i + T\ .$$

It follows that $\ell'_{\max} \leqslant \ell_{\max} + T$ and $\gamma_{\max} = \ell'_{\max} + (k+1)\tau$. Lemma 1 implies

$$F_{\max}(\sigma) \leqslant d'_{\max} + \ell'_{\max} + (k+1)\tau \leqslant \alpha\sqrt{m}\,\ell_{\max} + d_{\max} + \beta\sqrt{m}\,T + \ell_{\max} + T + \left(\left\lfloor \frac{\sqrt{m}}{\alpha} \right\rfloor + \left\lfloor \frac{\sqrt{m}}{\beta} \right\rfloor + 2 \right)\tau\ .$$

Since $T \leqslant \frac{3}{2}T^*$ and $\tau \leqslant \frac{1}{2}T^*$,

$$F_{\max}(\sigma) \leqslant \alpha\sqrt{m}\,\ell_{\max} + \left(\frac{3\beta}{2} + \frac{1}{2\alpha} + \frac{1}{2\beta} \right)\sqrt{m}\,T^* + d_{\max} + \ell_{\max} + \frac{5}{2}T^*\ .$$

By setting $\alpha = \frac{\sqrt{3}+\sqrt{5}}{2}$ and $\beta = \frac{\sqrt{3}}{3}$, we obtain

$$F_{\max}(\sigma) \leqslant \left(\frac{\sqrt{3}+\sqrt{5}}{2}\sqrt{m} + 3.5 \right)\bar{F}\ .$$

Note that the running time of step IV does not exceed $O(m^2)$ since $k \leqslant O(\sqrt{m})$. This implies the following

Theorem 1. *Algorithm ROS runs in $O(n^3+m^2)$ time and provides an $O(\sqrt{m})$-approximation solution for $\langle RO||F_{\max}\rangle$ with m machines.*

4 Two-Machine Routing Open Shop Problem

In this section we present a new approximation algorithm for the routing open shop problem with two machines. We remind the reader that this problem is strongly NP-hard as it contains the metric TSP. A 7/4-approximation algorithm $\mathcal{A}_{74}$ for the 2-machine case was given in [4]. It can be briefly described as follows.

Suppose, we have some near-optimal hamiltonian tour R. Let machine M_1 perform its operations in the order defined by R, while M_2 executes the jobs in the reverse order. Both machines travel between the nodes without unnecessary delays. If machines conflict at some job J_j (which is called a *conflict job*), we consider two schedules: schedule σ'_R, where the conflict job is first performed by M_1 and then by M_2, and schedule σ''_R with the reverse order of processing these two operations. The shortest of two schedules is denoted by σ_R. Note that in both schedules σ'_R, σ''_R at most one of the machines has an idle time.

It is shown in [4] that

$$F_{\max}(\sigma_R) \leqslant |R| + \ell_{\max} + \frac{1}{2}d_j \ . \tag{4}$$

Thus, for $R = R_{CS}$ we obtain from (2) and (4) that

$$F_{\max}(\sigma_R) \leqslant \frac{3}{2}T^* + \ell_{\max} + \frac{1}{2}d_j \ . \tag{5}$$

If $\ell_{\max} \geqslant d_j$ then (5) and (1) directly imply $F_{\max}(\sigma_R) \leqslant \frac{3}{2}F^*_{\max}$.

For the opposite case it can be shown that $F_{\max}(\sigma_R) \leqslant \frac{7}{4}F^*_{\max}$.

4.1 Approximation Algorithm for the Two-Machine Problem

In the approximation algorithm for the two-machine routing open shop problem we use the following modification of the Christofides-Serdyukov algorithm. Let J_j be a job with the maximum length, i.e., $d_{\max} = d_j$. We want to get a near-optimal tour which contains the edge from the depot to node v_j.

Algorithm $\mathcal{A}_{MCS}$

1. Find a minimal spanning tree H in G.
2. If $e_{0j} \notin H$, add e_{0j} to H and remove any other edge from the obtained cycle.
3. Let X be the set of nodes having odd degrees in H, and let G_X be the induced graph of G. Find a minimum weight matching M in G_X.
4. Find an eulerian walk in $H \bigcup M$. Without loss of generality we can assume that e_{0j} is the first edge in this walk. The order in which nodes appear in this walk defines a near-optimal tour R'.

By the triangle inequality the length of the tour R' does not exceed the total weight of edges in H plus the sum of weights of M and the length of e_{0j}. This immediately implies that the obtained tour is no longer than $\frac{3}{2}T^* + \tau_{0j}$.

Algorithm ROS-2

Step I. Find a near-optimal hamiltonian tour R_1 in G by algorithm $\mathcal{A}_{CS}$.
Step II. Find the schedule σ_{R_1} by algorithm $\mathcal{A}_{74}$.
Step III. Let J_j be a job with the maximum length. Find the near optimal hamiltonian tour R_2 containing the edge e_{0j} by means of algorithm $\mathcal{A}_{MCS}$.
Step IV. Find the schedule σ_{R_2} by algorithm $\mathcal{A}_{74}$.
Step V. Take the best of schedules σ_{R_1} and σ_{R_2}.

4.2 Performance Ratio of Algorithm ROS-2

Let $F_{\max} := \min\{F_{\max}(\sigma_{R_1}), F_{\max}(\sigma_{R_2})\}$ be the makespan of the schedule obtained by Algorithm ROS-2. We have by (5) that

$$F_{\max}(\sigma_{R_1}) \leqslant \frac{3}{2}T^* + \ell_{\max} + \frac{1}{2}d_j \tag{6}$$

If $\ell_{\max} \geqslant d_{\max}$, (1) and (6) imply

$$F_{\max} \leqslant F_{\max}(\sigma_{R_1}) \leqslant \frac{3}{2} F^*_{\max} \; .$$

Let $\ell_{\max} < d_{\max}$. We consider the following three cases.

Case 1. If there is no conflict job in σ_{R_2}, we have

$$F_{\max}(\sigma_{R_2}) \leqslant \frac{3}{2}T^* + \tau_{0j} + \ell_{\max} \tag{7}$$

Case 2. Let J_j be the conflict job in σ_{R_2}. In this case algorithm $\mathcal{A}_{74}$ chooses the best of two schedules: schedule $\sigma_{R_2''}$ (in which the second machine has an idle time, waiting until machine M_1 completes its operation of job J_j), and schedule $\sigma_{R_2'}$ in which machine M_1 has an idle time. In the first schedule machine M_2 finishes at time $d_j + 2\tau_{0j}$, while machine M_1 works without any idle time and finishes at time $|R_1| + \tau_{0j} + \ell_1 \leqslant \frac{3}{2}T^* + \tau_{0j} + \ell_{\max}$. Therefore, we get the bound

$$F_{\max}(\sigma_{R_2}) \leqslant \max\left\{ d_j + 2\tau_{0j}, \frac{3}{2}T^* + \tau_{0j} + \ell_{\max} \right\}, \tag{8}$$

regardless of which of two schedules has been chosen by algorithm $\mathcal{A}_{74}$.

Case 3. Let J_i, $i \neq j$ be the conflict job in σ_{R_2}. It follows from (4) that

$$F_{\max}(\sigma_{R_2}) \leqslant \frac{3}{2}T^* + \tau_{0j} + \ell_{\max} + \frac{d_i}{2} \tag{9}$$

Regardless of which of three cases takes place, we obtain the bound

$$F_{\max}(\sigma_{R_2}) \leqslant \max\left\{ d_j + 2\tau_{0j}, \frac{3}{2}T^* + \tau_{0j} + \ell_{\max} + \frac{d_i}{2} \right\}$$

Since $d_j + 2\tau_{0j} \leqslant \bar{F}$, either σ_{R_2} is the optimal solution, or

$$F_{\max}(\sigma_{R_2}) \leqslant \frac{3}{2}T^* + \tau_{0j} + \ell_{\max} + \frac{d_i}{2} \tag{10}$$

Suppose, schedule σ_{R_2} is not optimal. Then, using (6), (10), $d_i + d_j \leqslant 2\ell_{\max}$, and (1), we obtain

$$\begin{gathered}
4F_{\max} \leqslant 3F_{\max}(\sigma_{R_1}) + F_{\max}(\sigma_{R_2}) \leqslant 3\left(\frac{3}{2}T^* + \ell_{\max} + \frac{1}{2}d_j\right) \\
+\frac{3}{2}T^* + \tau_{0j} + \ell_{\max} + \frac{d_i}{2} = 6T^* + 4\ell_{\max} + \tau_{0j} + \frac{3}{2}d_j + \frac{d_i}{2} \\
= 4(T^* + \ell_{\max}) + \frac{2\tau_{0j} + d_j}{2} + \left(T^* + \frac{d_i + d_j}{2}\right) + \left(T^* + \frac{d_j}{2}\right) \leqslant \frac{13}{2}F^*_{\max} \; , \\
F_{\max} = \min\{F_{\max}(\sigma_{R_1}), F_{\max}(\sigma_{R_2})\} \leqslant \frac{13}{8}F^*_{\max}
\end{gathered}$$

Note that the algorithms $\mathcal{A}_{CS}$ and $\mathcal{A}_{MCS}$ have running times $O(n^3)$, while the schedules σ_{R_1} and σ_{R_2} can be built in linear time. Thus, we obtain

Theorem 2. *Algorithm ROS-2 provides a* 1.625*-approximation solution for the 2-machine routing open shop problem in* $O(n^3)$ *time.*

5 "Easy TSP" and Approximation Algorithms for the Two-Machine Routing Open Shop Problem

Though the general TSP is strongly NP-hard, many special cases can be efficiently solved. There are two broad classes of well-solvable TSP. In one class the problems are polynomially solvable because of restrictions on the matrix of distances. (For example, the matrix may be circulant or satisfy the Demidenko conditions.) Another class contains problems in which the transportation network has got a particular structure. (For example, the network may be a tree or may have a limited bandwidth. For more results see [10].)

In this section we suppose that we can find an optimal hamiltonian tour in G in polynomial time. (We remind the reader that the routing open shop problem with two machines remains NP-hard even on a two-node network.) The problem with this assumption will be referred to as $\langle RO2 \,|\, easy{-}TSP \,|\, F_{\max}\rangle$. Under this assumption, (4) implies

$$F_{\max}(\sigma_R) \leqslant \frac{3}{2} F^*_{\max}$$

A tighter ratio performance guarantee of 11/8 can be proved for a more complicated approximation algorithm that will be presented in the full version of our paper.

References

1. Averbakh, I., Berman, O.: Routing Two-Machine Flowshop Problems on Networks with Special Structure. Transportation Science 30(4), 303–314 (1996)
2. Averbakh, I., Berman, O.: A Simple Heuristic for m-machine Flow-Shop and its Applications in Routing-Scheduling Problems. Operations Research 47(1), 165–170 (1999)
3. Averbakh, I., Berman, O., Chernykh, I.: A $\frac{6}{5}$-approximation algorithm for the two-machine routing open shop problem on a 2-node network. European Journal of Operational Research 166(1), 3–24 (2005)
4. Averbakh, I., Berman, O., Chernykh, I.: The Routing Open-Shop Problem on a Network: Complexity and Approximation. European Journal of Operational Research 173(2), 521–539 (2006)
5. Bárány, I., Fiala, T.: Többgépes ütemezési problémàk közel optimális megoldása. Szigma-Mat.-Közgazdasági Folyóirat 15, 177–191 (1982)
6. Christofides, N.: Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem, Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA (1976)
7. Desrosiers, J., Dumas, Y., Solomon, M., Soumis, F.: Time Constrained Routing and Scheduling. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (eds.) Handbooks in Operations Research and Management Science, Network Routing, North-Holland, vol. 8, pp. 35–139 (1995)

8. Fischetti, M., Laporte, G., Martello, S.: The Delivery Man Problem and Cumulative Matroids. Operations Research 41(6), 1055–1064 (1993)
9. Garey, M., Johnson, D.: Computers and Intractability, A Guide to the theory of NP-completeness. W.H. Freemann and Company, San Francisco (1979)
10. Gilmore, P.C., Lawler, E.L., Shmoys, D.B.: Well-solved s pecial cases. In: Lawler, E.L., Lenstra, J.K., Rinnoy Kan, A.H.G., Shmoys, D.B. (eds.) The Travelling Salesman Problem. John Wiley & Sons Ltd, Chichester (1985)
11. Gonzalez, T., Sahni, S.: Open Shop Scheduling to Minimize Finish Time. J. Assoc. Comp. Math. 23, 665–679 (1976)
12. Karuno, Y., Nagamochi, H., Ibaraki, T.: Vehicle Scheduling on a Tree with Release and Handling Times. Annals of Operations Research 69, 193–207 (1997)
13. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and Scheduling: Algorithms and Complexity. In: Handbooks in Operations Research and Management Science. Logistics of Production and Inventory, vol. 4, pp. 445–522. North Holland, Amsterdam (1993)
14. Serdyukov, A.: On some extremal routes in graphs. Upravlyaemye Sistemy 17, 76–79 (1978) (in Russian)
15. Strusevich, V.A.: A Heuristic for the Two-Machine Open Shop Scheduling Problem with Transportation Times. Discrete Applied Mathematics 93, 287–304 (1999)
16. Williamson, D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevastianov, S.V., Shmoys, D.B.: Short shop schedules. Oper.Res. 45(2), 288–294 (1997)

On the Price of Stability for Undirected Network Design

George Christodoulou[1], Christine Chung[2,⋆], Katrina Ligett[3], Evangelia Pyrga[1], and Rob van Stee[1,⋆⋆]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
{gchristo,pyrga,vanstee}@mpi-inf.mpg.de
[2] Connecticut College, New London, CT, USA
cchung@conncoll.edu
[3] Cornell University, Ithaca, NY, USA
katrina@cs.cornell.edu

Abstract. We continue the study of the effects of selfish behavior in the network design problem. We provide new bounds for the price of stability for network design with fair cost allocation for undirected graphs. We consider the most general case, for which the best known upper bound is the Harmonic number H_n, where n is the number of agents, and the best previously known lower bound is $12/7 \approx 1.778$.

We present a nontrivial lower bound of $42/23 \approx 1.8261$. Furthermore, we show that for two players, the price of stability is exactly $4/3$, while for three players it is at least $74/48 \approx 1.542$ and at most 1.65. These are the first improvements on the bound of H_n for general networks. In particular, this demonstrates a separation between the price of stability on undirected graphs and that on directed graphs, where H_n is tight. Previously, such a gap was only known for the cases where all players have a shared source, and for weighted players.

1 Introduction

The effects of selfish behavior in networks is a natural problem with long-standing and wide-spread practical relevance. As such, a wide variety of network design and connection games have received attention in the algorithmic game theory literature (for a survey, see [1]).

One natural question is how much the users' selfish behavior affects the performance of the system. Koutsoupias and Papadimitriou [2,3] addressed this question using a worst-case measure, namely the *Price of Anarchy* (PoA). This notion compares the cost of the worst-case Nash equilibrium to that of the social optimum (the best that could be obtained by central coordination). From an optimistic point of view, Anshelevich et al. [4] proposed the *Price of Stability* (PoS), the ratio of the lowest Nash equilibrium cost to the social cost, as a measure of the minimal effect of selfishness.

There has been substantial work on the PoA for *congestion games*, a broad class of games with interesting properties originally introduced by Rosenthal [5]. Congestion games nicely model situations that arise in selfish routing, resource allocation and

⋆ Some of this research was conducted while this author was visiting the Max-Planck-Institut für Informatik, Saarbrücken, Germany.

⋆⋆ Research supported by German Research Foundation (DFG).

network design problems, and the PoA for these games is now quite well-understood [6,7,8,9]. By comparison, much less work has been done on the PoS: The PoS for network design games has been studied by [4,10,11,12,13], while the PoS for routing games[1] was studied by [4,8,14]. However, PoA techniques cannot easily be transferred to the study of PoS. New techniques thus need to be developed; this work moves toward this direction.

The particular network design problem we address here is the one which was initially studied by Anshelevich et al. [4], sometimes referred to as the fair cost sharing network design (or creation) game. In it, each player has a set of endpoints in a network that he must connect; to achieve this, he must choose a subset of the links in the network to utilize. Each link has a cost associated with it, and if more than one player wishes to utilize the same link, the cost of that link is split evenly among the players. Each player's goal is to pay as little as possible to connect his endpoints. The global social objective is to connect all player's endpoints as cheaply as possible.

Anshelevich et al. [4] showed that if G is a directed graph, the price of anarchy is equal to n, the number of players, whereas the price of stability is exactly the nth harmonic number H_n. The upper bound is proven by using the fact that our network design game, and in fact any congestion game, is a potential game. A *potential game*, first defined by Monderer and Shapley [15], is a game where the change to a player's payoff due to a deviation from a game solution can be reflected in a *potential function*, or a function that maps game states to real numbers.

This upper bound of H_n holds even in the case of undirected graphs (since the potential function of the game does not change when the underlying graph is undirected), however the lower bound does not. Hence the central open question we study is:

> *What is the price of stability in the fair cost sharing network design game on undirected graphs?*

In the case of two players and a single common sink vertex, Anshelevich et al. [4] show that the answer is $4/3$. Some further progress has also more recently been made toward answering this question. Fiat et al. [12] showed that in the case where there is a single common sink vertex and every other vertex is a source vertex, the price of stability is $O(\log\log n)$. They also give an n-player lower bound instance of $12/7$ [16]. For the more general case where the agents share a sink but not every vertex is a source vertex, Li [13] showed an upper bound of $O(\log n/\log\log n)$. Chen and Roughgarden [10] studied the price of stability for the *weighted* variant of the game, where each player pays a fraction of each edge cost proportional to her weight. Albers [11] showed that in this variant, the price of stability is $\Omega(\log W/\log\log W)$, where W is the sum of the players' weights.

Our contributions. We show for the first time that the price of stability in undirected networks is definitively different from the one for directed networks in the general case

[1] Both cost-sharing network design games and network routing games fall in the class of congestion games and they differ only in the edge cost functions. Cost sharing network design games come together with decreasing cost functions on the edges, e.g. $c_e(x) = c_e/x$, while routing games come with increasing latency functions, e.g. $c_e(x) = c_e \cdot x$.

(where all players may have distinct source and destination vertices). In particular, we show that PoS is exactly $4/3$ for two agents (strictly less than PoS in the directed case, which is $H_2 = 3/2$), while for three agents it is at least $74/48 \approx 1.542$ and at most 1.65 (again strictly less than PoS in the directed case, which is $H_3 = 11/6$). Furthermore, we show that the price of stability for general n is at least $42/23 > 1.8261$, improving upon the previous bound due to Fiat et al. [12].

1.1 The Model

We are given an underlying network, $G = (V, E)$, where V is the set of vertices and E is the set of edges in the network. Each player $i = 1 \dots n$ has a set of two nodes (endpoints) $s_i, t_i \in V$ to connect. We refer to s_i as the *source* endpoint of player i and t_i as the *destination* or *sink* endpoint of player i. The strategy set of each player i consists of all sets of edges $S_i \subseteq E$ such that S_i connects all the vertices in T_i. There is a cost c_e associated with each edge $e \in E$. The cost to player i of a solution $S = (S_1, S_2, \dots, S_n)$ is $C_i(S) = \sum_{e \in S_i} c_e/n_e$ where n_e is the number of players in S who chose a strategy that contains e. Each player i wants to minimize $C_i(S)$. The global objective is minimize $\sum_{i=1}^{n} C_i(S)$.

2 A Lower Bound of 1.826

Consider a 3 by N grid for some large N. There are three nodes and two horizontal edges in every row. The levels are numbered starting from the bottom. We denote the horizontal edges on level i by L_i and R_i (from left to right). The nodes on level i are denoted by v_{ij} $(j = 1, 2, 3)$ and the vertical edges connecting level i to level $i + 1$ are denoted by e_{ij} $(j = 1, 2, 3)$. Each node v_{ij} for $i = 1, \dots, N - 1$ and $j = 1, 2, 3$ is the source of some agent $p_{i,j}$, who has node $v_{i+1,j}$ as its sink. We say that player $p_{i,j}$ *starts at level* i. Also we will call player $p_{i,j}$ the *owner* of edge $e_{i,j}$, with $p_{i,j}$ *owning* only edge $e_{i,j}$ (one of the possible paths for a player to reach its sink is to use just the edge it owns).

Horizontal edges cost $6 + \varepsilon$ and $5 + \varepsilon$, vertical edges cost 12, 15, and 15 (from left to right), where ε is a small positive number. We do not refer to ε in the calculations, but simply state when relevant that the costs of horizontal edges are "more than" 6 and 5, respectively. One motivation for choosing the numbers as we do is shown in Figure 1, right.

Proof outline. It is possible to connect the sources and sinks of all the players by using all the horizontal edges and only the vertical edges on the left. For large N and small ε, the cost of this tends to 23 per level.

Our goal is to show that in a Nash equilibrium, all players use the direct link between their source and their sink. Let us assume that some players deviate from this. We start by considering a level i which is not visited by any agents with higher sources, and also not by agents that have lower sinks. In Lemma 1, we show that any agent that reaches such a level moves immediately to its sink.

We prove in Lemma 3 that as long as no agent uses any edge below its source vertex, all agents move straight to their sinks. Section 2.3 is devoted to showing that it is indeed

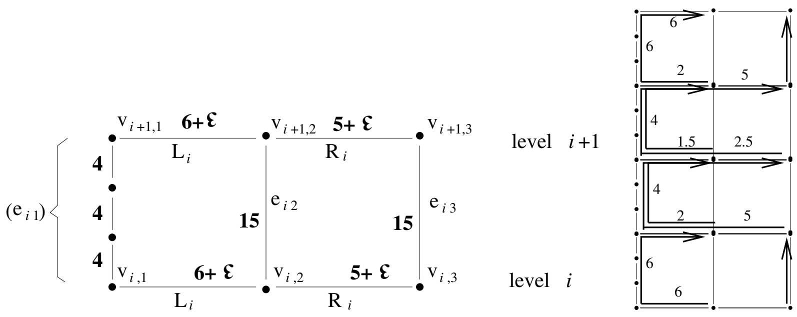


Fig. 1. On the left are two levels in our construction. The situation on the right is not a Nash equilibrium because of the added ε's on the horizontal edges. The numbers in the right figure give the costs for each agent that uses these edges.

the case that no player moves below its source vertex. To do so, we first bound the number of players that can reach a given level from below in Lemma 4. We find that there can be at most two such agents. This in turn helps us to show in Lemmas 5-7 that players that move below their sources would have to pay too much for their paths, thus showing that no agent moves below their starting levels. This immediately gives us our result, which is summarized in Theorem 1.

Due to lack of space, certain proofs are omitted.

Observation 1. *In a Nash equilibrium, all player paths are acyclic, and the graph that is formed by the paths of any pair of players is acyclic as well.*

Thus, whenever we find a cycle of one of these types, we know that this is not a Nash equilibrium.

Observation 2. *If e_{ij} is used by any player, it is used by player $p_{i,j}$.*

Proof. If this were not true, the path of any player using e_{ij} together with the path of $p_{i,j}$ forms a cycle.

Definition 1. *We call a node a* terminal *if it has a single incident edge at the graph induced by all the player paths in a Nash equilibrium.*

Observation 3. *Consider the graph induced by all the player paths in a Nash equilibrium. (This graph is not necessarily acyclic!) Any path which leads to a terminal and where all intermediate nodes have degree 2 is used only by agents with sources and/or sinks on that path. In particular, an edge which leads to a terminal is used by at most two players: the one with its source at the terminal, and the one with its sink at the terminal.*

Observation 4. *Any player that uses a vertical edge $e_{i,j}$ without owning it, must also be using at least one horizontal edge in some level $i' \le i$, and one in some level $i'' \ge i+1$.*

Players on the left We begin by making sure that players on the left always use the edge they own (the direct link between their source and sink). To do so, for all levels i, we *substitute* $e_{i,1}$ by a path of three edges $\hat{e}_{i,1}, \hat{e}_{i,2}, \hat{e}_{i,3}$ each of which has cost 4 (and thus the path of the three edges together has cost 12). Player $p_{i,1}$ is also substituted by three players $\hat{p}_{i,j} (j = 1, 2, 3)$, with $\hat{p}_{ij}$ having as source and sink the lower and upper endpoints of edge $\hat{e}_{i,j}$, respectively. (Player $\hat{p}_{i,1}$ has node $v_{i,1}$ as its source and player $\hat{p}_{i,3}$ has node $v_{i+1,1}$ as its sink.) One can now see that the players $\hat{p}_{i,j} (j = 1, 2, 3)$ will never deviate from their own edges; each such player would have to share *two* edges of cost 4 with only their owners, since its sink and/or its source would be terminals. Given that these players will never deviate, we will treat them as one player $p_{i,1}$, and the path $\hat{e}_{i,1}, \hat{e}_{i,2}, \hat{e}_{i,3}$ as the single edge $e_{i,1}$, with $p_{i,1}$ using edge $e_{i,1}$ in any Nash Equilibrium.

2.1 Separators

Definition 2. *Level i is called a* separator *if no player with source above level i and no player with sink below level i visits level i.*

Lemma 1. *Let level i be a separator. Let $p = p_{i-1,2}$ and $p' = p_{i-1,3}$.*

1. *If player p arrives at level i via edge $e_{i-1,1}$ ($e_{i-1,3}$), and there is no other player which uses that edge besides its owner, then p uses edge L_i (R_i), and shares that edge with at most 2 players.*
2. *If player p' arrives at level i via $e_{i-1,1}$ together with p, it uses L_i and R_i, and pays at least 4 for them. In particular, there are at most 4 agents on L_i.*
3. *If p' arrives at level i via edge $e_{i-1,2}$, it uses edge R_i, and pays at least $5/2$ for it.*

Lemma 2. *Let level i be a separator. Let $p = p_{i-1,2}$ and $p' = p_{i-1,3}$. Assume that player p' does not move below its source. If it arrives at level i via edge $e_{i-1,1}$, then there is some other player which uses $e_{i-1,1}$ besides its owner.*

Proof. The first three edges on the path of p' are R_{i-1}, L_{i-1}, and $e_{i-1,1}$ in this case. Consider agent p. It cannot use edge $e_{i-1,3}$ (in that case, by Observation 2, player p' would use it too) or edge $e_{i-1,1}$ (assumption) in this case, so p uses edge $e_{i-1,2}$. This means that p' cannot use edge L_i (Observation 1). It also implies that edges $e_{i-1,2}$ and $e_{i-1,3}$ are used by at most three players, since they are not used by any left player, any player with source at level $i + 1$ or higher, or p, leaving only $p_{i,2}, p_{i,3}$ and $p' = p_{i-1,3}$ as candidates. Therefore, the cost of these edges is at least 5 to any player. Player p' must use one of them. In addition, p' pays 6 for edge $e_{i-1,1}$, and also 6 for edge $e_{i,1}$ as long as player $p_{i,2}$ or $p_{i,3}$ do not join it. But in that case, the total cost of p' is at least $6+6+5 > 15$, a contradiction. So $p_{i,2}$ or $p_{i,3}$ must be on $e_{i,1}$. Only one of them can in fact be there since one of the vertical edges $e_{i,2}$ and $e_{i,3}$ must be in use. This means that the cost for $e_{i,1}$ is 4 in this case. However, in this case, the edge that p' uses to come back down to level i costs 7.5. We conclude that if p' pays 6 for $e_{i,1}$, its total cost is at least $6 + 6 + 5 > 15$, and otherwise, its total cost is at least $6 + 4 + 7.5 > 15$. In both cases, this implies that this is not a Nash equilibrium.

Lemma 3. *Consider a Nash equilibrium in which no agent uses any edge below its source. Then all agents move straight to their sinks.*

Proof. We twice use induction. We first show that all players on the right move straight to their sinks, while players in the middle either move straight to their sink or move left, up, and immediately right. Using this, we then show that all players in the middle move straight to their sink.

Consider first level 1. By the assumption of this Lemma, level 2 is a separator. If player $p_{1,3}$ uses L_1, player $p_{1,2}$ must do this as well by Lemma 2. In addition, in this case $p_{1,3}$ uses R_1 as well, but $p_{1,2}$ does not, and neither does any other player. Both $p_{1,3}$ and $p_{1,2}$ then use edge $e_{1,1}$, and then $p_{1,3}$ continues via edge L_2 and R_2 by Lemma 1, Case 2. This fixes its entire path. We can now calculate the cost for this path depending on the first edge on the path of $p_{2,2}$.

If this is L_2, the cost is more than $5 + 3 + 4 + 1.5 + 2.5 = 16$ ($p_{2,2}$ is not on R_2 in this case, and neither is $p_{1,2}$). If the first (and only) edge is $e_{2,2}$, the cost is more than $5 + 3 + 4 + 2 + 2.5 = 19$. If the first edge is R_2, the cost is more than $5 + 3 + 4 + 3 + 2.5 = 17.5$ ($p_{2,3}$ is not on R_2 in this case, because $p_{2,2}$ uses $e_{2,3}$, Observation 2). In all cases, this is too much.

This shows that $p_{1,3}$ does not use edge L_1. Suppose that $p_{1,3}$ uses R_1. It then uses $e_{1,2}$ together with $p_{1,2}$ (Observation 2). Since R_2 is used by at most one of the players $p_{2,2}$ and $p_{2,3}$ (by Observation 1 and because no agents move down below their source), $p_{1,3}$ pays more than $5 + 15/2 + 5/2 = 15$, a contradiction. The exact same calculation shows that $p_{1,2}$ does not use R_1.

The only case left open is the one where $p_{1,2}$ uses L_1, but $p_{1,3}$ does not. However, in this case, due to Lemma 1, Case 1, it also uses L_2 to reach its sink, making level 3 a separator, because level 3 is not visited by $p_{1,2}$ or $p_{1,3}$. Note that if $p_{1,2}$ does move directly to its sink from its source, then level 3 is a separator too.

We can now continue the proof by induction. Consider a level i and assume that all lower players on the right move straight to their source, where lower players in the middle might deviate and use the left edge. Also by induction, assume that level $i + 1$ is a separator, so that we can use the same lemmas as in the base case. Compared to the calculations above for the case where $p_{1,3}$ uses L_1, the only change is that edge L_i might cost only $2 + \varepsilon/3$ instead of $3 + \varepsilon/2$, since at most one additional agent ($p_{i-1,2}$) may be using it. This still gives a total cost of more than 15 in all cases, completing the first part of the proof.

We can now prove, also using induction, that agents in the middle move straight to their source. If $p_{1,2}$ uses L_1, it pays more than $6 + 6 + 3 = 15$ since L_1 now costs more than 3, so it does not do that. By induction, if no player below level i deviates, we find the same calculation for any middle player that moves left. This completes the proof.

2.2 The Number of Agents That Visit a Certain Level

Definition 3. *Let S_ℓ be a set of players that visit a* horizontal *edge at or above level ℓ and that all have sinks at or below level ℓ.*

Observation 4 implies the following Corollary:

Corollary 1. *For any level i, any player with source below i that uses an edge $e_{i-1,j}$, $j \in \{1, 2, 3\}$, without owning it, belongs to S_i.*

Lemma 4. *We have $|S_\ell| \leq 2$.*

Corollary 2. *Any horizontal edge is used by at most four agents, any vertical edge by at most five.*

2.3 Agents Do Not Move Down

Due to Lemma 3, all we need to show is that no player moves below its starting level in a Nash equilibrium. Consider the topmost level i such that there is a player A, with source at level i, that moves below i. Denote the other player that has its source on level i and that does not start on the left by A'. (Note that the player with source $v_{i,1}$ never deviates). A must visit levels below i before it reaches level $i+1$. Otherwise, either i reaches its sink before going down to $i-1$, or it will have to form a cycle within its path to go back up to $i+1$. Similarly, since A goes both below and above level i, it cannot use both L_i and R_i. In the following, we will be repeatedly making use of Lemma 4 and Corollary 2, and the fact that no player with source above level i ever visits a level $i' \leq i$ (by definition of A, A').

Lemma 5. *A does not move first horizontally and then down.*

Proof. Assume that A uses first one of the horizontal edges of level i and then immediately goes down. Since A has to go back up to level i, it creates a path connecting all three nodes of level i using only edges incident to nodes of levels $j \leq i$. This implies that there is no player p with source at level $i-1$ or below, that visits level $i+1$. To see this, note that after reaching $i+1$, p would eventually have to go back down to level i, thus creating another path connecting two nodes of i, this time containing only edges incident to nodes in levels $j' \geq i$ (with at least one vertical edge incident to a node of level $i+1$). Therefore, the paths of A and p would form a cycle. By definition of A, there is also no player with source above level i that visits level i.

Let c_1 be the column that A starts from, $c_2 \neq c_1$ the column it reaches after using the first horizontal edge, and c_3 the remaining column of the grid. Note that since A uses a horizontal edge of level i, one of c_1, c_2 must be the middle column. A cannot create a cycle going from its source back to level i, therefore it must use edge $e = e_{i-1,c_3}$. Moreover, edge $(v_{i,c_3}, v_{i,2})$ is not used by any player, otherwise a cycle with A's path would be formed. Therefore, any player on e that does not own it (including A), must also use $e' = e_{i,c_3}$. Given that no player with source below i visits level $i+1$, and no player with source above i visits level $i-1$, the edges e', e'' can only by used by the owners and A, A'. Therefore, A pays at least $2 \cdot \frac{12}{3} = 8$ for them.

Consider now the first edge that A uses to reach level $i-1$. By Corollary 2 there are at most 5 players using it, and thus A pays at least $\frac{12}{5} > 2$ for it. Finally, A visits both the first column and the third column of the grid, therefore it must use at least two "right" horizontal edges (of cost 5), and at least two "left" horizontal edges (of cost 6), each of which can be used by at most four players (by Corollary 2). Thus, A pays at least $2 \cdot \frac{6+5}{4} = 5.5$ for horizontal edges, implying a total cost more than $8+2+5.5 > 15$, a contradiction.

Lemma 6. *If A starts in the middle column, it does not move straight down from its source.*

Lemma 7. *If A starts in the right column, it does not move straight down from its source.*

Proof. Assume that A goes straight down from its source. We denote by e that first edge down (i.e., $e = e_{i-1,3}$). Let e' be the edge that A uses to reach level i again, after going down. By Lemma 5 and Lemma 6, A' does not move down which means that A' does not use e, e'. Any other player using them, apart from A and the owners, will belong to S_i (remember that no player with source above i visits a level below $i + 1$). Therefore A shares e, e' with at most 3 more players (the owners and two more players that will belong to S_i). Let e'' be the edge A uses to reach level $i + 1$ from i. Any player on e'' (apart from the owner) will belong to S_{i+1} together with A. Again, since $|S_{i+1}| \leq 2$, A shares e'' with at most two more players (the owner and one more player that will belong to S_{i+1}). If any of e', e'' is in the left column, then the path of A must cross from the right side of the grid to the left and back, implying a total cost of at least $15/4$ (for e) $+12/4$ (for e') $+12/3$ (for e'') $+2 \cdot \frac{6+5}{4} > 15$. If, on the other hand, none of e', e'' are in the left column, then the total cost of A is more than $15/4$ (for e) $+15/4$ (for e') $+15/3$ (for e'') $+2 \cdot 5/4 = 15$.

Theorem 1. *The price of stability in undirected networks is at least* $42/23 > 1.826$.

Proof. Due to Lemma 5, Lemma 6 and Lemma 7, no agents move down below their source. Therefore, by Lemma 3, all agents move straight to their sink in the (unique) Nash equilibrium. On every level, the total cost of the agents in the Nash equilibrium is $12 + 15 + 15 = 42$, whereas the optimal cost is only $12 + 6 + 5 = 23$. The optimal solution has an additional cost of 11 for the two horizontal edges on level 1, but this cost is negligible for large N.

3 Two and Three Players

We will describe here a lower and an upper bound for three players, as well as an unconditional upper bound for two players. Again, some proofs are omitted due to lack of space.

Lower bound for three players. Figure 2 shows a three-player instance where the best Nash equilibrium has cost $37/24$ times that of OPT. Node s_i, t_i is the source, destination, respectively of player i, $i \in \{1, 2, 3\}$. The optimal solution would only use the edges (s_1, s_2), (s_2, s_3), (s_3, t_1), (t_1, t_2), while the Nash solution uses the direct edges (s_1, t_1), (s_2, t_2), (s_3, t_3). The cost of the optimal solution sums then up to $48+4\varepsilon$, while the Nash Equilibrium solution has cost 74. We have therefore the following theorem.

Theorem 2. *In the fair cost sharing network design game with three players, the price of stability is at least* $74/48 \approx 1.5417$.

Upper bound for three players. Given an instance of our problem, let OPT refer to an optimal solution. We refer to the union of the players' paths at OPT as the *OPT*

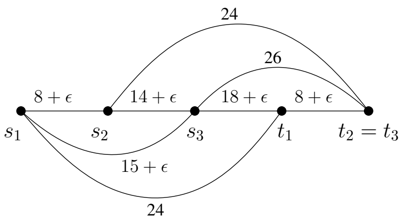


Fig. 2. A three-player instance with price of stability more than 1.54

graph. Recall that our game is a potential game, with potential function $\Phi(X) = \sum_{e \in E} c_e H(X_e)$ where c_e is the cost of edge e, $H(x)$ is the xth harmonic number, X is a game state or solution, and X_e is the number of players on edge e in X. Let N be a potential minimizing Nash solution (or, alternatively, N can be defined as a Nash solution reached by starting from OPT and making alternating best-response moves). Hence, we have

$$\Phi(N) \leq \Phi(OPT). \tag{1}$$

We now give names for various sets of edges, each of which may or may not be empty. Let A, B, and C be the sets of edges that player 1, player 2, and player 3 (respectively) use alone in N. Let S_{ij} for $i = 1 \ldots 2$ and $j = i+1 \ldots 3$ be the set of edges that players i and j alone *share* in N. Let S_{123} be the set of edges that all three players share in N. Let $A^*, B^*, C^*, S^*_{12}, S^*_{13}, S^*_{23}$ and S^*_{123} be defined analogously for OPT. We will also use the same names to refer to the total *cost* of the edges in each set.

Let $C(X)$ refer to the cost of the solution X and let $C_i(X)$ refer to the cost just to player i of the solution X. By definition, we have

$$\begin{aligned}
C(N) &= A + B + C + S_{12} + S_{23} + S_{13} + S_{123} \\
C(OPT) &= A^* + B^* + C^* + S^*_{12} + S^*_{23} + S^*_{13} + S^*_{123} \\
C_1(N) &= A + \frac{S_{12}}{2} + \frac{S_{13}}{2} + \frac{S_{123}}{3} \\
C_2(N) &= B + \frac{S_{12}}{2} + \frac{S_{23}}{2} + \frac{S_{123}}{3} \\
C_3(N) &= C + \frac{S_{13}}{2} + \frac{S_{23}}{2} + \frac{S_{123}}{3}
\end{aligned}$$

Lemmas 8, 9 show how to bound the POS depending on whether $S^*_{123} > 0$ or not.

Lemma 8. *In the fair cost sharing network design game with three players, if all three players share at least one edge of positive cost in the optimal solution, the price of stability is at most* $33/20 = 1.65$.

Proof. First observe that the edges in the set S^*_{123} must form a contiguous path, that is, once the three players' paths in the OPT graph merge, as soon as one player's path breaks off, the three may never merge again. (Otherwise the OPT graph would have a cycle, contradicting the fact that it is an optimal solution.) Without loss of generality, we can exchange the labels on the endpoint vertices so that the three endpoints on the same side of the edges in S^*_{123} are all source endpoints, and the three endpoints on the other side are all destination endpoints.

Then observe that at least one of S^*_{12}, S^*_{23}, and S^*_{13} must be empty. Otherwise the OPT graph would have a cycle, contradicting the definition of OPT. Without loss of generality, we assume that S^*_{13} is empty, hence $S^*_{13} = 0$ and $C(OPT) = A^* + B^* + C^* + S^*_{12} + S^*_{23} + S^*_{123}$.

We know by definition of N that each player i pays not more at N than by unilaterally defecting to any alternate $s_i - t_i$ connection path. The right hand sides of each of the following inequalities represents an upper bound on the cost of a feasible alternate $s_i - t_i$ path for each player i. The existence of these alternate paths depends on the assumption that the OPT graph is connected and $S^*_{13} = 0$.

$$C_1(N) \le A^* + B^* + S^*_{23} + \frac{B}{2} + \frac{S_{12}}{2} + \frac{S_{23}}{3} + \frac{S_{123}}{3} \tag{2}$$

$$C_2(N) \le B^* + A^* + S^*_{23} + \frac{A}{2} + \frac{S_{12}}{2} + \frac{S_{13}}{3} + \frac{S_{123}}{3} \tag{3}$$

$$C_2(N) \le B^* + C^* + S^*_{12} + \frac{C}{2} + \frac{S_{23}}{2} + \frac{S_{13}}{3} + \frac{S_{123}}{3} \tag{4}$$

$$C_3(N) \le C^* + B^* + S^*_{12} + \frac{B}{2} + \frac{S_{23}}{2} + \frac{S_{12}}{3} + \frac{S_{123}}{3} \tag{5}$$

To interpret the above inequalities intuitively, consider for example the first inequality. It states the fact that player 1 pays an amount at Nash that is at most the cost of unilaterally deviating and instead taking the path in the OPT graph from s_1 to s_2 where player 2's OPT path begins (possibly using edges from A^*, B^*, and S^*_{23}), then following along player 2's path in N from s_2 to t_2 (using edges from B, S_{12}, S_{23}, and S_{123}), then taking edges in the OPT graph from t_2 to t_1 (again possibly using edges from A^*, B^*, and S^*_{23}). The costs of S^*_{12} and S^*_{123} need not be included in the right-hand side of the first inequality for the following reasoning. Recall that by assumption, source vertices are on one side of the edges in S^*_{123} and sink vertices are on the other side of the edges in S^*_{123}, so traversing any edges in S^*_{123} is not necessary for player 1 to go from s_1 to s_2 or from t_2 to t_1 in the OPT graph. Also note that the edges in S^*_{12} must be adjacent to the contiguous path formed by edges in S^*_{123} (since otherwise, the OPT graph would contain a cycle), and so in fact, s_1 and s_2 are on one side of $S^*_{12} \cup S^*_{123}$, while t_1 and t_2 are on the other.

From inequality (1) and the assumption that $S^*_{13} = 0$, we can say

$$A + B + C + \frac{3}{2}(S_{12} + S_{13} + S_{23}) + \frac{11}{6} S_{123} \le A^* + B^* + C^* + \frac{3}{2}(S^*_{12} + S^*_{23}) + \frac{11}{6} S^*_{123}. \tag{6}$$

Scaling the inequalities 2 and 5 each by $10/99$, 3 and 4 each by $8/99$, and 6 by $6/11$, then summing all five resulting inequalities yields

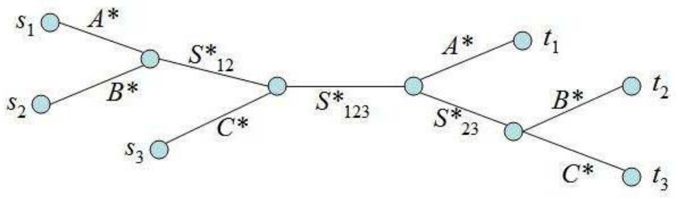


Fig. 3. A sample OPT graph. Each edge is labeled with the name of the set of edges it belongs to. Each edge here may represent a sequence of edges forming a path. Note that more generally, any of the sets A^*, B^*, C^*, S^*_{12}, S^*_{23}, and S^*_{13} could be empty.

$$\begin{aligned}\frac{20}{33}(A+B+C) + \frac{257}{297}S_{13} + \frac{245}{297}(S_{12}+S_{23}) + S_{123}&\\ \leq \frac{8}{11}(A^*+C^*) + \frac{10}{11}B^* + S^*_{12} + S^*_{23} + S^*_{123}.&\end{aligned} \tag{7}$$

Hence $20/33C(N) \leq C(OPT)$.

Lemma 9. *In the fair cost sharing network design game with three players, if no positive-cost edge is shared by all three players in the optimal solution, the price of stability is at most 3/2.*

We are now ready to present our main theorem of this section.

Theorem 3. *In the fair cost sharing network design game with three players, the price of stability is at most* $33/20 = 1.65$.

Proof. All possible OPT graph structures are handled by Lemmas 9 and 8. The worst upper bound for price of stability over these two exhaustive cases is that given by Lemma 8.

Upper bound for two players. Anshelevich et al. [4] gave a two player lower bound instance for our problem showing that the price of stability is at least $4/3$. They then show that if both players share a sink, the price of stability is at most $4/3$. The following theorem states an unconditional two-player upper bound on the price of stability of $4/3$.

Theorem 4. *In the fair cost sharing network design game with two players, price of stability is at most* $4/3$.

4 Conclusions

The lower bound instance that we use for large n could be generalized by adding more columns. However, it seems that this would require a significantly longer and more involved proof. More importantly, we believe that even with an unbounded number of columns we could only show a lower bound of a small constant. Hence, the question of whether the price of stability grows with n remains open. We conjecture that it is in fact constant.

Acknowledgments. The authors would like to thank Susanne Albers, Elliot Anshelevich, Jian Li, Kirk Pruhs and Alexander Souza for interesting discussions.

References

1. Tardos, E., Wexler, T.: Network Formation Games and the Potential Function Method. In: Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
2. Koutsoupias, E., Papadimitriou., C.H.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
3. Papadimitriou, C.H.: Algorithms, games, and the internet. In: Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 749–753 (2001)
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. In: 45th Symposium on Foundations of Computer Science (FOCS), pp. 295–304 (2004)
5. Rosenthal, R.: A class of games possessing pure-strategy Nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
6. Roughgarden, T., Tardos, E.: How bad is selfish routing? J. ACM 49(2), 236–259 (2002)
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 (STOC), pp. 67–73 (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. Awerbuch, B., Azar, Y., Epstein, A.: Large the price of routing unsplittable flow. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 57–66 (2005)
10. Chen, H.L., Roughgarden, T.: Network design with weighted players. In: SPAA 2006: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures, pp. 29–38. ACM, New York (2006)
11. Albers, S.: On the value of coordination in network design. In: SODA 2008: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 294–303. Society for Industrial and Applied Mathematics, Philadelphia (2008)
12. Fiat, A., Kaplan, H., Levy, M., Olonetsky, S., Shabo, R.: On the price of stability for designing undirected networks with fair cost allocations. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 608–618. Springer, Heidelberg (2006)
13. Li, J.: An O (lognloglogn) upper bound on the price of stability for undirected Shapley network design games. Information Processing Letters (2009)
14. 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)
15. Monderer, D., Shapley, L.S.: Potential games. Games and Economic Behavior 14, 124–143 (1996)
16. Olonetsky, S.: On the price of stability for designing undirected networks with fair cost allocations. Master's thesis, Tel-Aviv University (2006)

Finding Dense Subgraphs in $G(n, 1/2)$

Atish Das Sarma[1], Amit Deshpande[2], and Ravi Kannan[2]

[1] Georgia Institute of Technology[⋆]
atish@cc.gatech.edu
[2] Microsoft Research-Bangalore
amitdesh,kannan@microsoft.com

Abstract. Finding the largest clique in random graphs is a well known hard problem. It is known that a random graph $G(n, 1/2)$ almost surely has a clique of size about $2 \log n$. A simple greedy algorithm finds a clique of size $\log n$, and it is a long-standing open problem to find a clique of size $(1 + \epsilon) \log n$ in randomized polynomial time. In this paper, we study the generalization of finding the largest subgraph of any given edge density. We show that a simple modification of the greedy algorithm finds a subset of $2 \log n$ vertices with induced edge density at least 0.951. We also show that almost surely there is no subset of $2.784 \log n$ vertices whose induced edge density is at least 0.951.

1 Introduction

Finding the largest clique is a notoriously hard problem, even on random graphs. It is known that the clique number of a random graph $G(n, 1/2)$ is almost surely either k or $k+1$, where $k = \lceil 2 \log n - 2 \log \log n - 1 \rceil$ (Section 4.5 in [1], also [2]). However, a simple greedy algorithm finds a clique of size only $\log n \,(1 + o(1))$, with high probability, and finding larger cliques – that of size even $(1+\epsilon) \log n$ – in randomized polynomial time has been a long-standing open problem [3]. In this paper, we study the following generalization: given a random graph $G(n, 1/2)$ find the largest subgraph with edge density at least $(1 - \delta)$. We show that a simple modification of the greedy algorithm finds a subset of $2 \log n$ vertices whose induced subgraph has edge density at least 0.951, with high probability. To complement this, we show that almost surely there is no subset of $2.784 \log n$ vertices whose induced subgraph has edge density 0.951 or more.

We use $G(n, p)$ to denote a random graph on n vertices where each pair of vertices appears as an edge independently with probability p. We use V to denote its set of vertices and E to denote its set of edges. Moreover, given two subsets $S \subseteq V$ and $T \subseteq V$, we use $E(S, T)$ to denote the set of edges with one endpoint in S and another endpoint in T. The density of the subgraph induced by vertices in S is given by

$$\text{density}\,(S) = \frac{|E(S, S)|}{\binom{|S|}{2}}.$$

Therefore, the expected density of $G(n, 1/2)$ is $1/2$ and the density of any clique is 1.

[⋆] Work done while at Microsoft Research.

In Section 2 we describe our algorithm for finding subgraphs of density $1 - \delta$. We give a bound on the largest subgraph of density $1 - \delta$ in the following Section 3. Finally, in Section 4, we present some open problems.

2 Algorithm for Finding Large Subgraph of Density 1 −

In this section, we describe our algorithm and give a relationship between the size of the subgraph obtained by the algorithm, and its density. In particular, we show that the algorithm can be used to obtain a subset of $2 \log n$ vertices of density 0.951, with high probability.

```
GREEDY ALGORITHM TO PICK A DENSE SUBGRAPH:
Input: a random graph $G(n, 1/2)$ and $\delta > 0$.
Output: a subset $S \subseteq V$ of size $k = 2 \log n$.

1. Partition the vertices into disjoint sets $V = V_1 \cup V_2 \cup \cdots \cup V_k$, each of
   size $n/k$.
2. Initialize $S_0 = \emptyset$.
3. For $i = 0$ to $k - 1$ do:
   (a) Pick $v_{i+1} \in V_{i+1}$ that has the maximum number of edges to $S_i$, i.e.,
           $v_{i+1} = \operatorname{argmax}_{v \in V_{i+1}} |E(v_{i+1}, S_i)|.$
   (b) $S_{i+1} \leftarrow S_i \cup \{v_{i+1}\}$.
4. Return $S = S_{k-1}$.
```

Notice that the algorithm first partitions all nodes into k random subsets of the same size, and then picks one vertex from each partition. This partitioning is necessary to argue about independence in our analysis of choosing vertices greedily.

In the analysis below, $H(\delta)$ is the standard notation of the Shannon entropy function, which is $-(\delta \log \delta + (1 - \delta) \log(1 - \delta))$. The following lemma gives a lower bound on the number of edges we can expect to add to our subgraph, for the i-th vertex added by the algorithm.

Lemma 1. *For any $0 \le i \le k$ and δ_i that satisfies*

$$H(\delta_i) \ge 1 - \frac{1}{i} \log \left(\frac{n}{2k \ln(\log n)} \right),$$

we have

$$\mathsf{Pr} \left(|E(v_{i+1}, S_i)| \ge (1 - \delta_i)\, i \right) \ge 1 - \frac{1}{\log^2 n}.$$

Proof. We know by the previous results, that as long as $k < \log n$, the vertex added has all edges to S_{k-1}. Consider $k \geq \log n$. The algorithm has $\frac{n}{l}$ vertices to choose from. The expected number of vertices among these, with at least $(1 - \delta_k)k$ vertices is given by,

Fix $v \in V_{i+1}$. The probability that v has at least $(1 - \delta_i)i$ edges to S_i is

$$\mathsf{Pr}\left(|E(v, S_i)| \geq (1 - \delta_i)i\right) = \sum_{t=(1-\delta_i)i}^{i} \binom{i}{t} 2^{-i} = 2^{(H(\delta_i)+o(1)-1)i},$$

where $H(\delta) = -\delta \log \delta - (1 - \delta)\log(1 - \delta)$ is the Shannon entropy (here log is taken with base 2). Using independence of these events for different $v \in V_{i+1}$, we get

$$\begin{aligned}\mathsf{Pr}\left(|E(v, S_i)| < (1 - \delta_i)i,\ \forall v \in V_{i+1}\right) &\leq \left(1 - 2^{(H(\delta_i)-1)i}\right)^{n/k} \\ &\leq \left(1 - \frac{2k \ln(\log n)}{n}\right)^{n/k} \\ &\leq \frac{1}{\log^2 n}.\end{aligned}$$

Therefore,

$$\mathsf{Pr}\left(|E(v_{i+1}, S_i)| \geq (1 - \delta_i)\, i\right) \geq 1 - \frac{1}{\log^2 n}.$$

We now give a union bound over all k additions of vertices, using the previous lemma.

Lemma 2

$$\mathsf{Pr}\left(|E(S, S)| \geq \sum_{i=0}^{k-1} (1 - \delta_i)\, i\right) \to 1 \ \textit{as}\ n \to \infty.$$

Proof. Since $V_1, V_2, \ldots, V_k$ are disjoint, using independence and Lemma 1 we get

$$\begin{aligned}\mathsf{Pr}\left(|E(S, S)| \geq \sum_{i=0}^{k-1} (1 - \delta_i)\, i\right) &\geq \prod_{i=0}^{k-1} \mathsf{Pr}\left(|E(v_{i+1}, S_i)| \geq (1 - \delta_i)\, i\right) \\ &\geq \left(1 - \frac{1}{\log^2 n}\right)^{k-1} \\ &\geq e^{1/\log n} \qquad \text{using } k = 2\log n\end{aligned}$$

The point is that we are picking exactly one vertex from each vertex set/partition, and hence do not lose any randomness or independence of the edges. This now gives us a bound on the minimum number of edges one can expect, w.h.p., in the chosen set of k vertices. We are not able to express, in a closed form, the size of a subgraph obtainable using this algorithm for a specific density. Therefore, we state the best density one can guarantee w.h.p. for $k = 2\log n$. This is stated as a theorem below, which we prove subsequently.

Theorem 1. *Our algorithm produces a subset $S \subseteq V$ of size $k = 2 \log n$ such that* density $(S) \gtrsim 0.951$*, almost surely.*

Proof. From Lemma 2 we have that, almost surely,

$$\begin{aligned}
|E(S,S)| &\geq \sum_{i=0}^{k-1} (1-\delta_i)\, i \\
&\geq \sum_{i=0}^{k-1} \left(1 - H^{-1}\left(1 - \frac{1}{i}\log\left(\frac{n}{2k\ln(\log n)}\right)\right)\right) i \\
&= \sum_{i=0}^{k-1} i - \sum_{i=\log m}^{k-1} iH^{-1}\left(1 - \frac{\log m}{i}\right) \\
&= \binom{k}{2} - \sum_{i=\log m}^{k-1} iH^{-1}\left(1 - \frac{\log m}{i}\right), \qquad (1)
\end{aligned}$$

where $m = n/2k\ln(\log n)$. Here we use the fact that we can choose $\delta_i = 0$ for the first $\log m$ steps. Now let $k - 1 = (1+\alpha)\log m$. Then

$$\begin{aligned}
&\sum_{i=\log m}^{k-1} iH^{-1}\left(1 - \frac{\log m}{i}\right) \\
&= \sum_{i=\log m}^{(1+\alpha)\log m} iH^{-1}\left(1 - \frac{\log m}{i}\right) \\
&= \sum_{t=0}^{\alpha \log m} (\log m + t)\, H^{-1}\left(1 - \frac{\log m}{\log m + t}\right) \\
&= \log^2 m \sum_{x=0}^{\alpha} (1+x) H^{-1}\left(1 - \frac{1}{1+x}\right) \\
&\leq \log^2 m \int_0^{\alpha} (1+x) H^{-1}\left(1 - \frac{1}{1+x}\right) dx, \qquad (2)
\end{aligned}$$

Now using Equations (1) and (2) we have

$$\begin{aligned}
\text{density}\,(S) &= \frac{|E(S,S)|}{\binom{k}{2}} \\
&\geq 1 - \frac{\log^2 m}{\binom{k}{2}} \int_0^{\alpha} (1+x) H^{-1}\left(1 - \frac{1}{1+x}\right) dx \\
&\geq 1 - \frac{1}{2}\,(1+o(1)) \int_0^{\alpha} (1+x) H^{-1}\left(1 - \frac{1}{1+x}\right) dx \\
&\gtrsim 0.951.
\end{aligned}$$

using

$$\alpha = \frac{k}{\log m} - 1 = \frac{2\log n}{\log n - \log\left(4\log n \cdot \ln(\log n)\right)} - 1 = 1 + o(1).$$

and computing an upper bound on the integral numerically.

3 Upper Bound on Largest Subgraph of Density 1 −

In this section, we upper bound the size of the largest subgraph of density $1-\delta$ in $G(n, 1/2)$.

Theorem 2. *A random graph $G(n, 1/2)$ has no subgraph of size*

$$\frac{2\log n + 2\log e}{1 - H(\delta) - o(1)} + 1$$

and density at least $1-\delta$, almost surely. In particular, there is no subgraph of size $2.784\log n$ and density at least 0.951, almost surely.

Proof. For every $S \subseteq V$ of size k, define an indicator random variable X_S as follows.

$$X_S = \begin{cases} 1 & \text{if } S \text{ induces a subgraph of density} \geq 1-\delta \\ 0 & \text{otherwise.} \end{cases}$$

Thus

$$\mathsf{E}\left[X_S\right] = \sum_{i=(1-\delta)\binom{k}{2}}^{\binom{k}{2}} \binom{\binom{k}{2}}{i} 2^{-\binom{k}{2}} = 2^{(H(\delta)+o(1)-1)\binom{k}{2}}.$$

By linearity of expectation, expected number of subgraphs of size k and density $\geq 1-\delta$ follows.

$$\begin{aligned}
\mathsf{E}\left[\sum_{S\,:\,|S|=k} X_S\right] &= \sum_{S\,:\,|S|=k} \mathsf{E}\left[X_S\right] \\
&= \binom{n}{k} 2^{(H(\delta)+o(1)-1)\binom{k}{2}} \\
&\leq \left(\frac{en}{k}\right)^k \left(2^{(H(\delta)+o(1)-1)\frac{k-1}{2}}\right)^k \\
&= \left(\frac{en}{k} \cdot 2^{(H(\delta)+o(1)-1)\frac{k-1}{2}}\right)^k \\
&= \left(\frac{2^{(1-H(\delta)-o(1))\frac{k}{2}}}{k} \cdot 2^{(H(\delta)+o(1)-1)\frac{k-1}{2}}\right)^k \\
&= \left(\frac{2^{(1-H(\delta)+o(1))/2}}{k}\right)^k \to 0, \quad \text{as } n \to \infty,
\end{aligned}$$

using

$$k = \frac{2\log n + 2\log e}{1 - H(\delta) - o(1)} + 1.$$

Therefore, by Markov inequality we have

$$\mathsf{Pr}\left(\sum_{S\,:\,|S|=k} X_S \geq 1\right) \leq \mathsf{E}\left[\sum_{S\,:\,|S|=k} X_S\right] \to 0,$$

as $n \to \infty$. Or in other words, almost surely there is no subset of k vertices that induce a subgraph of density at least $1 - \delta$.

Notice that for density 0.951, the gap/ratio between the largest subgraph that exists and the largest subgraph that we can find is smaller than in the case of cliques. This is interesting, although not entirely unexpected as for density 0.5, the whole graph can be output. This ratio for density 0.951 is however significantly smaller than 2; it is $2.784/2 = 1.392$.

4 Conclusions

For a concrete open problem, is there a polynomial time algorithm that outputs a subgraph of density $1 - \epsilon$ and size $2\log n$ for any choice of $\epsilon > 0$?

Are there simple algorithms that beat the density bound of 0.95 for subgraphs of size $2\log n$. If not, what is the maximum density obtainable for a subgraph of size $2\log n$? Spectral techniques could be tried. Is there an $O(n^{\log n})$ time algorithm that finds the largest clique in $G(n, 1/2)$? We thank an anonymous reviewer for pointing out a solution to this: Simply enumerate all cliques of all sizes, generating them from smaller ones to bigger ones; once all cliques of size k have been found, try to add to each of them every vertex and see if it can be expanded. The expected number of cliques of size k is $\binom{n}{k}2^{-\binom{k}{2}} < (n2^{-(k-1)/2})^k$, and with high probability the number of cliques of each size does not exceed this expectation by much. The function $(n2^{-(k-1)/2})^k$ attains its maximum around $k = \log_2 n$, and its value for this k is roughly $n^{0.5\log_2 n}$, showing that the above algorithm will run in time $n^{(0.5+o(1))\log_2 n}$ (and find the largest clique with high probability over the input graphs). Can one do better than this?

References

1. Alon, N., Spencer, J.: The probabilistic method, 2nd edn. Wiley Interscience, Hoboken (2000)
2. Bollobás, B.: Random graphs. Academic Press, London (1985)
3. Karp, R.: The probabilistic analysis of some combinatorial search algorithms, pp. 1–19 (1976)

Parameterized Analysis of Paging and List Update Algorithms

Reza Dorrigiv[1], Martin R. Ehmsen[2], and Alejandro López-Ortiz[1]

[1] Cheriton School of Computer Science, University of Waterloo, Canada
{rdorrigiv,alopez-o}@uwaterloo.ca
[2] Department of Mathematics and Computer Science,
University of Southern Denmark, Odense, Denmark
ehmsen@imada.sdu.dk

Abstract. It is well-established that input sequences for paging and list update have locality of reference. In this paper we analyze the performance of algorithms for these problems in terms of the amount of locality in the input sequence. We define a measure for locality that is based on Denning's working set model and express the performance of well known algorithms in term of this parameter. This introduces parameterized-style analysis to online algorithms. The idea is that rather than normalizing the performance of an online algorithm by an (optimal) offline algorithm, we explicitly express the behavior of the algorithm in terms of two more natural parameters: the size of the cache and Denning's working set measure. This technique creates a performance hierarchy of paging algorithms which better reflects their intuitive relative strengths. Also it reflects the intuition that a larger cache leads to a better performance. We obtain similar separation for list update algorithms. Lastly, we show that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the parameterized case, which matches experimental results.

1 Introduction

The competitive ratio, first introduced formally by Sleator and Tarjan [34], has served as a practical measure for the study and classification of online algorithms. An algorithm (assuming a minimization problem) is said to be α-competitive if the cost of serving any specific request sequence never exceeds α times the cost of an optimal *offline* algorithm which knows the entire sequence. The competitive ratio is a relatively simple measure to apply yet powerful enough to quantify, to a large extent, the performance of many online algorithms. Notwithstanding the wide applicability of competitive analysis, it has been observed by numerous researchers (e.g. [9,11,28,37,14]) that in certain settings the competitive ratio produces results that are too pessimistic or otherwise found wanting. Indeed, the original paper by Sleator and Tarjan discusses the various drawbacks of the competitive ratio and uses resource augmentation to address some of the observed drawbacks.

A well known example of the shortcomings of competitive analysis is the paging problem. A paging algorithm mediates between a slower and a faster memory. Assuming a cache of size k, it decides which k memory pages to keep in the cache without the benefit of knowing in advance the *sequence* of upcoming page requests. After receiving the i^{th} page request the online algorithm must decide which page to evict, in the event the request results in a fault and the cache is full. The objective is to design online algorithms that minimize the total number of faults. Three well known paging algorithms are Least-Recently-Used (LRU), First-In-First-Out (FIFO), and Flush-When-Full (FWF) [10]. All these paging algorithms have competitive ratio k, which is the best among all deterministic online paging algorithms [10]. On the other hand, experimental studies show that LRU has a performance ratio at most four times the optimal offline [37]. Furthermore, it has been empirically well established that LRU (and/or variants thereof) are, in practice, preferable paging strategies to all other known paging algorithms [33].

Such anomalies have led to the introduction of many alternatives to competitive analysis of online algorithms (see [19] for a comprehensive survey). Some examples are *loose competitiveness* [37,39], *diffuse adversary* [28,38], the *Max/Max ratio* [9], the *relative worst order ratio* [14], and the *random order ratio* [27]. None of them fully resolve all the known issues with competitive analysis.

It is well known that input sequences for paging and several other problems show locality of reference. This means that when a page is requested it is more likely to be requested in the near future. Therefore several models for paging with locality of reference have been proposed. In the early days of computing, Denning recognized the locality of reference principle and modeled it using the well known *working set* model [16,17]. He defined the working set of a process as the set of most recently used pages and addressed thrashing using this model. After the introduction of the working set model, the locality principle has been adopted in operating systems, databases, hardware architectures, compilers, and many other areas. Therefore it holds even more so today. Indeed, [18] states "locality of reference is one of the cornerstones of computer science."

One apparent reason for the drawbacks of competitive analysis of paging is that it does not incorporate the concept of locality of reference. Several models incorporating locality have been proposed. The *access graph* model by Borodin et al. [11,25,15,21] and its generalization by Karlin et al. [26] model the request sequences as a graph, possibly weighted by probabilistic transitions. Becchetti [8] refined the diffuse adversary model of Koutsoupias and Papadimitriou by considering only probabilistic distributions in which locality of reference is present. Albers et al. [2] introduced a model in which input sequences are classified according to a measure of locality of reference.

Recently, Angelopoulos et al. introduced *Bijective Analysis* and *Average Analysis* [4] which combined with the locality model of Albers et al. [2], shows that LRU is the sole optimal paging algorithm on sequences with locality of reference. This resolved an important disparity between theory and practice of online paging algorithms, namely the superiority in practice of LRU. An analogous result for list update and MTF is shown in [5] and the separation of LRU was strengthened by Angelopoulos and Schweitzer in [6]. These last separation results are based on heavy machinery specifically designed to resolve this singular long-standing question and leave open the question of how to efficiently characterize the full spectrum of performance of the various known paging and list update algorithms. In contrast, the new measure we propose is easier to apply and creates a performance hierarchy of paging and list update algorithms which better reflects their intuitive relative strengths. Several previously observed experimental properties can be readily proven using the new model. This is a strength of the new model in that it is effective, is readily applicable to a variety of algorithms, and provides meaningful results.

Paging and list update are the best testbeds for developing alternative measures, given our extensive understanding of these problems. We know why competitive analysis fails, what are typical sequences in practice and we can better evaluate whether a new technique indeed overcomes known shortcomings. It is important to note that even though well studied, most of the alternative models for these problems are only partially successful in resolving the issues posed by them and as such these problems are still challenging case studies against which to test a new model.

In this paper we apply parameterized analysis and analyze the performance of well known paging and list update algorithms in terms of a measure of locality of reference. This measure is related to Denning's working set model [16], the locality of reference model of [2], and the working set theorem in the context of the splay trees and other self-organizing data structures [35,23,24,12]. For paging, this leads to better separation than the competitive ratio. Furthermore, in contrast to competitive analysis it reflects the intuition that a larger cache leads to better performance. We also provide experimental results that justify the applicability of our measure in practice. For list update, we show that this new model produces the finest separation yet of list update algorithms. We obtain bounds on the parameterized performance of several list update algorithms and prove the superiority of MTF. We also apply our measures to randomized list update algorithms and show that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the parameterized case. Some of the proofs follow the general outline of standard competitive analysis proofs (e.g., those in [10]), yet in some cases provide finer separation of paging and list update algorithms.

2 Parameterized Analysis of Paging Algorithms

Recall that on a fault (with a full cache), LRU evicts the page that is least recently requested, FIFO evicts the page that is first brought to the cache, FWF empties the cache, Last-In-First-Out (LIFO) evicts the page that is most recently brought to the cache, and Least-Frequently-Used (LFU) evicts the page that has been requested the least since entering the cache. LFU and LIFO do not have a constant competitive ratio [10]. A paging algorithm is called *conservative* if it incurs at most k faults on any consecutive subsequence that contains at most k distinct pages. A paging algorithm is said to be a *marking algorithm* if it never evicts a marked page, where a page is marked if accessed and all pages are unmarked at the end of each phase. LRU and FIFO are conservative algorithms, while LRU and FWF are marking algorithms.

As stated before input sequences for paging show locality of reference in practice. We want to express the performance of paging algorithms on a sequence in terms of the amount of the locality in that sequence. Therefore we need a measure that assigns a number proportional to the amount of locality in each sequence. None of the previously described models provide a unique numerical value as a measure of locality of reference. We define a quantitative measure for non-locality of paging instances.

Definition 1. *For a sequence σ we define $d_\sigma[i]$ as either $k+1$ if this is the first request to page $\sigma[i]$, or otherwise, the number of distinct pages that are requested since the last request to $\sigma[i]$ (including $\sigma[i]$).*[1] *Now we define $\overline{\lambda}(\sigma)$, the "non-locality" of σ, as $\overline{\lambda}(\sigma) = \frac{1}{|\sigma|}\sum_{1\leq i\leq|\sigma|} d_\sigma[i]$. We denote the non-locality by $\overline{\lambda}$ if the choice of σ is clear from the context.*

If σ has high locality of reference, the number $d_\sigma[i]$ of distinct pages between two consecutive requests to a page is small for most values of i and thus σ has a low non-locality. Note that while this measure is related to the working set model [16] and the locality model of [2], it differs from both in several aspects. Albers et al. [2] consider the maximum/average number of distinct pages in all windows of the same size, while we consider the number of distinct pages requested since the last access to each page. Also our analysis does not depend on a concave function f whose identification for a particular application might not be straightforward. Our measure is also closely related to the working set theorem in area of self-organizing data structures [35]. For binary search trees (like splay trees), the working set bound is defined as $\sum_{1\leq i\leq|\sigma|}\log(d_\sigma[i]+1)$. The logarithm can be explained by the logarithmic bounds on most operations in binary search trees. Thus our measure of locality of reference can be considered as variant of this measure in which we remove the logarithm.

2.1 Experimental Evaluation of the Measure

In order to check validity of our measure we ran some experiments on traces of memory reference streams from the NMSU TraceBase [36]. Here we present the results of our experiments on address traces collected from SPARC processors running the SPEC92 benchmarks. We considered a page size of 2048 bytes and truncated them after 40000 references. The important thing to notice is that these are not special cases or artificially generated memory references, but are access patterns which a real-life implementation of any paging algorithm might face. The results for the corresponding eleven program traces are shown in

[1] Asymptotically, and assuming the number of requests is much larger than the number of distinct pages, any constant can replace $k+1$ for the $d_\sigma[i]$ of the first accesses.

Table 1. Locality of address traces collected from SPARC processors running the SPEC92 benchmarks

	espresso	li	eqntott	compress	tomcatv	ear	sc	swm	gcc
Distinct	3913	3524	9	189	5260	1614	561	3635	2663
$\overline{\lambda}$	193.1	195.2	1.7	2.3	348.3	34.1	5.4	166.7	90.6
Ratio	4.9%	5.5%	19.3%	1.2%	6.6%	2.1%	1.0%	4.6%	3.4%

Table 1. The first row shows the number of distinct pages, the second row shows $\overline{\lambda}$, and finally the third row shows the ratio of the actual locality to the worst possible locality. The worst possible locality of a trace asymptotically equals the number of distinct pages in that trace. It is clear from the low ratios that in general these traces exhibit high locality of reference as defined by our measure.

2.2 Theoretical Results

Next we analyze several well known paging algorithms in terms of the non-locality parameter. We consider the fault rate, the measure usually used by practitioners. The fault rate of a paging algorithm $\mathcal{A}$ on a sequence σ is defined as $\mathcal{A}(\sigma)/|\sigma|$, i.e., the number of faults $\mathcal{A}$ incurs on σ normalized by the length of σ. The fault rate of $\mathcal{A}$, $FR(A)$, is defined as the asymptotic worst case fault rate of $\mathcal{A}$ on any sequence. The bounds are in the worst case sense, i.e., when we say $FR(\mathcal{A}) \geq f(\overline{\lambda})$ we mean that there is a sequence σ such that $\frac{\mathcal{A}(\sigma)}{|\sigma|} \geq f(\overline{\lambda}(\sigma))$ and when we say $FR(\mathcal{A}) \leq g(\overline{\lambda})$ we mean that for every sequence σ we have $\frac{\mathcal{A}(\sigma)}{|\sigma|} \leq g(\overline{\lambda}(\sigma))$. Observe that $0 \leq FR(\mathcal{A}) \leq 1$. Also for simplicity, we ignore the details related to the special case of the first few requests (the first block or phase). Asymptotically and as the size of the sequences grow, this can only change the computation by additive lower order terms. Proof of some lemmas have been omitted due to space constraints but can be found in the full version of the paper.

Lemma 1. *For any deterministic paging algorithm $\mathcal{A}$, $\frac{\overline{\lambda}}{k+1} \leq FR(\mathcal{A}) \leq \frac{\overline{\lambda}}{2}$.*

Proof. For the lower bound consider a slow memory containing $k+1$ pages. Let σ be a sequence of length n obtained by first requesting $p_1, p_2, \ldots, p_k, p_{k+1}$, and afterwards repeatedly requesting the page not currently in $\mathcal{A}$'s cache. Since $\frac{\mathcal{A}(\sigma)}{|\sigma|} = n/n = 1$, and $\overline{\lambda}$ is at most $k+1$ (there are $k+1$ distinct pages in σ), the lower bound follows.

For the upper bound, consider any request sequence σ of length n. If the i^{th} request is a fault charged to $\mathcal{A}$, then $d_\sigma[i] \geq 2$ (otherwise $\sigma[i]$ cannot have been evicted). Hence, $2\mathcal{A}(\sigma) \leq \sum_{i=1}^{n} d_\sigma[i]$ and the upper bound follows. □

We now show that LRU has the best possible performance in terms of $\overline{\lambda}$.

Theorem 1. $FR(\text{LRU}) = \frac{\overline{\lambda}}{k+1}$.

Proof. It follows from the definition of LRU that it faults on the i^{th} request if and only if $d_\sigma[i] \geq k+1$, which implies $\text{LRU}(\sigma) \leq \frac{\overline{\lambda}}{k+1}$. The lower bound follows directly from Lemma 1. □

Next, we show a general upper bound for conservative and marking algorithms.

Lemma 2. *Let $\mathcal{A}$ be a conservative or marking algorithm, then $FR(\mathcal{A}) \leq \frac{2\overline{\lambda}}{k+3}$.*

Proof. Let σ be an arbitrary sequence and let φ be an arbitrary phase in the decomposition of σ. $\mathcal{A}$ incurs at most k faults on φ. For any phase except the first, the first request in φ, say $\sigma[i]$, is to a page that was not requested in the previous phase, which contained k distinct pages. Hence, $d_\sigma[i] \geq k+1$. There are at least $k-1$ other requests in φ to $k-1$ distinct pages, which all could have been present in the previous phase. But these pages contribute at least $\sum_{j=1}^{k-1}(j+1) = k - 1 + \frac{k^2-k}{2}$ to $\overline{\lambda}$. It follows that the contribution of this phase to $|\sigma|\overline{\lambda}$ is at least $k+1+k-1+\frac{k^2-k}{2} = \frac{k^2+3k}{2}$. Hence,

$$\frac{\mathcal{A}(\sigma)}{|\sigma|\overline{\lambda}} \leq \frac{k}{\frac{k^2+3k}{2}} = \frac{2}{k+3} \Rightarrow FR(\mathcal{A}) \leq \frac{2\overline{\lambda}}{k+3}.$$

□

There is a matching lower bound for FWF.

Lemma 3. $FR(\text{FWF}) = \frac{2\overline{\lambda}}{k+3}$.

Thus FWF has approximately twice as many faults as LRU on sequences with the same locality of reference, in the worst case. FIFO also has optimal performance in terms of $\overline{\lambda}$.

Lemma 4. $FR(\text{FIFO}) = \frac{\overline{\lambda}}{k+1}$.

Lemma 5. $FR(\text{LFU}) \geq \frac{2\overline{\lambda}}{k+3}$.

In contrast LIFO has much poorer performance than most other paging algorithms (the worst possible) in terms of $\overline{\lambda}$.

Lemma 6. $FR(\text{LIFO}) = \frac{\overline{\lambda}}{2}$.

LRU-2 is another paging algorithm proposed by O'Neil et al. for database disk buffering [29]. On a fault, LRU-2 evicts the page whose second to the last request is least recent. If there are pages in the cache that have been requested only once so far, LRU-2 evicts the least recently used among them. Boyar et al. proved that LRU-2 has competitive ratio $2k$, which is worse than FWF [13].

Lemma 7. $\frac{2k\overline{\lambda}}{(k+1)(k+2)} \leq FR(\text{LRU-2}) \leq \frac{2\overline{\lambda}}{k+1}$.

While no deterministic on-line paging algorithm can have competitive ratio better than k, there are randomized algorithms with better competitive ratio. The randomized marking algorithm MARK, introduced by Fiat et al. [20], is $2H_k$-competitive, where H_k is the k^{th} harmonic number. On a fault, MARK evicts a page chosen uniformly at random from among the unmarked pages. Let σ be a sequence and $\varphi_1, \varphi_2, \ldots, \varphi_m$ be its phase decomposition. A page requested in phase φ_i is called *clean* if it was not requested in phase φ_{i-1} and *stale* otherwise. Let c_i be the number of clean pages requested in phase φ_i. Fiat et al. proved that the expected number of faults MARK incurs on phase φ_i is $c_i(H_k - H_{c_i} + 1)$.

Lemma 8. $FR(\text{MARK}) = \frac{2\overline{\lambda}}{3k+1}$.

Proof. Let σ be $\{p_1p_2 \ldots p_kp_{k+1}p_{k+2} \ldots p_{2k}p_kp_{k-1} \ldots p_1p_{2k} \ldots p_{k+1}\}^n$. This sequence has $4n$ phases. All pages of each phase are clean. Therefore we have $c_i = k$ for $1 \le i \le 4n$ and the expected number of faults MARK incurs on each phase is $k \times (H_k - H_k + 1) = k$. Thus $E(\text{MARK}(\sigma)) = 4nk$. We have $|\sigma|\overline{\lambda} = 4n(k+1+k+2+\cdots+2k) = 4n(k^2 + k(k+1)/2) = 2n(3k^2+k)$. Hence

$$\frac{E(\text{MARK}(\sigma))}{|\sigma|\overline{\lambda}} = \frac{4nk}{2n(3k^2+k)} = \frac{2}{3k+1},$$

which proves the lower bound. For the upper bound, consider an arbitrary sequence σ and let $\varphi_1, \varphi_2, \ldots, \varphi_m$ be its phase decomposition. Suppose that the i^{th} phase φ_i has c_i clean pages. Therefore the expected cost of MARK on phase i is at most $c_i(H_k - H_{c_i} + 1)$. The first request to the j^{th} clean page in a phase contributes at least $k + j$ to $|\sigma|\overline{\lambda}$ (k pages from previous phase and $j-1$ clean pages that have been seen so far). The first request to the j^{th} stale page in a phase contributes at least $j+1$. Therefore the contribution of phase i to $|\sigma|\overline{\lambda}$ is at least $\sum_{j=1}^{c_i}(k+j) + \sum_{j=1}^{k-c_i}(j+1) = (2c_i^2 - 2c_i + k^2 + 3k)/2$, and

$$\frac{E(\text{MARK}(\sigma))}{|\sigma|\overline{\lambda}} \le \frac{2c_i(H_k - H_{c_i} + 1)}{2c_i^2 - 2c_i + k^2 + 3k},$$

where $1 \le c_i \le k$. This is an increasing function in terms of c_i and attains its maximum at $c_i = k$. Then we have

$$\frac{E(\text{MARK}(\sigma))}{|\sigma|\overline{\lambda}} \le \frac{2k(H_k - H_k + 1)}{2k^2 - 2k + k^2 + 3k} = \frac{2}{3k+1}.$$

□

Finally we bound the performance of LONGEST-FORWARD-DISTANCE (LFD), an optimal offline algorithm. On a fault, LFD evicts the page whose next request is farthest in the future.

Lemma 9. $\frac{\overline{\lambda}}{3k+1} \le FR(\text{LFD}) \le \frac{2\overline{\lambda}}{3k+1}$.

The results are summarized in Table 2. According to these results, LRU and FIFO have optimal performance among deterministic algorithms. Marking algorithms can be twice as bad and FWF is among the worst marking algorithms.

Table 2. The results for paging

Algorithm	Lower Bound	Upper Bound
Deterministic	$\frac{\overline{\lambda}}{k+1}$	$\frac{\overline{\lambda}}{2}$
LRU	$\frac{\overline{\lambda}}{k+1}$	$\frac{\overline{\lambda}}{k+1}$
Marking	$\frac{\overline{\lambda}}{k+1}$	$\frac{2\overline{\lambda}}{k+3}$
FWF	$\frac{2\overline{\lambda}}{k+3}$	$\frac{2\overline{\lambda}}{k+3}$
FIFO	$\frac{\overline{\lambda}}{k+1}$	$\frac{\overline{\lambda}}{k+1}$
LFU	$\frac{2\overline{\lambda}}{k+3}$	$\frac{\overline{\lambda}}{2}$
LIFO	$\frac{\overline{\lambda}}{2}$	$\frac{\overline{\lambda}}{2}$
LRU-2	$\frac{2k\overline{\lambda}}{(k+1)(k+2)}$	$\frac{2\overline{\lambda}}{k+1}$
MARK	$\frac{2\overline{\lambda}}{3k+1}$	$\frac{2\overline{\lambda}}{3k+1}$
LFD	$\frac{\overline{\lambda}}{3k+1}$	$\frac{2\overline{\lambda}}{3k+1}$

Table 3. The results for list update

Algorithm	Lower Bound	Upper Bound
General	$\widehat{\lambda}$	$m \cdot \widehat{\lambda}$
MTF	$\widehat{\lambda}$	$\widehat{\lambda}$
Transpose	$\frac{m \cdot \widehat{\lambda}}{2}$	$m \cdot \widehat{\lambda}$
FC	$\approx \frac{m \cdot \widehat{\lambda}}{2}$	$m \cdot \widehat{\lambda}$
TS	$\approx 2\widehat{\lambda}$	$m \cdot \widehat{\lambda}$
Bit	$\approx \frac{3}{2}\widehat{\lambda}$	$m \cdot \widehat{\lambda}$

LIFO has the worst performance possible and LRU-2 is almost twice as bad as LRU. The performance of the randomized algorithm MARK is better than any deterministic algorithms: it behaves almost 2/3 better than LRU. LFD, an optimal offline algorithm, performs almost three times better than LRU, which coincides with the observed result that the competitive ratio of FIFO and LRU in practice is a small constant independent of the cache size [10].

3 Parameterized Analysis of List Update Algorithms

In this section we study the parameterized complexity of list update algorithms in terms of locality of reference. In the list update problem, we have an unsorted list of m items. The input is a sequence of n requests that should be served in an online manner. Let $\mathcal{A}$ be an arbitrary online list update algorithm. To serve a request to an item x, $\mathcal{A}$ should linearly search the list until it finds x. If x is i^{th} item in the list, $\mathcal{A}$ incurs cost i to access x. Immediately after accessing x, $\mathcal{A}$ can move x to any position closer to the front of the list at no extra cost. This is called a *free exchange*. Also $\mathcal{A}$ can exchange any two consecutive items at a cost of 1. These are called *paid exchanges*. The idea is to use free and paid exchanges to minimize the overall cost of serving a sequence. Three well known deterministic online algorithms are *Move-To-Front* (MTF), *Transpose*, and *Frequency-Count* (FC). MTF moves the requested item to the front of the list and TRANSPOSE exchanges the requested item with the item that immediately precedes it. FC maintains a frequency count for each item, updates this count after each access, and makes necessary moves so that the list always contains items in non-increasing order of frequency count. Sleator and Tarjan showed that MTF is 2-competitive, while TRANSPOSE and FC do not have constant competitive ratios [34]. While list update algorithms can be more easily distinguished using competitive analysis than in the paging case, the experimental study by Bachrach and El-Yaniv suggests that the relative performance hierarchy as computed by the competitive ratio does not correspond to the observed relative performance of the algorithms in practice [7]. Several authors have pointed out that input sequences of list update algorithms in practice show locality of reference [22,32,10] and indeed online list update algorithms try to take advantage of this property [22,31]. Recently, Angelopoulos et al. [5] and Albers and Lauer [3] have studied list update with locality of reference. We define the non-locality of sequences for list update in an analogous way to the corresponding definition for paging (Definition 1) . The only differences are:

1. We do not normalize the non-locality by the length of the sequence, i.e., $\widehat{\lambda}(\sigma) = \sum_{1 \leq i \leq |\sigma|} d_\sigma[i]$.
2. If $\sigma[i]$ is the first access to an item we assign the value m to $d_\sigma[i]$[2].

Next, we prove bounds on the performance of well known list update algorithms. Again, proof of some lemmas have been omitted due to space constraints but can be bound in the full version of the paper.

Theorem 2. *For any deterministic list update algorithm $\mathcal{A}$, $\widehat{\lambda} \leq \mathcal{A}(\sigma) \leq m \cdot \widehat{\lambda}$.*

We now show that MTF is optimal in terms of $\widehat{\lambda}$.

Theorem 3. MTF$(\sigma) = \widehat{\lambda}$.

Proof. Consider the i^{th} request of σ. If this is the first request to item $\sigma[i]$, then $d_\sigma[i] = m$, while the cost of MTF on $\sigma[i]$ is at most m. Otherwise, the cost of MTF is $d_\sigma[i]$. Thus the cost of MTF on $\sigma[i]$ is at most $d_\sigma[i]$. Hence, MTF$(\sigma) \leq \sum_{1 \leq i \leq n} d_\sigma[i] = \widehat{\lambda}$, and the upper bound follows. Theorem 2 shows that this bound is tight. □

Lemma 10. Transpose$(\sigma) \geq \frac{m \cdot \widehat{\lambda}}{2}$.

Lemma 11. FC$(\sigma) \geq \frac{(m+1)\widehat{\lambda}}{2} \approx \frac{m \cdot \widehat{\lambda}}{2}$.

Albers introduced the algorithm *Timestamp* (TS) and showed that it has competitive ratio 2 [1]. After accessing an item a, TS inserts a in front of the first item b that is before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorganize the list.

Lemma 12. TS$(\sigma) \geq \frac{2m \cdot \widehat{\lambda}}{m+1} \approx 2\widehat{\lambda}$.

Observe that parameterized analysis by virtue of its finer partition of the input space resulted in the separation of several of these strategies which are not separable under the classical model. This introduces a hierarchy of algorithms better reflecting the relative strengths of the strategies considered above. We

[2] As for paging, asymptotically, and assuming the number of requests is much larger than m, any constant can replace m for the $d_\sigma[i]$ of the first accesses.

can also apply the parameterized analysis to randomized list update algorithms by considering their expected cost.

In the next theorem we show that, surprisingly, certain randomized algorithms which are superior to MTF in the standard model are not so in the parameterized case. Observe that in the competitive ratio model a deterministic algorithm must serve a pathological, rare worst case even if at the expense of a more common but not critical case, while a randomized algorithm can hedge between the two cases, hence in the classical model the randomized algorithm is superior to the deterministic one. In contrast, in the parameterized model the rare worst case has a larger non-locality measure if it is pathological, leading to a larger denominator. Hence such a cases can safely be ignored, with a resulting overall increase in the measured quality of the algorithm.

The algorithm *Bit*, considers a bit $b(a)$ for each item a and initializes these bits uniformly and independently at random. Upon an access to a, it first complement $b(a)$, then if $b(a) = 0$ it moves a to the front, otherwise it does nothing. Bit has competitive ratio 1.75, thus beating any deterministic algorithm [30]. In the parameterized model this situation is reversed.

Theorem 4. *$E(\textsc{Bit}(\sigma)) \geq \frac{(3m+1)\widehat{\lambda}}{2m+2} \approx 3\widehat{\lambda}/2$.*

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$ be the initial list and n be an arbitrary integer. Consider the sequence $\sigma = \{a_m^2 a_{m-1}^2 \ldots a_1^2\}^n$. Let σ_i and σ_{i+1} be two consecutive accesses to a_j. After two consecutive accesses to each item, it will be moved to the front of the list with probability 1. Therefore a_j is in the last position of the list maintained by Bit at the time of request σ_i and Bit incurs cost m on this request. After this request, Bit moves a_j to the front of the list if and only if $b(a_j)$ is initialized to 1. Since $b(a_j)$ is initialized uniformly and independently at random, this will happen with probability 1/2. Therefore the expected cost of Bit on σ_{i+1} is $\frac{1}{2}(m+1)$ and the expected cost of Bit on σ is $nm(m + \frac{m+1}{2})$. We have $\widehat{\lambda} = m(m+1)n$. Therefore

$$\frac{E(\textsc{Bit}(\sigma))}{\widehat{\lambda}} = \frac{n \cdot m(m + \frac{m+1}{2})}{m(m+1)n} = \frac{3m+1}{2m+2}.$$

□

The results are summarized in Table 3. According to these results, MTF has the best performance among well known list update algorithms. TS has performance at least twice as bad as MTF. The performance of Transpose and FC is at least $m/2$ times worse than MTF. The performance of Bit is worse than MTF, while its competitive ratio is better. Experimental results of [7] show that MTF has better performance than Bit in practice, which favors our result over competitive analysis.

4 Conclusions

We applied parameterized analysis in terms of locality of reference to paging and list update algorithms and showed that this model gives promising results.

The plurality of results shows that this model is effective in that we can readily analyze well known strategies. Using a finer, more natural measure we separated paging and list update algorithms which were otherwise indistinguishable under the classical model. We showed that a randomized algorithm which is superior to MTF in the classical model is not so in the cooperative case, which matches experimental evidence. This confirms that the ability of the online adaptive algorithm to ignore pathological worst cases can lead to the selection of algorithms that are more efficient in practice.

References

1. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM Journal on Computing 27(3), 682–693 (1998)
2. Albers, S., Favrholdt, L.M., Giel, O.: On paging with locality of reference. JCSS 70(2), 145–175 (2005)
3. Albers, S., Lauer, S.: On list update with locality of reference. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 96–107. Springer, Heidelberg (2008)
4. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: Proc. SODA, pp. 229–237 (2007)
5. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: List update with locality of reference. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 399–410. Springer, Heidelberg (2008)
6. Angelopoulos, S., Schweitzer, P.: Paging and list update under bijective analysis. In: Proc. SODA, pp. 1136–1145 (2009)
7. Bachrach, R., El-Yaniv, R.: Online list accessing algorithms and their applications: Recent empirical evidence. In: Proc. SODA, pp. 53–62 (1997)
8. Becchetti, L.: Modeling locality: A probabilistic analysis of LRU and FWF. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 98–109. Springer, Heidelberg (2004)
9. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica 11, 73–91 (1994)
10. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
11. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. JCSS 50, 244–258 (1995)
12. Bose, P., Douïeb, K., Langerman, S.: Dynamic optimality for skip lists and B-trees. In: Proc. SODA, pp. 1106–1114 (2008)
13. Boyar, J., Ehmsen, M.R., Larsen, K.S.: Theoretical evidence for the superiority of LRU-2 over LRU for the paging problem. In: Erlebach, T., Kaklamanis, C. (eds.) WAOA 2006. LNCS, vol. 4368, pp. 95–107. Springer, Heidelberg (2007)
14. Boyar, J., Favrholdt, L.M.: The relative worst order ratio for on-line algorithms. In: Proc. Italian Conf. on Algorithms and Complexity (2003)
15. Chrobak, M., Noga, J.: LRU is better than FIFO. Algorithmica 23(2), 180–185 (1999)
16. Denning, P.J.: The working set model for program behaviour. CACM 11(5), 323–333 (1968)

17. Denning, P.J.: Working sets past and present. IEEE Transactions on Software Engineering SE-6(1), 64–84 (1980)
18. Denning, P.J.: The locality principle. CACM 48(7), 19–24 (2005)
19. Dorrigiv, R., López-Ortiz, A.: A survey of performance measures for on-line algorithms. SIGACT News 36(3), 67–81 (2005)
20. Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive paging algorithms. Journal of Algorithms 12, 685–699 (1991)
21. Fiat, A., Rosen, Z.: Experimental studies of access graph based heuristics: Beating the LRU standard? In: Proc. SODA, pp. 63–72 (1997)
22. Hester, J.H., Hirschberg, D.S.: Self-organizing linear search. ACM Computing Surveys 17(3), 295 (1985)
23. Iacono, J.: Improved upper bounds for pairing heaps. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 32–45. Springer, Heidelberg (2000)
24. Iacono, J.: Alternatives to splay trees with O(log n) worst-case access times. In: Proc. SODA, pp. 516–522 (2001)
25. Irani, S., Karlin, A.R., Phillips, S.: Strongly competitive algorithms for paging with locality of reference. SIAM Journal on Computing 25, 477–497 (1996)
26. Karlin, A.R., Phillips, S.J., Raghavan, P.: Markov paging. SIAM Journal on Computing 30(3), 906–922 (2000)
27. Kenyon, C.: Best-fit bin-packing with random order. In: Proc. SODA, pp. 359–364 (1996)
28. Koutsoupias, E., Papadimitriou, C.: Beyond competitive analysis. SIAM Journal on Computing 30, 300–317 (2000)
29. O'Neil, E.J., O'Neil, P.E., Weikum, G.: The LRU-K page replacement algorithm for database disk buffering. In: Proc. ACM SIGMOD Conf., pp. 297–306 (1993)
30. Reingold, N., Westbrook, J.: Randomized algorithms for the list update problem. Technical Report YALEU/DCS/TR-804, Yale University (June 1990)
31. Reingold, N., Westbrook, J., Sleator, D.D.: Randomized competitive algorithms for the list update problem. Algorithmica 11, 15–32 (1994)
32. Schulz, F.: Two new families of list update algorithms. In: Chwa, K.-Y., Ibarra, O.H. (eds.) ISAAC 1998. LNCS, vol. 1533, pp. 99–108. Springer, Heidelberg (1998)
33. Silberschatz, A., Galvin, P.B., Gagne, G.: Operating System Concepts. John Wiley & Sons, Chichester (2002)
34. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. CACM 28, 202–208 (1985)
35. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. JACM 32(3), 652–686 (1985)
36. N.M.S. University: Homepage of new mexico state university tracebase (online), http://tracebase.nmsu.edu/tracebase.html
37. Young, N.E.: The k-server dual and loose competitiveness for paging. Algorithmica 11(6), 525–541 (1994)
38. Young, N.E.: On-line paging against adversarially biased random inputs. Journal of Algorithms 37(1), 218–235 (2000)
39. Young, N.E.: Online file caching. Algorithmica 33(3), 371–383 (2002)

Online Scheduling of Bounded Length Jobs to Maximize Throughput

Christoph Dürr[1], Łukasz Jeż[2], and Kim Thang Nguyen[1]

[1] CNRS, LIX UMR 7161, Ecole Polytechnique, 91128 Palaiseau, France[*]
[2] Institute of Computer Science, University of Wrocław, 50-383 Wrocław, Poland[**]

Abstract. We consider an online scheduling problem, motivated by the issues present at the joints of networks using ATM and TCP/IP. Namely, IP packets have to broken down to small ATM cells and sent out before their deadlines, but cells corresponding to different packets can be interwoven. More formally, we consider the online scheduling problem with preemptions, where each job j is revealed at release time r_j, has processing time p_j, deadline d_j and weight w_j. A preempted job can be resumed at any time. The goal is to maximize the total weight of all jobs completed on time. Our main results are as follows: we prove that if all jobs have processing time *exactly* k, the deterministic competitive ratio is between 2.598 and 5, and when the processing times are *at most* k, the deterministic competitive ratio is $\Theta(k/\log k)$.

1 Introduction

Many Internet service providers use an ATM network which has been designed to send telephone communication and television broadcasts, as well as usual network data. However, the Internet happens to use TCP/IP, so at the joints of these networks IP packets have to be broken down into small ATM cells and fed into the ATM network. This raises many interesting questions, as ATM network works with fixed sized cells (48 bytes), while IP network works with variable sized packets. In general, packet sizes are bounded by the capacity of Ethernet, i.e. 1500 bytes, and in many cases they actually achieve this maximal length. Ideally packets also have deadlines and priorities (weights). The goal is to maximise the *quality of service*, i.e. the total weight of packets that have been entirely sent out on time.

This problem can be formulated as an online-scheduling problem on a single machine, where jobs arrive online at their release times, have some processing times, deadlines and weights, and the objective is to maximise the total weight of jobs completed on time. Preemption is allowed, so a job i can be scheduled in several separated time intervals, as long as their lengths add up to p_i. Time is divided into integer time steps, corresponding to the transmission time of an ATM cell, and all release times, deadlines and processing times are assumed

[*] Supported by ANR Alpage.
[**] Supported by MNiSW N N206 1723 33, 2007–2010 and COST 295 "DYNAMO".

to be integer. This problem can be denoted as $1|\text{online-}r_i;\text{pmtn}|\sum w_i(1-U_i)$, according to the notation of [6].

1.1 Our Results

In this paper we consider the case when processing times of all jobs are bounded by some constant k, and the case when they equal k. Both variants are motivated by the network application in mind. We study the competitive ratio as a function of k. Our main results are as follows.

– We provide an optimal online algorithm for the bounded processing time case that reaches the ratio $O(k/\log k)$.
– We provide a 5-competitive algorithm for the equal processing time case, and a lower bound of $3\sqrt{3}/2 \approx 2.598$ on the competitive ratio of any determistic algorithm for that case. The lower bound employs a previously known input sequence, for which a bound of 2.59 was claimed with an incomplete proof [5] (see discussion at the end of section 5).

We also provide several minor results, which are are only sketched due to space constraints.

– For the bounded processing time case, we show that the competitive ratio of a well-known Smith Ratio Algorithm is between k and $2k$. We also show that asymptotically the competitive ratio of any deterministic algorithm is at least $k/\ln k$, improving the previous bound [13] by a factor of 2.
– For bounded processing time with unit weights, it is known that the competitive ratio is $\Omega(\log k/\log\log k)$ when time points are allowed to be rationals [3]. We provide an alternative proof for the more restricted integer variant, obtaining better multiplicative constant at the same time.
– $O(\log k)$-competitiveness of Shortest Remaining Processing Time First for the bounded processing time, unit weight model follows as a byproduct from an involved proof of [11]. We provide an alternative concise proof of its $2H_k$-competitiveness and note that this is tight up to a constant factor.

1.2 Related Work

It is known that the general problem without a bound on processing times has an unbounded deterministic competitive ratio [3], so different directions of research were considered. Two related approaches are to consider resource augmentation and randomisation. For the former an online algorithm that has constant competitive ratio provided it is allowed a constant speed-up of its machine compared to the adversary is known [10], and for the latter a constant competitive randomised algorithm is known [11] Finally, a third direction is to restrict to instances with bounded processing time.

Bounded processing time, unit weights. ($\forall j\ p_j \leq k,\ w_j = 1$) The offline problem can be solved in time $O(n^4)$ [1] already when the processing time is unbounded. Baruah et al. [3] showed that any deterministic online algorithm is $\Omega(\log k/\log\log k)$-competitive in a model where processing times, release times and deadlines of jobs can be rational. The currently best known algorithm is Shortest Remaining Processing Time First, which is $O(\log k)$-competitive [11]. The same paper provides a constant competitive randomized algorithm, however with a large constant.

Bounded processing time, arbitrary weights. $(\forall j\ p_j \le k)$ For fixed k the offline problem has not been studied to our knowledge, and when the processing times are unbounded the offline problem is $\mathcal{NP}$-hard by a trivial reduction from Knapsack Problem. It is known that any deterministic online algorithm for this case has competitive ratio $k/(2\ln k)-1$ [13]. For the variant with tight jobs only, i.e. jobs that satisfy $d_j = r_j + p_j$, Canetti and Irani [4] provide an $O(\log k)$-competitive randomised online algorithm and show a $\Omega(\sqrt{\log k/\log\log k})$ lower bound for any randomized competitive algorithm against an oblivious adversary.

Equal processing time, unit weights. $(\forall j\ p_j = k,\ w_j = 1)$ The offline problem can be solved in time $O(n\log n)$ [12], and the same algorithm can be turned into a 1-competitive online algorithm, see for example [14].

Equal processing time, arbitrary weights. $(\forall j\ p_j = k)$ The offline problem can be solved in time $O(n^4)$ [2]. For $k = 1$ the problem is well studied, and the deterministic competitive ratio is between 1.618 and 1.83 [9, 8].

Our model is sometimes called the *preemptive model with resume*, as opposed to *preemptive model with restarts* [7], in which an interrupted job can only be processed from the very beginning. *Overloaded real-time systems* [3] form another related model, in which all the job parameters are reals, the time is continuous, and uniform weights are assumed.

2 Preliminaries

For a job i we denote its release time by r_i, its deadline by d_i, its processing time by p_i and its weight by w_i. All these quantities, except w_i, are integers. Let $q_i(t)$ be the remaining processing time of job i for the algorithm at time t. When there is no confusion, we simply write q_i. We say that job i is *pending* for the algorithm at time t if it has not been completed yet, $r_i \le t$, and $t + q_i(t) < d_i$. We say a job j is *tight* at time t if $t + q_j(t) = d_j$. For a job j uncompleted by the algorithm, the *critical time* of j is the latest time when j was still pending for the algorithm. In other words, the critical time s of job j for the algorithm is such moment s that if the algorithm does not schedule j at time s, it cannot finish j anymore, i.e. $s = \max\{\tau : \tau + q_j(\tau) = d_j\}$. We assume that a unit (i, a) scheduled at time t is processed during the time interval $[t, t + 1)$, i.e. its processing is finished just before time $t + 1$. For this reason by *completion time* of a job i we mean $t + 1$ rather than t, where t is the time its last unit was scheduled.

Throughout the paper we analyse many algorithms with similar charging schemes sharing the following outline: for every job j completed by the adversary we consider its p_j units. Each unit of job j charges w_j/p_j to some job i_0 completed by the algorithm. The charging schemes satisfy the property that every job i_0 completed by the algorithm receives a total charge of at most Rw_{i_0}, which implies R-competitiveness of the algorithm.

More precisely we distinguish individual units scheduled by both the algorithm and the adversary, where unit (i, a) stands for execution of job i when its remaining processing time was a. In particular a complete job i consists of the units $(i, p_i), (i, p_i - 1), \dots, (i, 1)$. With every algorithm's unit (i, a) we associate a *capacity* $\pi(i, a)$ that depends on w_i and a, whose exact value will be different from proof to proof. The algorithms, with their capacities, will be designed in such a way that they satisfy the following properties, with respect to π.

ρ-monotonicity: If the algorithm schedules (i, a) with $a > 1$ at t and (i', a') at $t + 1$, then $\rho\pi(i', a') \geq \pi(i, a)$,

validity: If a job j is pending for the algorithm at any time t, then the algorithm schedules a unit (i, a) at t such that $\pi(i, a) \geq w_j/p_j$.

Let us remark that our algorithms are ρ-monotone for some $\rho < 1$. Also note that if at time t there is a job j pending for a valid algorithm, the algorithm schedules a unit of some job at t.

We distinguish 3 types of charges in the charging scheme; these are depicted in Figure 1. Let (j, b) be a unit of job j scheduled by the adversary at time t.

Type 1: If the algorithm already completed j by time t, then charge w_j/p_j to j.

Type 2: Otherwise if the algorithm schedules a job unit (i, a) at time t that has capacity at least w_j/p_j then we charge w_j/p_j to i_0, where i_0 is the next job completed by the algorithm from time $t + 1$ on.

Type 3: In the remaining case, j is not pending any more for the algorithm by the its validity. Let s be the critical time of j. We charge w_j/p_j to i_0, where i_0 is the first job completed by the algorithm from time $s + 1$ on.

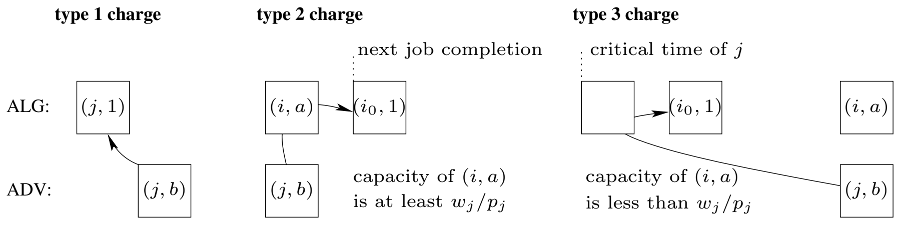


Fig. 1. The general charging scheme

Clearly every job i_0 completed by the algorithm can get at most p_{i_0} charges of type 1 summing up to at most w_{i_0}. We can bound the other types as well.

Lemma 1. *Let $\mathcal{J}$ be the set of job units that are type 3 charged to a job i_0 completed by a monotone and valid algorithm. Then for all p there are at most*

$p-1$ units $(j,b) \in \mathcal{J}$ s.t. $p_j \leq p$. In particular, $|\mathcal{J}| \leq k-1$ if all jobs have processing time at most k. Moreover, $w_j/p_j \leq \pi(i_0,1)$ holds for each $(j,b) \in \mathcal{J}$.

Proof. To be more precise we denote the elements of $\mathcal{J}$ by triplets (s,t,j) such that a job unit (j,b) scheduled at time t by the adversary is type 3 charged to i_0 and its critical time is s. Let $t_0 \geq s$ be the completion time of i_0 by the algorithm. Between s and t_0 there is no idle time, nor any other job completion, so by monotonicity and validity of the algorithm the capacities of all units in $[s,t_0)$ are at least w_j/p_j. However by definition of type 3 charges, the algorithm schedules some unit with capacity strictly smaller than w_j/p_j at t, so $t_0 \leq t$.

Since s is the critical time of j, $s+q_j(s)=d_j$. However, since the adversary schedules j at time t we have $t<d_j$. Thus $t-s<q_j(s)\leq p_j$. Note that all triplets $(s,t,j)\in\mathcal{J}$ have distinct times t. The first part of the lemma follows from the observation that there can be at most $c-1$ pairs (s,t) with distinct t that satisfy $s\leq t_0\leq t$ and $t-s<c$.

Since j was pending at time s, the unit scheduled by the algorithm at time s had capacity at least w_j/p_j. By monotonicity of the algorithm the same holds at time t_0-1, so $\pi(i_0,1)\geq w_j/p_j$. □

Lemma 2. *Let $\rho<1$. Then the total type 2 charge a job i_0 completed by a ρ-monotone and valid algorithm receives is at most $\pi(i_0,1)/(1-\rho)$.*

Proof. Let t_0 be the completion time of i_0, and let s the smallest time such that $[s,t_0)$ contains no idle time and no other job completion. Then the unit scheduled at time t_0-i for $1\leq i\leq t_0-s+1$ has capacity at most $\pi(i_0,1)\rho^{i-1}$, by ρ-monotonicity. Thus the total type 2 charge is bounded by

$$\pi(i_0,1)(1+\rho+\rho^2+\rho^3\ldots)=\pi(i_0,1)/(1-\rho) \ .$$

□

In the next sections, we adapt this general charging scheme to individual algorithms, demonstrating that the class of algorithms that can be analysed this way is very rich. Note that as this is only an analysis framework, one still needs to design their algorithm carefully, and then appropriately choose the capacity function. In particular, it is possible to analyse a fixed algorithm using different capacity functions, and their choice greatly affects the upper bound on the algorithm's competitive ratio one obtains.

All our algorithms at every step schedule the job with maximum capacity, but this is not a requirement for the scheme to work. For example, some of our preliminary algorithms did not work this way. We also believe our scheme could be adapted to the model with real numbered release times, processing times and deadlines, as in the case of *overloaded real-time systems* for example. Note that our algorithms need to select jobs only at release times or completion times of some jobs.

3 Bounded Processing Times

This time we consider instances with arbitrary weights. A natural algorithm for this model, the Smith Ratio Algorithm, schedules the pending job j that maximizes the Smith ratio w_j/p_j at every step. We sketch its analysis first, and then introduce an optimal algorithm.

Theorem 1. *The competitive ratio of* Smith Ratio Algorithm *is between* $k+1$ *and* $2k$.

Proof (sketch). The upper bound follows easily from our general charging scheme once it is established that the algorithm is $\frac{k-1}{k}$-monotone and valid w.r.t. $\pi(i,a) = w_i/a$. The lower bound follows from analysis of a simple instance with two jobs a and b s.t. $r_a = r_b = 0$, $p_a = d_a = w_a = k$, $p_b = 1$, $w_b = 1+\epsilon$, $d_b = k+1$. $\square$

The Exponential Capacity Algorithm in every step schedules the job j that maximises the value of $\pi(j, q_j) = w_j \cdot \alpha^{q_j - 1}$, which is the capacity function we use in the analysis; $\alpha < 1$ is a parameter that we specify later.

In fact, the constant α depends on k, seemingly making Exponential Capacity Algorithm semi-online. However, the $\alpha(k)$ we use is an increasing function of k, and the algorithm can be made fully online by using the value $\alpha(k^*)$ in each step, where k^* is the maximum processing time among all jobs released up to that step. Let π^* denote the capacity function defined by $\alpha(k^*)$. As π^* only increase as time goes, it is straightforward to observe that the algorithm can be analysed using the final values of k^* and π^*.

Theorem 2. *The* Exponential Capacity Algorithm *is* $(3+o(1))\,k/\ln k$-*competitive.*

Proof. As before, we use the general charging scheme. Let us define the proper value of $\alpha(k)$ now: $\alpha(k) = 1 - c^2 \cdot \ln k/k$, where $c = 1-\epsilon$ for arbitrarily small $\epsilon > 0$. The algorithm is clearly α-monotone.

To prove validity it suffices to prove that $p\alpha^{p-1} \geq 1$ for all $p \leq k$, as this implies $w_j/p_j \leq w_j\alpha^{p_j-1}$, and for any time step t, any job j pending at t, the job h scheduled by the algorithm at t and the first job i_0 completed by the algorithm from time $t+1$ on the following holds by monotonicity and the choice of π.

$$w_j\alpha^{p_j-1} \leq \pi\left(j, q_j(t)\right) \leq \pi\left(h, q_h(t)\right) \leq \pi\left(i_0, 1\right) = w_{i_0} \ . \tag{1}$$

Hence we introduce the function $f(x) = x\alpha^{x-1}$, and claim the following holds for any large enough k and any $x \in \{1, 2, \ldots, k\}$.

$$f(x) \geq 1 \qquad \text{for } 1 \leq x \leq \frac{k}{c^2 \ln k} \ , \tag{2}$$

$$f(x) \geq \ln k \qquad \text{for } \frac{k}{c^2 \ln k} < x \leq k \ . \tag{3}$$

In particular $f(x) \geq 1$ for $x \in \{1, 2, \ldots, k\}$, hence the algorithm is valid by (1).

Now we bound the total charge of type 3 any job i_0 can receive. Let $\mathcal{J}$ denote the set of job units that are type 3 charged to i_0. For each $(j,b) \in \mathcal{J}$ the charge from it is w_j/p_j, while $w_j\alpha^{p_j-1} \leq w_{i_0}$, by (1). Thus $w_j/p_j \leq w_{i_0}/(p_j\alpha^{p_j-1}) = w_{i_0}/f(p_j)$. Recall that for every $p \leq k$ the number of $(j,b) \in \mathcal{J}$ such that $p_j \leq p$

is at most $p-1$ by Lemma 1. Applying it for $p = k/(c^2 \ln k)$ and $p = k$, as well as using (2) and (3), we get

$$\sum_{(j,b)\in\mathcal{J}} 1/f(p_j) \leq \frac{k}{c^2 \ln k} + \frac{k}{\ln k} = \frac{k}{\ln k}\left(1 + \frac{1}{c^2}\right) \ .$$

Putting things together, each job i_0 completed by the algorithm receives a type 1 charge of at most w_{i_0}. By Lemma 2 for $\rho = \alpha$ it can receive at most $w_{i_0} k/c^2 \ln k$ type 2 charges in total. And we have just shown that type 3 charges are, for large k, at most $w_{i_0}(1+1/c^2)k/\ln k$ in total. Together, this is $w_{i_0}(1+2/c^2)\cdot k/\ln k = w_{i_0}(3+o(1))\cdot k/\ln k$.

It remains to prove the claims (2) and (3). To this end let us first observe that for every constant $c < 1$ and large enough x,

$$\left(1 - \frac{c}{x}\right)^x \geq \frac{1}{e} \ , \tag{4}$$

as for x tending to infinity the left hand side tends to $e^{-c} > e^{-1}$.

Clearly $f(1) = 1$, and if k is sufficiently large, then, by (4),

$$\begin{aligned} f(k) &= k\left(1 - \frac{c^2 \ln k}{k}\right)^{k-1} = k\left(1 - \frac{c^2 \ln k}{k}\right)^{\frac{k}{c \ln k}(k-1)\frac{c \ln k}{k}} \\ &\geq k\left(\frac{1}{e}\right)^{(k-1)\frac{c \ln k}{k}} = k \cdot k^{c(1-k)/k} = k^{(1-\epsilon+k\epsilon)/k} \geq \ln k \ . \end{aligned}$$

Now we observe that the sequence $(f(x))_{x=1}^{k}$ is non-decreasing for $x \leq k/(c^2 \ln k)$ and decreasing for $x > k/(c^2 \ln k)$. For this we analyze the ratio $f(x)/f(x-1) = \alpha x/(x-1)$, and see that it is at least 1 if and only if $x \geq k/(c^2 \ln k)$. Inequalities (2) and (3) follow. This completes the proof. □

We complement this result with an improved matching lower bound, as well as bounds for the case of uniform weights.

Lemma 3. *For any deterministic algorithm its competitive ratio is at least $k/\ln k - o(1)$. In particular, it is at least $k/\ln k - 0.06$ for $k \geq 16$.*

Proof (sketch). Our lower bound constructions and its analysis are similar to Ting's [13], but we are able to tighten the analysis and show that ratio $k/\ln k - o(1)$ can be forced, as opposed to $k/(2\ln k) - 1$. □

Theorem 3. *Any deterministic online algorithm for the case of uniform weights has ratio at least $\lfloor \ln k/\ln\ln k \rfloor - 1$.*

Proof (sketch). Our lower bound constructions and its analysis are similar to those of [3], but we devise it carefully so that all the job parameters are integral and yet obtain slightly better bound. □

Proposition 1 ([11]). *The competitive ratio of* SHORTEST REMAINING PROCESSING TIME FIRST *for the case of uniform weights is between* $\lfloor \log_3(2k+1) \rfloor$ *and* $2H_k \approx 2\ln k$.

Proof (sketch). The upper bound follows easily from our general charging scheme once it is established that the algorithm is $\frac{k-1}{k}$-monotone and valid w.r.t. $\pi(i,a) = 1/a$, and that there are only k different values of π. The lower bound is obtained by fine-tuning of the lower bound construction of Theorem 3 to this particular algorithm. □

4 Identical Processing Times, Upper Bound

In this section we consider instances where each job has the same processing time $k \geq 2$ and arbitrary weight.

THE CONSERVATIVE ALGORITHM: At every step execute the pending job which maximises the priority $\pi(j, q_j) = 2^{1-q_j/k} \cdot w_j$.

Theorem 4. *The* CONSERVATIVE ALGORITHM *is 5-competitive.*

Proof. The proof is based on a charging scheme, different from the general charging scheme of section 2.

Fix some instance. Consider the jobs scheduled by the algorithm and jobs scheduled by the adversary. Without loss of generality we assume that the adversary completes every job that he starts, and that he follows the EARLIEST DEADLINE FIRST policy. We also assume wlog that whenever the algorithm has no pending jobs at the very beginning of some step, the adversary will release no further jobs until he has no pending jobs as well for at least one step. This partitions the sequence into independent phases in a natural way. From now on we analyse a single phase.

Every job j scheduled by the adversary that is also completed by the algorithm, is charged to itself. From now on we ignore those jobs, and focus on the remaining ones.

All jobs scheduled by the adversary will be charged to some jobs completed by the algorithm, in such a way that job i completed by the algorithm receives a charge of at most $4w_i$ in total.

For convenience we renumber the jobs completed by the algorithm from 1 to n, such that the completion times are ordered $C_1 < \ldots < C_n$. Also we denote $C_0 = 0$. For every $i = 1, \ldots, n$ we divide $[C_{i-1}, C_i)$ further into subintervals: Let $a_i = \lceil (C_i - C_{i-1})/k \rceil$. The first subinterval is $[C_{i-1}, C_i - (a_i - 1)k)$. The remaining subintervals are $[C_i - (b+1)k, C_i - bk)$ for every $b = a_i - 2, \ldots, 0$. We label every subinterval I with a pair (b, i) such that $I = [s, C_i - bk)$ for $s = \max\{C_{i-1}, C_i - (b+1)k\}$.

The charging is done by the following procedure, which maintains for every interval $[s, t)$ a set of jobs P that are started before t by the adversary and that are not yet charged to any job of the algorithm.

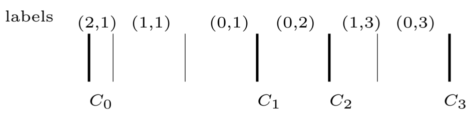


Fig. 2. The (sub)intervals as used by the charging procedure

```
Initially $P = \emptyset$.
For all subintervals $[s,t)$ as defined above in left to right order, do
  – Let $(b,i)$ be the label of the subinterval.
  – Add to $P$ all jobs $j$ started by the adversary in $[s,t)$.
  – If $P$ is not empty, then remove from $P$ the job $j$ with the smallest
    deadline and charge it to $i$. Mark $[s,t)$ with $j$.
  – If $P$ is empty, then $[s,t)$ is unmarked.
  – Denote by $P_t$ the current content of $P$.
```

Lemma 4. *For every subinterval $[s,t)$, all jobs $j \in P_t$ are still pending for the algorithm at time t.*

Proof. Assume that P_t is not empty, and let j be the earliest-deadline job in P_t.

First we claim that there is a time s_0, such that every subinterval contained in $[s_0,t)$ is marked with some job j' satisfying $s_0 \le r_{j'}$ and $d_{j'} \le d_j$. Indeed, let s_0 be the minimal starting point of any subinterval in this phase such that all the subintervals contained in $[s_0,t)$ are marked with some job j' satisfying $d_{j'} \le d_j$. If s_0 is the beginning of the phase, it trivially satisfies our requirements. In the opposite case, the very previous subinterval is marked with a job j'' such that $d_{j''} > d_j$. Thus all the jobs j' marking the subintervals contained in $[s_0,t)$ satisfy $r_{j'} \ge s_0$ for otherwise one of them would be selected instead of j'' by the marking procedure.

Now let $\mathcal{M}$ be the set of jobs charged during all subintervals in $[s_0,t)$. In an Earliest Deadline First schedule of the adversary, job j completes in a step no earlier than $s_0 + (|\mathcal{M}|+1)k$, so $d_j \ge s_0 + (|\mathcal{M}|+1)k$. As every subinterval has size at most k, $t \le |\mathcal{M}|k + s_0$ holds. Thus $d_j \ge t + k$, which shows that j is still pending for the algorithm at time t. □

Lemma 5. *Let $[s,t)$ be an subinterval with label (b,i) and j a job pending for the algorithm at some time $t_0 \in [s,t)$. Then $w_j \le 2^{1-b} w_i$.*

Proof. Let $u = C_i$ and let $x_{t_0}, x_{t_0+1}, \ldots, x_{u-1}$ be the respective priorities of the job units scheduled in $[t_0,u)$. Clearly the algorithm is $2^{-1/k}$-monotone, i.e. $x_{t'} \le 2^{-1/k} x_{t'+1}$ for every $t' \in [t_0,u)$.

We have $x_{u-1} = 2^{-1/k} w_i$ since i completes at u and the remaining processing time of i at time $(u-1)$ is 1. As the priority of j at time t_0 is at least $2^{-1} w_j$,

$$2^{-1} w_j \le \pi\left(j, q_j(t_0)\right) \le x_{t_0} \le 2^{-(u-1-t_0)/k} x_{u-1} \le 2^{-(u-t_0)/k} w_i = 2^{-b} w_i.$$

□

This lemma permits to bound the total charge of a job i completed by the algorithm. Let $a = \lceil (C_i - C_{i-1})/k \rceil$. Then i gets at most one charge of weight at most $2^{1-b} w_i$ for every $b = a-1, \ldots, 0$. Summing the bounds shows that job i receives at most 4 times its own weight, plus one possible self-charge.

At time $t = C_n$ the algorithm is idle, so $P_t = \emptyset$ by Lemma 4. Therefore all jobs scheduled by the adversary have been charged to some job of the algorithm, and this completes the proof. □

5 Identical Processing Time, Lower Bound

Theorem 5. *Any deterministic online algorithm for the equal processing time model with $k \geq 2$ has competitive ratio at least $\frac{3}{2} \cdot \sqrt{3} \approx 2.598$.*

Proof. We describe the adversary's strategy for $k = 2$ only, as it can be easily adapted to larger values of k. Every job j will be tight, i.e. $d_j = r_j + p_j = r_j + 2$. We specify the set of jobs completed by the adversary once the sequence is finished, and only describe job releases for the time being. We also assume that when there are pending jobs with positive weights, Alg will process one of them, and that it will never process a job with non-positive weight.

Initially ($t = 0$) the adversary releases a job with weight $x_0 = 1$. In every step $t > 0$ the adversary releases a job with weight x_t that we specify later, unless the algorithm has already completed one job (this has to be the one with weight x_{t-2}). In that case the adversary releases no job at time t and the sequence is finished. The adversary, in that case, completes every other job starting from the last one, for a total gain of

$$X_{t-1} = x_{t-1} + x_{t-3} + \ldots + x_{b+2} + x_b \ ,$$

where $b = t - 1 \bmod 2$, while Alg's gain is only x_{t-2}.

Now we describe the sequence x_i that forces ratio at least $R = 1.5\sqrt{3} - \epsilon$ for arbitrarily small epsilon. As we later prove, there is a non-positive element x_{i_0} in the sequence, so by previous assumptions the algorithm completes some job released before the step i_0.

If Alg completes a job released in step t, the ratio is

$$R_t = \frac{X_{t+1}}{x_t} = \frac{X_{t+1}}{X_t - X_{t-2}} \ ,$$

assuming $X_{-2} = X_{-1} = 0$. As we want to force ratio R, we let $R_t = R$, i.e.

$$X_{t+1} = R\,(X_t - X_{t-2})$$

for each $t > 0$. Note that this defines the sequence x_i, as $x_i = X_i - X_{i-2}$.

To prove existence of i_0, we introduce two sequences: $q_i = R \cdot X_{i-1}/X_{i+1}$ and $s_i = R - q_i = R(1 - X_{i-1}/X_{i+1})$. We shall derive a recursive formula defining q_i and s_i, and then prove that s_i is a strictly decreasing sequence. Next we prove by contradiction that $s_i \leq 0$ for some i.

Assume that $s_i > 0$ for all i, and observe that

$$X_i = R\left(X_{i-1} - X_{i-3}\right) = X_{i-1}\left(R - R\frac{X_{i-3}}{X_{i-1}}\right) = X_{i-1}\left(R - q_{i-2}\right) \ ,$$

which implies

$$q_i = R \cdot \frac{X_{i-1}}{X_{i+1}} = \frac{R}{(R-q_{i-1})(R-q_{i-2})} \ . \tag{5}$$

Rewriting (5) in terms of s_i, we get

$$s_i = R\left(1 - \frac{1}{s_{i-1}s_{i-2}}\right) \ , \tag{6}$$

and one can calculate that $s_0 = R$, $s_1 = R - 1/R$ and $s_2 = R(R^2-2)/(R^2-1)$; in particular $s_0 > s_1 > s_2 > 0$.

We prove by induction that s_i is a decreasing sequence. Observe that

$$s_{i+1} - s_i = R\left(\frac{1}{s_{i-1}s_{i-2}} - \frac{1}{s_i s_{i-1}}\right) = R \cdot \frac{s_i - s_{i-2}}{s_i s_{i-1} s_{i-2}} < 0 \ ,$$

since by induction hypothesis $s_{i-2} > s_{i-1} > s_i$. Since the sequence $\{s_i\}$ is positive and decreasing by assumption, it converges to $\inf s_i = g \geq 0$. Moreover, $g \geq 1$ since otherwise we would have $0 < s_{i_0-2}, s_{i_0-1} < 1$ for some i_0, which would imply $s_{i_0} < 0$ by (6), a contradiction. So $g \geq 1$, and, by (6),

$$P(g) = g^3 - Rg^2 + R = 0 \ . \tag{7}$$

Since $R = 1.5\sqrt{3} - \epsilon$, $4R^2(R^2 - 27/4)$, the discriminant of P, is negative, so P has a single real root. This sole root lies in $(-1, 0)$ as $P(-1) = -1$ and $P(0) = R > 0$. This proves that $\{s_i\}$ is not positive, contradiction. □

Discussion. The same construction was used before [5], and it was claimed to yield 2.59 lower bound on the competitive ratio. However, the proof therein concludes with a statement that for $R < 2.59$ an $x_i \leq 0$ exists.

6 Conclusion

It remains open to determine the best competitive ratio a deterministic algorithm can achieve for the equal processing time model. Even for $k = 1$ the question is not completely answered.

How much the competitive ratio can be improved by use of randomization remains unknown. The only papers we are aware of study rather restricted variants: either the case of jobs with uniform weights (though with unbounded processing times), for which there is an $O(1)$-competitive algorithm [11], or the case of tight weighted jobs only. For the latter there is a lower bound of $\Omega(\sqrt{\log k / \log\log k})$ and an upper bound of $O(\log k)$ on the competitive ratio [4]. Can a similar ratio be achieved when jobs are not tight?

We would like to thank Artur Jeż for his valuable comments.

References

[1] Baptiste, P.: An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. Oper. Res. Lett. 24(4), 175–180 (1999)

[2] Baptiste, P., Chrobak, M., Dürr, C., Jawor, W., Vakhania, N.: Preemptive scheduling of equal-length jobs to maximize weighted throughput. Operations Research Letters 32(3), 258–264 (2004)

[3] Baruah, S.K., Haritsa, J., Sharma, N.: On-line scheduling to maximize task completions. In: Real-Time Systems Symposium, December 1994, pp. 228–236 (1994)

[4] Canetti, R., Irani, S.: Bounding the power of preemption in randomized scheduling. SIAM J. Comput. 27(4), 993–1015 (1998)

[5] Chan, W.-T., Lam, T.W., Ting, H.-F., Wong, P.W.H.: New results on on-demand broadcasting with deadline via job scheduling with cancellation. In: Proc. 10th International on Computing and Combinatorics Conference, pp. 210–218 (2004)

[6] Chen, B., Potts, C.N., Woeginger, G.J.: A review of machine scheduling: Complexity, algorithms and approximability. In: Handbook of Combinatorial Optimization, vol. 3, pp. 21–169. Kluwer Academic Publishers, Dordrecht (1998)

[7] Chrobak, M., Jawor, W., Sgall, J., Tichý, T.: Online scheduling of equal-length jobs: Randomization and restarts help. SIAM J. Comput. 36(6), 1709–1728 (2007)

[8] Englert, M., Westermann, M.: Considering suppressed packets improves buer management in QoS switches. In: Proc. 18th Symp. on Discrete Algorithms (SODA), pp. 209–218. ACM/SIAM (2007)

[9] Hajek, B.: On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In: Proceedings of Conference on Information Sciences and Systems (CISS), pp. 434–438 (2001)

[10] Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. Journal of the ACM 47(4), 617–643 (2000)

[11] Kalyanasundaram, B., Pruhs, K.: Maximizing job completions online. Journal of Algorithms 49(1), 63–85 (2003)

[12] Lawler, E.L.: Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the “tower of sets” property. Mathl. Comput. Modelling 20(2), 91–106 (1994)

[13] Ting, H.-F.: A near optimal scheduler for on-demand data broadcasts. Theoretical Computer Science 401(1-3), 77–84 (2008)

[14] Vakhania, N.: A fast on-line algorithm for the preemptive scheduling of equal-length jobs on a single processor. In: Proceedings of the 2nd WSEAS International Conference on Computer Engineering and Applications, pp. 158–161 (2008)

On the Additive Constant of the k-Server Work Function Algorithm

Yuval Emek[1,⋆], Pierre Fraigniaud[2,⋆⋆], Amos Korman[2,⋆⋆], and Adi Rosén[3,⋆⋆⋆]

[1] Tel Aviv University, Tel Aviv, 69978 Israel
yuvale@eng.tau.ac.il
[2] CNRS and University Paris Diderot, France
{pierre.fraigniaud,amos.korman}@liafa.jussieu.fr
[3] CNRS and University of Paris 11, France
adiro@lri.fr

Abstract. We consider the Work Function Algorithm for the k-server problem [2,4]. We show that if the Work Function Algorithm is c-competitive, then it is also *strictly* $(2c)$-competitive. As a consequence of [4] this also shows that the Work Function Algorithm is strictly $(4k-2)$-competitive.

1 Introduction

A (deterministic) online algorithm `Alg` is said to be *c-competitive* if for all finite request sequences ρ, it holds that $\texttt{Alg}(\rho) \leq c \cdot OPT(\rho) + \beta$, where $\texttt{Alg}(\rho)$ and $OPT(\rho)$ are the costs incurred by `Alg` and the optimal algorithm, respectively, on σ and β is a constant independent of ρ. When this condition holds for $\beta = 0$, then `Alg` is said to be *strictly c-competitive*.

The k-server problem is one of the most extensively studied online problems (cf. [1]). To date, the best known competitive ratio for the k-server problem on general metric spaces is $2k-1$ [4], which is achieved by the Work Function Algorithm [2]. A lower bound of k for any metric space with at least $k+1$ nodes is also known [5]. The question whether online algorithms are strictly competitive, and in particular if there is a *strictly* competitive k-server algorithm, is of interest for two reasons. First, as a purely theoretical question. Second, at times one attempts to build a competitive online algorithm by repeatedly applying another online algorithm as a subroutine. In that case, if the online algorithm applied as a subroutine is not strictly competitive, the resulting online algorithm may not be competitive at all due to the growth of the additive constant with the length

⋆ This work was partially done during this author's visit at LIAFA, CNRS and University Paris Diderot, supported by Action COST 295 DYNAMO. Also partially supported by the Israel Science Foundation, grants 664/05 and 221/07.
⋆⋆ Additional support from the ANR project ALADDIN, by the INRIA project GANG, and by COST Action 295 DYNAMO.
⋆⋆⋆ Research partially supported by ANR projects AlgoQP and ALADDIN.

of the request sequence. In the context of the k-server Work Function Algorithm this idea was already proved fruitful [3].

In this paper we show that there exists a strictly competitive k-server algorithm for general metric spaces. In fact, we show that if the Work Function Algorithm is c-competitive, then it is also strictly $(2c)$-competitive. As a consequence of [4], we thus also show that the Work Function Algorithm is strictly $(4k-2)$-competitive.

2 Preliminaries

Let $\mathcal{M} = (V, \delta)$ be a metric space. We consider instances of the k-server problem on $\mathcal{M}$, and when clear from the context, omit the mention of the metric space. At any given time, each server resides in some node $v \in V$. A subset $X \subseteq V$, $|X| = k$, where the servers reside is called a *configuration*. The *distance* between two configurations X and Y, denoted by $D(X, Y)$, is defined as the weight of a minimum weight matching between X and Y. In every *round*, a new *request* $r \in V$ is presented and should be *served* by ensuring that a server resides on the request r. The servers can move from node to node, and the movement of a server from node x to node y incurs a *cost* of $\delta(x, y)$.

Fix some initial configuration A_0 and some finite request sequence ρ. The *work function* $w_\rho(X)$ of the configuration X with respect to ρ is the optimal cost of serving ρ starting in A_0 and ending up in configuration X. The collection of work function values $w_\rho(\cdot) = \{(X, w_\rho(X)) \mid X \subseteq V, |X| = k\}$ is referred to as the *work vector* of ρ (and initial configuration A_0).

A move of some server from node x to node y in round t is called *forced* if a request was presented at y in round t. (An empty move, in case that $x = y$, is also considered to be forced.) An algorithm for the k-server problem is said to be *lazy* if it only makes forced moves. Given some configuration X, an offline algorithm for the k-server problem is said to be *X-lazy* if in every round other than the last round, it only makes forced moves, while in the last round, it makes a forced move and it is also allowed to move servers to nodes in X from nodes not in X. Since unforced moves can always be postponed, it follows that $w_\rho(X)$ can be realized by an X-lazy (offline) algorithm for every choice of configuration X.

Given an initial configuration A_0 and a request sequence ρ, we denote the total cost paid by an online algorithm $\texttt{Alg}$ for serving ρ (in an online fashion) when it starts in A_0 by $\texttt{Alg}(A_0, \rho)$. The optimal cost for serving ρ starting in A_0 is denoted by $\texttt{Opt}(A_0, \rho) = \min_X\{w_\rho(X)\}$. The optimal cost for serving ρ starting in A_0 and ending in configuration X is denoted by $\texttt{Opt}(A_0, \rho, X) = w_\rho(X)$. (This seemingly redundant notation is found useful hereafter.)

Consider some metric space $\mathcal{M}$. In the context of the k-server problem, an algorithm $\texttt{Alg}$ is said to be *c-competitive* if for any initial configuration A_0, and any finite request sequence ρ, $\texttt{Alg}(A_0, \rho) \leq c \cdot \texttt{Opt}(A_0, \rho) + \beta$, where β may depend on the initial configuration A_0, but not on the request sequence ρ. $\texttt{Alg}$

is said to be *strictly c-competitive* if it is c-competitive with additive constant $\beta = 0$, that is, if for any initial configuration A_0 and any finite request sequence ρ, $\texttt{Alg}(A_0, \rho) \leq c \cdot \texttt{Opt}(A_0, \rho)$. As common in other works, we assume that the online algorithm and the optimal algorithm have the same initial configuration.

3 Strictly Competitive Analysis

We prove the following theorem.

Theorem 1. *If the Work Function Algorithm is c-competitive, then it is also strictly* $(2c)$*-competitive.*

In fact, we shall prove Theorem 1 for a (somewhat) larger class of k-server online algorithms, referred to as *robust* algorithms (this class will be defined soon). We say that an online algorithm for the k-server problem is *request-sequence-oblivious*, if for every initial configuration A_0, request sequence ρ, current configuration X, and request r, the action of the algorithm on r after it served ρ (starting in A_0) is fully determined by X, r, and the work vector $w_\rho(\cdot)$. In other words, a request-sequence-oblivious online algorithm can replace the explicit knowledge of A_0 and ρ with the knowledge of $w_\rho(\cdot)$. An online algorithm is said to be *robust* if it is lazy, request-sequence-oblivious, and its behavior does not change if one adds to all entries of the work vector any given value d. We prove that if a robust algorithm is c-competitive, then it is also strictly $(2c)$-competitive. Theorem 1 follows as the work function algorithm is robust.

In what follows, we consider a robust online algorithm $\texttt{Alg}$ and a lazy optimal (offline) algorithm $\texttt{Opt}$ for the k-server problem. (In some cases, $\texttt{Opt}$ will be assumed to be X-lazy for some configuration X. This will be explicitly stated.) We also consider some underlying metric $\mathcal{M} = (V, \delta)$ that we do not explicitly specify. Suppose that $\texttt{Alg}$ is α-competitive and given the initial configuration A_0, let $\beta = \beta(A_0)$ be the additive constant in the performance guarantee.

Subsequently, we fix some arbitrary initial configuration A_0 and request sequence ρ. We have to prove that $\texttt{Alg}(A_0, \rho) \leq 2\alpha\texttt{Opt}(A_0, \rho)$. A key ingredient in our proof is a designated request sequence σ referred to as the *anchor* of A_0 and ρ. Let $\ell = \min\{\delta(x, y) \mid x, y \in A_0, x \neq y\}$. Given that $A_0 = \{x_1, \ldots, x_k\}$, the anchor is defined to be

$$\sigma = (x_1 \cdots x_k)^m ,$$

where

$$m = \left\lceil \max\left\{\frac{2k\texttt{Opt}(A_0, \rho)}{\ell} + k^2, \frac{2\alpha\texttt{Opt}(A_0, \rho) + \beta(A_0)}{\ell}\right\}\right\rceil + 1 .$$

That is, the anchor consists of m *cycles* of requests presented at the nodes of A_0 in a round-robin fashion.

Informally, we shall append σ to ρ in order to ensure that both $\texttt{Alg}$ and $\texttt{Opt}$ return to the initial configuration A_0. This will allow us to analyze request sequences of the form $(\rho\sigma)^q$ as q disjoint executions on the request sequence $\rho\sigma$, thus preventing any possibility to "hide" an additive constant in the performance guarantee of $\texttt{Alg}(A_0, \rho)$. Before we can analyze this phenomenon, we have to establish some preliminary properties.

Proposition 1. *For every initial configuration A_0 and request sequence ρ, we have $\mathit{Opt}(A_0, \rho, A_0) \leq 2 \cdot \mathit{Opt}(A_0, \rho)$.*

Proof. Consider an execution η that (i) starts in configuration A_0; (ii) serves ρ optimally; and (iii) moves (optimally) to configuration A_0 at the end of round $|\rho|$. The cost of step (iii) cannot exceed that of step (ii) as we can always retrace the moves η did in step (ii) back to the initial configuration A_0. The assertion follows since η is a candidate to realize $\texttt{Opt}(A_0, \rho, A_0)$.

Since no moves are needed in order to serve the anchor σ from configuration A_0, it follows that

$$\texttt{Opt}(A_0, \rho) \leq \texttt{Opt}(A_0, \rho\sigma) \leq 2 \cdot \texttt{Opt}(A_0, \rho) \ . \tag{1}$$

Proposition 1 is also employed to establish the following lemma.

Lemma 1. *Given some configuration X, consider an X-lazy execution η that realizes $\mathit{Opt}(A_0, \rho\sigma, X)$. Then η must be in configuration A_0 at the end of round t for some $|\rho| \leq t < |\rho\sigma|$.*

Proof. Assume by way of contradiction that η's configuration at the end of round t differs from A_0 for every $|\rho| \leq t < |\rho\sigma|$. The cost $\texttt{Opt}(A_0, \rho\sigma, X)$ paid by η is at most $2 \cdot \texttt{Opt}(A_0, \rho) + D(A_0, X)$ as Proposition 1 guarantees that this is the total cost paid by an execution that (i) realizes $\texttt{Opt}(A_0, \rho, A_0)$; (ii) stays in configuration A_0 until (including) round $|\rho\sigma|$; and (iii) moves (optimally) to configuration X.

Let Y be the configuration of η at the end of round $|\rho|$. We can rewrite the total cost paid by η as $\texttt{Opt}(A_0, \rho\sigma, X) = \texttt{Opt}(A_0, \rho, Y) + \texttt{Opt}(Y, \sigma, X)$. Clearly, the former term $\texttt{Opt}(A_0, \rho, Y)$ is not smaller than $D(A_0, Y)$ which lower bounds the cost paid by any execution that starts in configuration A_0 and ends in configuration Y. We will soon prove (under the assumption that η's configuration at the end of round t differs from A_0 for every $|\rho| \leq t < |\rho\sigma|$) that the latter term $\texttt{Opt}(Y, \sigma, X)$ is (strictly) greater than $2 \cdot \texttt{Opt}(A_0, \rho) + D(Y, X)$. Therefore $D(A_0, Y) + 2 \cdot \texttt{Opt}(A_0, \rho) + D(Y, X) < \texttt{Opt}(A_0, \rho, Y) + \texttt{Opt}(Y, \sigma, X) = \texttt{Opt}(A_0, \rho\sigma, X)$. The inequality $\texttt{Opt}(A_0, \rho\sigma, X) \leq 2 \cdot \texttt{Opt}(A_0, \rho) + D(A_0, X)$ then implies that $D(A_0, X) > D(A_0, y) + D(Y, X)$, in contradiction to the triangle inequality.

It remains to prove that $\texttt{Opt}(Y, \sigma, X) > 2 \cdot \texttt{Opt}(A_0, \rho) + D(Y, X)$. For that purpose, we consider the suffix ϕ of η which corresponds to the execution on the subsequence σ (ϕ is an X-lazy execution that realizes $\texttt{Opt}(Y, \sigma, X)$). Clearly, ϕ must shift from configuration Y to configuration X, paying cost of at least $D(Y, X)$. Moreover, since ϕ is X-lazy, and by the assumption that ϕ does not reside in configuration A_0, it follows that in each of the m cycles of the round-robin, at least one server must move between two different nodes in A_0. (To see this, recall that each server's move of the lazy execution ends up in a node of A_0. On the other hand, all k servers never reside in configuration A_0.) Thus ϕ pays a cost of at least ℓ per cycle, and $m\ell$ altogether. A portion of this $m\ell$ cost can be charged on the shift from configuration Y to configuration X, but we show that the remaining cost is strictly greater than $2 \cdot \mathtt{Opt}(A_0, \rho)$, thus deriving the desired inequality $\mathtt{Opt}(Y, \sigma, X) > 2 \cdot \mathtt{Opt}(A_0, \rho) + D(Y, X)$.

The k servers make at least m moves between two different nodes in A_0 when ϕ serves the subsequence σ, hence there exists some server s that makes at least m/k such moves as part of ϕ. The total cost paid by all other servers in ϕ is bounded from below by their contribution to $D(Y, X)$. As there are k nodes in A_0, at most k out of the m/k moves made by s arrive at a new node, i.e., a node which was not previously reached by s in ϕ. Therefore at least $m/k - k$ moves of s cannot be charged on its shift from Y to X. It follows that the cost paid by s in ϕ is at least $(m/k - k)\ell$ plus the contribution of s to $D(Y, X)$. The assertion now follows by the definition of m, since $(m/k - k)\ell > 2 \cdot \mathtt{Opt}(A_0, \rho)$.

Since the optimal algorithm Opt is assumed to be lazy, Lemma 1 implies the following corollary.

Corollary 1. *If the optimal algorithm Opt serves a request sequence of the form $\rho\sigma\tau$ (for any choice of suffix τ) starting from the initial configuration A_0, then at the end of round $|\rho\sigma|$ it must be in configuration A_0.*

Consider an arbitrary configuration X. We want to prove that $w_{\rho\sigma}(X) \geq w_{\rho\sigma}(A_0) + D(A_0, X)$. To this end, assume by way of contradiction that $w_{\rho\sigma}(X) < w_{\rho\sigma}(A_0) + D(A_0, X)$. Fix $w_0 = w_{\rho\sigma}(A_0)$. Lemma 1 guarantees that an X-lazy execution η that realizes $w_{\rho\sigma}(X) = \mathtt{Opt}(A_0, \rho\sigma, X)$ must be in configuration A_0 at the end of some round $|\rho| \leq t < |\rho\sigma|$. Let w_t be the cost paid by η up to the end of round t. The cost paid by η in order to move from A_0 to X is at least $D(A_0, X)$, hence $w_{\rho\sigma}(X) \geq w_t + D(A_0, X)$. Therefore $w_t < w_0$, which derives a contradiction, since w_0 can be realized by an execution that reaches A_0 at the end of round t and stays in A_0 until it completes serving σ without paying any more cost. As $w_{\rho\sigma}(X) \leq w_{\rho\sigma}(A_0) + D(A_0, X)$, we can establish the following corollary.

Corollary 2. *For every configuration X, we have $w_{\rho\sigma}(X) = w_{\rho\sigma}(A_0) + D(A_0, X)$.*

Recall that we have fixed the initial configuration A_0 and the request sequence ρ and that σ is their anchor. We now turn to analyze the request sequence $\chi = (\rho\sigma)^q$, where q is a sufficiently large integer that will be determined soon. Corollary 1 guarantees that Opt is in the initial configuration A_0 at the end of round $|\rho\sigma|$. By induction on i, it follows that Opt is in A_0 at the end of round $i \cdot |\rho\sigma|$ for every $1 \leq i \leq q$. Therefore the total cost paid by Opt on χ is merely

$$\mathtt{Opt}(A_0, \chi) = q \cdot \mathtt{Opt}(A_0, \rho\sigma) \ . \tag{2}$$

Suppose by way of contradiction that the online algorithm $\mathtt{Alg}$, when invoked on the request sequence $\rho\sigma$ from initial configuration A_0, does not end up in A_0. Since $\mathtt{Alg}$ is lazy, we conclude that $\mathtt{Alg}$ is not in configuration A_0 at the end of round t for any $|\rho| \leq t < |\rho\sigma|$. Therefore in each cycle of the round-robin, $\mathtt{Alg}$ moves at least once between two different nodes in A_0, paying cost of at least ℓ. By the definition of m (the number of cycles), this sums up to $\mathtt{Alg}(A_0, \rho\sigma) \geq m\ell > 2\alpha\mathtt{Opt}(A_0, \rho) + \beta(A_0)$. By inequality (1), we conclude that $\mathtt{Alg}(A_0, \rho\sigma) > \alpha\mathtt{Opt}(A_0, \rho\sigma) + \beta(A_0)$, in contradiction to the performance guarantee of $\mathtt{Alg}$. It follows that $\mathtt{Alg}$ returns to the initial configuration A_0 after serving the request sequence $\rho\sigma$.

Consider some two request sequences τ and τ'. We say that the work vector $w_\tau(\cdot)$ is *d-equivalent* to the work vector $w_{\tau'}(\cdot)$, where d is some real, if $w_\tau(X) - w_{\tau'}(X) = d$ for every $X \subseteq V$, $|X| = k$. It is easy to verify that if $w_\tau(\cdot)$ is d-equivalent to $w_{\tau'}(\cdot)$, then $w_{\tau r}(\cdot)$ is d-equivalent to $w_{\tau' r}(\cdot)$ for any choice of request $r \in V$. Corollary 2 guarantees that the work vector $w_{\rho\sigma}(\cdot)$ is d-equivalent to the work vector $w_\omega(\cdot)$ for some real d, where ω stands for the empty request sequence. (In fact, d is exactly $w_{\rho\sigma}(A_0)$.) By induction on j, we show that for every prefix π of $\rho\sigma$ and for every $1 \leq i < q$ such that $|(\rho\sigma)^i\pi| = j$, the work vector $w_{(\rho\sigma)^i\pi}(\cdot)$ is d-equivalent to the work vector $w_\pi(\cdot)$ for some real d. Therefore the behavior of the robust online algorithm $\mathtt{Alg}$ on χ is merely a repetition (q times) of its behavior on $\rho\sigma$ and

$$\mathtt{Alg}(A_0, \chi) = q \cdot \mathtt{Alg}(A_0, \rho\sigma) \ . \tag{3}$$

We are now ready to establish the following inequality:

$$\begin{aligned}
\mathtt{Alg}(A_0, \rho) \leq\ & \mathtt{Alg}(A_0, \rho\sigma) \\
=\ & \frac{\mathtt{Alg}(A_0, \chi)}{q} && \text{by inequality (3)} \\
\leq\ & \frac{\alpha\mathtt{Opt}(A_0, \chi) + \beta(A_0)}{q} && \text{by the performance guarantee of } \mathtt{Alg} \\
=\ & \frac{\alpha q\mathtt{Opt}(A_0, \rho\sigma) + \beta(A_0)}{q} && \text{by inequality (2)} \\
\leq\ & \frac{2\alpha q\mathtt{Opt}(A_0, \rho) + \beta(A_0)}{q} && \text{by inequality (1)} \\
=\ & 2\alpha\mathtt{Opt}(A_0, \rho) + \frac{\beta(A_0)}{q} \ .
\end{aligned}$$

For any real $\epsilon > 0$, we can fix $q = \lceil \beta(A_0)/\epsilon \rceil + 1$ and conclude that $\mathtt{Alg}(A_0, \rho) < 2\alpha\mathtt{Opt}(A_0, \rho) + \epsilon$. Theorem 1 follows.

As the Work Function Algorithm is known to be $(2k-1)$-competitive [4], we also get the following corollary.

Corollary 3. *The Work Function Algorithm is strictly* $(4k-2)$*-competitive.*

Acknowledgments. We thank Elias Koutsoupias for useful discussions.

References

1. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
2. Chrobak, M., Larmore, L.L.: The server problem and on-line games. In: On-line algorithms: Proc. of a DIMACS Workshop. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 7, pp. 11–64 (1991)
3. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. To appear in ICALP (2009)
4. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM 42(5), 971–983 (1995)
5. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. Journal of Algorithms 11, 208–230 (1990)

A $(4 + \quad)$-Approximation for the Minimum-Weight Dominating Set Problem in Unit Disk Graphs

Thomas Erlebach[1] and Matúš Mihalák[2]

[1] Department of Computer Science, University of Leicester, England
[2] Institute of Theoretical Computer Science, ETH Zurich, Switzerland

Abstract. We present a $(4 + \epsilon)$-approximation algorithm for the problem of computing a minimum-weight dominating set in unit disk graphs, where ϵ is an arbitrarily small constant. The previous best known approximation ratio was $5+\epsilon$. The main result of this paper is a 4-approximation algorithm for the problem restricted to constant-size areas. To obtain the $(4 + \epsilon)$-approximation algorithm for the unrestricted problem, we then follow the general framework from previous constant-factor approximations for the problem: We consider the problem in constant-size areas, and combine the solutions obtained by our 4-approximation algorithm for the restricted case to get a feasible solution for the whole problem. Using the shifting technique (selecting a best solution from several considered partitionings of the problem into constant-size areas) we obtain the claimed $(4 + \epsilon)$-approximation algorithm. By combining our algorithm with a known algorithm for node-weighted Steiner trees, we obtain a 7.875-approximation for the minimum-weight connected dominating set problem in unit disk graphs.

1 Introduction

A subset $D \subseteq V$ of the vertices of an undirected graph $G = (V, E)$ is called a *dominating set* if every vertex in V is contained in D or has a neighbor in D. A vertex in D is called a *dominator*, and we say that a dominator *dominates* itself and all its neighbors. The *minimum dominating set problem* (MDS) is to compute a dominating set of smallest size. MDS belongs to the classical $\mathcal{NP}$-hard optimization problems listed in the book of Garey and Johnson [7]. MDS for general graphs is equivalent to the *set cover* problem, and can thus be approximated within a factor of $O(\log n)$ for graphs with n vertices using a greedy algorithm (see, e.g., [16]), but no better unless all problems in $\mathcal{NP}$ can be solved in $n^{O(\log \log n)}$ time [6]. If every vertex of the input graph is associated with a weight, the *minimum-weight dominating set problem* (MWDS) is to compute a dominating set of minimum weight. Approximation ratio $O(\log n)$ can also be achieved for the weighted set cover problem and thus for MWDS [7]. The variants of the problems where the dominating set is asked to be connected in the input graph are called, in an obvious way, the *minimum connected dominating set* *problem* (MCDS) and the *minimum-weight connected dominating set problem* (MWCDS), respectively. The best known approximation ratio for MWCDS in general graphs is $O(\log n)$ as well [8].

We consider the problem of computing a minimum-weight (connected) dominating set in *unit disk graphs.* A unit disk graph is a graph where every vertex is associated with a disk of unit radius in the plane and there is an edge between two vertices of the graph if the two corresponding disks intersect. These problems are $\mathcal{NP}$-hard already for the unweighted case [4,12]. We are thus interested in approximation algorithms. An algorithm for MDS (or MWDS) is called a ρ-*approximation algorithm*, and has *approximation ratio* ρ, if it runs in polynomial time and always outputs a dominating set whose size (or total weight) is at most a factor of ρ larger than the size (or total weight) of the optimal solution. The definitions for MCDS and MWCDS are analogous. A *polynomial-time approximation scheme* (PTAS) is a family of approximation algorithms with ratio $1+\varepsilon$ for every constant $\varepsilon > 0$.

Constant-factor approximation algorithms for MDS and MCDS in unit disk graphs were given by Marathe et al. [13]. For MDS in unit disk graphs, a PTAS was presented by Hunt et al. [11], based on the shifting strategy [2,9]. These algorithms, however, do not extend to the weighted version. In particular, the PTAS is based on the fact that the optimal dominating set for unit disks in a $k \times k$ square has size at most $O(k^2)$ and can thus be found in polynomial time using complete enumeration if k is a constant. In the weighted case, there is no such bound on the size of an optimal (or near-optimal) solution, as an optimal solution may consist of a large number of disks with tiny weight. For MCDS in unit disk graphs, a PTAS was presented in [3]. For unit disk graphs with bounded density, asymptotic fully polynomial-time approximation schemes (with running time polynomial in $\frac{1}{\varepsilon}$ and in the size of the input, but achieving ratio $1+\varepsilon$ only for large enough inputs) were presented for MDS and MCDS in [15].

The first constant-factor approximation algorithms for MWDS and MWCDS in unit disk graphs were given by Ambühl et al. [1], with approximation ratios 72 and 89, respectively. Huang et al. [10] presented approximation algorithms with approximation ratio $6+\epsilon$ and $10+\epsilon$, respectively. Currently the best approximation algorithm for MWDS is due to Dai and Yu [5], with approximation ratio $5+\epsilon$. Zou et al. [17] present an approximation algorithm with ratio $2.5\rho < 3.875$ for the node-weighted Steiner tree problem in unit disk graphs, where $\rho = 1+\frac{\ln 3}{2}$ is the best known approximation ratio for the classical Steiner tree problem [14]. This result can be used to connect a dominating set by adding nodes of weight at most 2.5ρ times the weight of an optimal connected dominating set, yielding the currently best approximation ratio of 8.875 for MWCDS.

Our Results. We present a $(4+\epsilon)$-approximation algorithm for MWDS in unit disk graphs. Our algorithm is based on several ideas of previous constant-factor approximation algorithms for the problem [1,10]. We partition the plane into areas of size $K\times K$, where K is an arbitrary constant. For each of these areas we consider the following subproblem: find a minimum-weight set of disks that dominate all disks that have a center in the area. The union of feasible solutions for each subproblem yields a dominating set for the original problem. Using the shifting technique as presented in [10], the loss in the approximation factor is only $(1+O(1)/K)$, i.e., if the solution for every subproblem is a ρ-approximation, then, using the shifting technique, the (best) combination of the solutions is a $(\rho+O(1)/K)$-approximation for the original problem. Thus, for any constant ϵ, one can set K such that the obtained solution is a $(\rho+\epsilon)$-approximation. We present a 4-approximation algorithm for the subproblem, which thus leads, using the shifting technique and setting K appropriately, to a $(4+\epsilon)$-approximation algorithm. We note that Huang et al. [10] presented a 6-approximation for the subproblem, and Dai and Yu [5] a 5-approximation for the subproblem. Connecting the dominating set computed by our algorithm using the node-weighted Steiner tree algorithm of Zou et al. [17], we obtain a 7.875-approximation for MWCDS, which improves the previously best approximation ratio of 8.875.

We note that independently from our work, Zou et al. [18] have also obtained a $(4+\epsilon)$-approximation algorithm for MWDS.

The Problem as a Covering Problem. We assume an instance of the problem is given by a set $\mathcal{D}$ of n weighted unit disks in the plane, where every disk $d \in \mathcal{D}$ has radius 1 and weight w_d. We denote by $\mathcal{C}$ the centers of the disks in $\mathcal{D}$. Also, for a set of disks $X \subseteq \mathcal{D}$, we denote by $w(X)$ the total weight of disks in X, i.e., $w(X) = \sum_{d \in X} w_d$.

In the following we consider the problem as a covering problem – for a set $\mathcal{C}$ of centers of disks $\mathcal{D}$, every disk of the same radius, find a minimum-weight set $\mathcal{D}' \subseteq \mathcal{D}$ of disks the union of which contains all points in $\mathcal{C}$. If a disk d contains point p, we say that the disk d *covers* point p, and that p *is covered by* d. It is not difficult to see that the original problem and this covering problem are in fact equivalent – a dominating set for input $\mathcal{D}$ of unit disks induces a solution for the covering problem given by disks of radius 2 with centers identical to centers $\mathcal{C}$, and a solution to the covering problem induces a dominating set for the original problem. Thus, given an instance of MWDS, we can consider the equivalent covering problem with disks of radius 2. Scaling the setting down by a factor of 2 (i.e., dividing the coordinates of the center of every disk by 2, and considering disks of unit radius) we obtain an instance of the covering problem with unit disks. From now on we assume we have performed such a modification to the setting, and our goal is to find a minimum-weight subset of unit disks $\mathcal{D}$ that cover all points $\mathcal{C}$.

Structure of the paper. We first present the general approach to solving the covering problem by considering covering subproblems induced by constant-size squares in Sect. 2. In Sect. 3 we present our 4-approximation algorithm for the covering subproblem. We conclude the paper in Sect. 4.

2 General Algorithm for the Covering Problem

The general algorithm follows the approach of Huang et al. [10]. First, we partition the plane into squares of size $\mu \times \mu$, where $\mu = \frac{\sqrt{2}}{2}$. The square S_{ij}, $i, j \in \mathbb{Z}$, contains points with coordinates (x, y), where $i\cdot\mu \le x < (i+1)\cdot\mu$, and $j\cdot\mu \le y < (j+1)\cdot\mu$. We say that a disk $d \in \mathcal{D}$ *is from* square S_{ij}, if the center of disk d lies in S_{ij}. For a square S_{ij} we denote by $\mathcal{D}_{ij}$ the disks from S_{ij}, and by $\mathcal{C}_{ij}$ the centers of disks in S_{ij}. Notice that the size of squares is chosen such that any disk from square S_{ij} contains the whole square S_{ij}, and thus covers all centers $\mathcal{C}_{ij}$.

Second, we consider the squares S_{ij}, $i, j \in \mathbb{Z}$, in groups, each group consisting of $k \times k$ squares, $k \ge 1$. We call such a group of squares a *block*. Formally a block $B_{a,b}$ consists of squares S_{ij}, $a \cdot k \le i < (a+1)\cdot k$, and $b \cdot k \le j < (b+1)\cdot k$. We say that a disk $d \in \mathcal{D}$ *is from* block B, if the center of d lies in B. We denote by $\mathcal{D}_B$ the set of disks from block B, and by $\mathcal{C}_B$ the set of centers from block B.

For each block B, we consider the following *covering subproblem induced by block* B: find a minimum-weight set of disks in $\mathcal{D}$ that covers the centers $\mathcal{C}_B$. Let X_B denote a feasible solution to the covering subproblem for block B. Clearly, the union X of the solutions X_B for every block B is a feasible solution for the whole covering problem. Observe that only disks from B and disks with centers at distance at most one from B need to be considered for X_B. Thus, only disks close to the boundary of every block can be part of solutions to more than one block. Consider now an optimum solution OPT for the covering problem. For an appropriate choice of the origin of the coordinate system (which causes different positioning of the grid formed by the blocks), a substantial part of OPT, in terms of the weight of disks, is formed by disks in the area formed by the "central" parts (i.e., not close to the boundary) of nonempty blocks. Thus, we can use the shifting technique to try out different choices for the origin and construct a good feasible solution X for the covering problem: Consider the $k/4$ partitionings in blocks induced by the origin set to $(p \cdot (4\mu), p \cdot (4\mu))$, $p = 0, 1, \ldots, k/4$ (observe that for every such choice of the origin the partitioning into squares S_{ij}, $i, j \in \mathbb{Z}$, is the same, just every square has now different subscripts i, j). For the partitioning induced by p, let X^p be the union of solutions for the covering subproblems induced by every block B that were obtained by a ρ-approximation algorithm. Our algorithm then returns $X = \arg\min_p w(X^p)$ as the solution to the covering problem. Generalizing (and restating) the result of Huang et al. (phase 2 in the proof of Theorem 1 in [10]) we have that X is a $(\rho + O(1)/k)$-approximation.

Lemma 1 (Generalized formulation of [10]). *Solution X is a $(\rho+O(1)/k)$-approximation for the covering problem.*

In the following section we present a 4-approximation algorithm for the covering subproblem induced by a block B. This together with the preceding discussion and Lemma 1 yields the main theorem of this paper.

Theorem 1. *There is a $(4+\epsilon)$-approximation algorithm for MWDS in unit disk graphs.*

With the node-weighted Steiner tree algorithm by Zou et al. [17] that has approximation ratio smaller than 3.875, we obtain:

Theorem 2. *There is a 7.875-approximation algorithm for MWCDS in unit disk graphs.*

3 4-Approximation for the Covering Subproblem

In this section we present a 4-approximation algorithm for the covering subproblem induced by a block B: given a block B of $k \times k$ squares S_{ij}, compute a minimum-weight set of disks that covers all points $\mathcal{C}_B$.

Let OPT_B denote the set of disks in an optimal solution for the covering subproblem. In the following, we will often write that the algorithm "guesses" certain properties of OPT_B. By this we mean that the algorithm enumerates all possible choices for the guess (there will be a polynomial number of such choices) and computes a solution for each choice. If a choice leads to an infeasible solution, the algorithm does not consider that solution anymore. The algorithm keeps the best solution found and outputs it at the end. In the analysis of the algorithm, we concentrate on the solution X_B^{guess} for which the algorithm makes the right guesses about OPT_B. As the output of the algorithm is at least as good as this solution, it is enough to show that the approximation ratio of X_B^{guess} is 4.

First, the algorithm guesses for each of the $k \times k$ squares S_{ij} in block B whether there is a disk from OPT_B in S_{ij}, or not. If yes, the algorithm also guesses one such disk (clearly, there are no more than $(n+1)^{k^2}$ guesses). The guessed disks are then added to the set X_B^{guess} (empty in the beginning) that will form a solution to the covering problem.

Next, for every square S_{ij} containing an uncovered point, the algorithm guesses whether there is a point in S_{ij} that is covered in OPT_B only by disks from regions UM and LM. The regions UM and LM lie above and below S_{ij}, respectively, and between the vertical lines that contain the vertical parts of the boundary of S_{ij} (see Fig. 1). We call such a point a *middle-unique* point. If there is a middle-unique point in S_{ij}, the algorithm further guesses whether there is a middle-unique point that is covered by a disk from UM and, if yes, the algorithm also guesses the "leftmost" and the "rightmost" (with respect to the resulting sandglass lines, see below) such point p_l and p_r (p_l and p_r can be the same point), together with the corresponding disks d_{p_l} and d_{p_r} from UM. Similarly, the algorithm guesses whether there is a middle-unique point that is covered by a disk from LM, and if yes, the algorithm also guesses the leftmost and rightmost such point q_l and q_r, together with the corresponding disks d_{q_l} and d_{q_r} from LM.

The leftmost and rightmost middle-unique points p_l, p_r, q_l, q_r define a special region inside S_{ij}, called the *sandglass* of S_{ij} [10]: The union of the *upper sandglass* and the *lower sandglass*. The upper sandglass is a region inside S_{ij} determined by the line of slope -1 going through p_l and the line of slope 1 going through p_r as outlined in Fig. 1. The lower sandglass is, defined in a similar way, a region inside S_{ij} determined by the line of slope 1 going through q_l and the line of slope -1 going through q_r. Note that the sandglass region can be empty, if there is no middle-unique point in S_{ij}. Note also that the guessed disks $d_{p_l}, d_{p_r}, d_{q_l}, d_{q_r}$ partially cover the sandglass, but there may be a region that is not covered by these four disks. We define a *sandglass point* to be a point that lies in a sandglass but is not covered by $d_{p_l}, d_{p_r}, d_{q_l}, d_{q_r}$. Again, we add these four disks to the solution set X_B^{guess}.

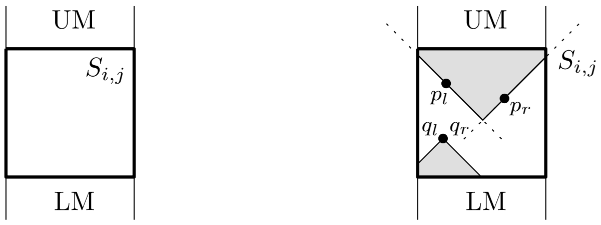


Fig. 1. Definition of regions UM and LM of a square S_{ij} (left figure). An example of a sandglass (the shaded area) for the case where $q_l = q_r$. For the upper sandglass, also the lines with slopes 1 and -1 are depicted (right figure).

The following lemma allows us to split the yet uncovered points in B into two parts, which we consider separately in the following. We say that a point *lies left of* S_{ij} if the x-coordinate of the point is smaller than the smallest x-coordinate of any point of S_{ij}. We define similarly in a natural way the notions of *lying right of*, *above*, and *below* S_{ij}.

Lemma 2 ([10]). *Any sandglass point is covered in* OPT_B *only by disks with center above or below* S_{ij}*. Any point of* S_{ij} *not contained in the sandglass of* S_{ij} *is covered in* OPT_B *only by disks with center left or right of* S_{ij}*.*

Using this lemma, we can partition the yet uncovered points into two parts. The *horizontal part* contains yet uncovered points that can be covered in OPT_B only by disks with center above or below the respective S_{ij}. The *vertical part* contains the rest of the yet uncovered points, i.e., the points that can be covered in OPT_B only by disks with center left or right of S_{ij}.

In the following we concentrate on the problem of covering the points in the horizontal part by a set of disks of minimum weight. We present a 2-approximation algorithm for this problem. Clearly, as the problem of covering the points in the vertical part can be solved by the same 2-approximation algorithm by rotating the setting by 90 degrees, we obtain a solution X_B^{guess} – all guessed disks plus the disks obtained by applying the 2-approximation algorithm to cover points in the horizontal and vertical part – which is a 4-approximation of OPT_B, thus showing the following theorem.

Theorem 3. *There is a 4-approximation algorithm for the covering subproblem in a block* B*.*

3.1 Covering Points Only from above or below

In the following we consider a generalized version of the covering problem of points in the horizontal part of block B. Let us consider k horizontal strips, each of height μ, containing m points P, where strip S_i, $i = 1, 2, \ldots, k$, lies between the horizontal lines $y = (i-1) \cdot \mu$ and $y = i \cdot \mu$. Let $\mathcal{D}$ be a set of n unit disks. We say that a disk d covers point $p \in P$ *from above*, if the center of disk d lies above the strip in which p is located. Similarly, we say that a disk d covers point $p \in P$

from below, if the center of disk d lies below the strip in which p is located. For a given set $X \subseteq \mathcal{D}$ of disks we say that *p is covered in X only from above or from below*, if p is covered by at least one disk from X, and for every disk $d \in X$, d covers p from below, or d covers p from above, or d does not cover p. We call the problem of covering points inside a horizontal strip of constant height with a minimum-weight set of disks for instances where there is an optimum solution such that every point $p \in P$ is covered in the optimal solution only from above or from below the *horizontal covering problem.*

Theorem 4. *There is a 2-approximation algorithm for the horizontal covering problem.*

Clearly, a constant-height horizontal strip can be seen as k strips of height μ, k being a constant. The simplest version of the problem is when $k = 1$. For this case, Ambühl et al. [1] present an algorithm that computes an optimum solution in polynomial time. The algorithm is based on the dynamic programming technique. The main idea is to consider the boundary of the disks that form an optimum solution inside the strip: the disks from above form in the strip the *upper envelope*, and the disks from below form in the strip the *lower envelope* of the optimum solution. The dynamic programming considers the points from left to right and stores, for each considered point and for each choice of current disks on the lower and upper envelope at that point, a minimum-weight set of disks that covers all points from the left up to the considered point.

The 2-approximation algorithm for the general case $k > 1$ can be obtained by extending this approach. Let $X \subseteq \mathcal{D}$ denote a feasible solution for the covering problem in k strips. In every strip S_i, $i = 1, \ldots, k$, we define the *upper envelope* U_i of X to be the intersection of the strip with the disks of X that lie above strip S_i. Similarly, the *lower envelope* L_i of X is the intersection of strip S_i with the disks of X that lie below S_i.

The algorithm uses a sweep line ℓ_i in every strip S_i to move on the boundary of the upper and lower envelope of every strip. Suppose we know an optimum solution OPT to the covering problem which covers every point only from above or below. Consider the upper and lower envelopes of OPT. We can sweep the lines ℓ_i through the solution OPT. All sweep lines l_i, $i = 1, 2, \ldots, k$, start somewhere to the left of the setting such that they do not intersect any disk or point. We move the sweep lines in discrete steps, always one line at a time. Every line l_i moves to the right, and visits (with its x-coordinate) the *corners* of the upper and lower envelope of strip S_i. A corner of an envelope is the intersection point of two disks which lies on the boundary of the envelope, or the intersection of a disk with one of the horizontal lines that delineate the strip, and the intersection lies on the boundary of the envelope. The sweeping process finishes when all corners of every strip have been visited. For this we make the sweep lines finish somewhere to the right of the setting, where no sweep line intersects a disk or a point of the setting. If we count the weight of every visited disk, at the end we end up with a weight that is at most three times $w(\mathrm{OPT})$, as every disk can be visited in at most three strips, and in every strip, we cannot count a disk more than once (as the disk cannot appear more than once on the boundary of an envelope [1]).

Our algorithm uses the sweeping approach to actually find a solution, i.e., to find the corners of envelopes which then define the disks in the final solution. We start with all sweep lines to the left of the setting. This indicates that no disk was chosen to cover a point in any strip. For every sweep line ℓ_i we remember the disks of the boundary of the upper and lower envelope that ℓ_i intersects at any time (for this we see the horizontal lines that define the strip as virtual disks of weight zero). The sweep line moves between corner points of the envelopes, so the disks we remember are the two disks that form the newly visited corner point, plus a disk that forms the boundary on the other envelope (upper or lower). We assume here that if a sweep line visits a corner of the upper (lower) envelope, then the lower (upper) envelope at this x-coordinate is formed by one disk only. We note that this assumption is without loss of generality, as we can initially rotate the whole problem setting so that this is true in every strip. For this purpose, we denote the current status of line ℓ_i by $((\bar{d}_l, \bar{d}_r), (\underline{d}_l, \underline{d}_r))$ with the meaning that $\bar{d}_l$ and $\bar{d}_r$ form the boundary of the upper envelope and $\underline{d}_l$ and $\underline{d}_r$ form the boundary of the lower envelope at the position of the sweep line l_i. Since we assume that at any position of the sweep line one of the two envelopes is formed by one disk only, we have $\bar{d}_l = \bar{d}_r$ or $\underline{d}_l = \underline{d}_r$. For our algorithm we require that a line ℓ_i can move from a corner point c to a corner point c' only when all points between c and c' are covered by the disks that form the boundary of the envelopes at c and at c'. (That is, for example, if line ℓ_i is at position x at state $((\bar{d}_l, \bar{d}_r), (\underline{d}_l, \underline{d}_r))$ and moves to position x' with state $((\bar{d}'_l, \bar{d}'_r), (\underline{d}'_l, \underline{d}'_r))$ then all points in strip S_i between x and x' have to be covered by disks $\bar{d}_l, \bar{d}_r, \underline{d}_l, \underline{d}_r, \bar{d}'_l, \bar{d}'_r, \underline{d}'_l, \underline{d}'_r$.) This restriction makes sure that if a sweep line gets from the start to the end, all points in the strip are covered by the chosen (visited) disks. If we do not pose any other restriction on the way the sweep line may move, we could use the dynamic programming approach of Ambühl et al. [1] for each strip individually and then combine the solutions of each strip to obtain a solution for the whole covering problem in k strips. As was shown in [10] this leads to a 3-approximation algorithm. The approximation ratio 3 comes from the fact that every disk can be counted three times, as it can appear as part of an upper or lower envelope in three strips.

We now show how to do sweeping in all strips simultaneously, achieving a better approximation ratio. We pose a new constraint on when a sweep line can move. Consider a disk d from strip S_i (i.e., the center of d lies in S_i). The disk can cover from above or from below points in at most three strips. Recall that the disk cannot cover any point in the strip S_i, as we are looking for solutions where every point is covered only from above or from below. Fig. 2 illustrates how a disk can intersect, besides S_i, two or three strips. In any case, a disk from strip S_i always intersects strips S_{i-1} and S_{i+1}. The constraint we pose on the sweep lines is that a line ℓ_{i-1} for which the lower envelope L_{i-1} is formed by disk d from strip S_i can move to the next corner point of L_{i-1} not formed by d only if disk d has already appeared on the upper envelope of the sweep line ℓ_{i+1} in strip S_{i+1} (provided that it appears on that upper envelope at all). In other words, sweep line ℓ_{i-1} at L_{i-1} formed by d can move and "leave behind" disk d

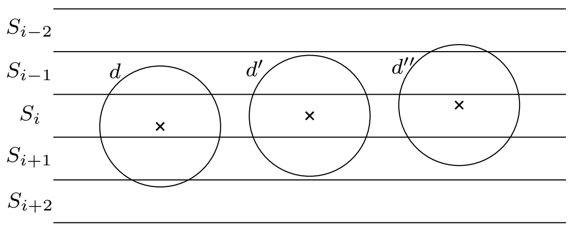


Fig. 2. Various examples of how a disk can cover points in strips. Disk d'' can cover points in S_{i-2} from below. Disks d, d' and d'' can cover points in S_{i-1} from below, and points in S_{i+1} from above. Disk d can cover points in S_{i+2} from above.

only if sweep line ℓ_{i+1} has already "met" d. If this is not the case, the line ℓ_{i-1} cannot move and we say that ℓ_{i-1} *waits* for the sweep line ℓ_{i+1}. Naturally, we pose a similar constraint for the line ℓ_{i+1} with respect to line ℓ_{i-1}, i.e., line ℓ_{i+1} can move from a corner point (d, d') of the upper envelope U_{i+1} to the right and "leave behind" disk d only if the sweep line ℓ_{i-1} has already "met" the disk d in strip S_{i-1}. We call these constraints the *move compatibility* constraints.

While sweeping through the strips, we count the weight of disks that were visited (i.e., the weight of disks that form the corner points which the sweep lines visit). We do not count, however, the weight of disk d every time (otherwise we would obtain a 3-approximation). If a line ℓ_i moves from a corner point (d, d') to a corner point (d', d''), the weight of disk d'' is added to the considered total weight only if at that moment no other sweep line contains the disk d'' already. Assume without loss of generality that d'' forms the lower envelope L_i in S_i. Then, with the previously posed constraint on how the sweep lines can move, we count the weight of the disk d'' in strips S_i and S_{i+2} only once. Thus, in total, the weight of any disk d'' used in the solution found by sweeping the lines in the strips is counted at most twice. This motivates the sweep lines to visit already used disks, as subsequent visits of a visited disk can cover points at no cost. This is the main reason why we get a 2-approximation algorithm.

We want to find a minimum-weight solution that moves the sweep lines from left to right and covers all points with visited disks. To find such a solution, we construct an auxiliary graph G_A and compute a shortest path in this graph. The vertex set V_A of the auxiliary graph G_A contains every possible configuration of the sweep lines. We will interchangeably call a vertex of G_A a configuration of the sweep lines. Clearly, every sweep line can be in at most n^3 different configurations, as there are at most n^2 corner points in every strip, and thus for any sweep line at a corner point, there can be at most n other disks forming the boundary of the other envelope. Thus, having k strips, there are no more than $O\left((n^3)^k\right)$ vertices in G_A. There are two special configurations. The *start vertex* (or the *start configuration*) s corresponds to the situation when all sweep lines are left of any disk, i.e., no disk forms an upper or lower envelope in any strip. Similarly, the *target vertex* t of G_A corresponds to the configuration where every sweep line is right of any disk. We connect the vertices in G_A with weighted edges. There is an edge between two configurations v and v' if one move of a sweep line ℓ_i in v results into the configuration v', and the move of the sweep line obeys the rule that all points between the original and new position of the moved sweep line are covered by the disks that the sweep line registers. Let d'' be the disk that forms the corner point in v' where the line ℓ_i moved to, but in v the disk was not part of the envelopes in S_i. The weight of the edge connecting v and v' is zero, if the disk d'' appears at another sweep line in v, otherwise the weight of the edge is $w_{d''}$, the weight of the disk d''.

Our algorithm finds a shortest path in G_A from s to t. This can be done in polynomial time if k is a constant. The computed path determines a move of the sweep lines from s to t, and the disks that the sweep lines meet is the solution of our algorithm. If there exists a path between s and t then clearly any such path gives a solution to the covering problem in k strips. In the following we show that the considered optimum solution OPT for the covering problem induces a path between s and t that additionally satisfies the move compatibility constraints. As our algorithm computes a shortest path between s and t, the total weight incurred by the sweep lines of our algorithm is at most the total weight incurred by the sweep lines that follow the s-t path induced by OPT. As we have argued above, the total weight of the s-t path induced by OPT is at most twice the weight of disks in OPT, as every disk in OPT can be counted by the sweep lines at most twice. Thus, the solution to the covering problem produced by our algorithm is at most twice the weight of OPT, which shows that the algorithm is a 2-approximation algorithm.

Lemma 3. *Let G_A be the auxiliary graph of the covering problem in k strips. Let* OPT *be an optimum solution for the problem. There is a path from s to t in G_A that corresponds to* OPT*, i.e., the disks visited on the s-t path are exactly the disks of* OPT*, and satisfies the move compatibility constraints.*

Proof. We will prove the claim by showing that at no point of time the sweep lines traversing the optimum solution OPT get stuck, i.e., we show that there is always a sweep line that can move to the right (unless, of course, the sweep lines are at the target configuration t).

Clearly, at the beginning, all sweep lines are left of any disk (the configuration s), and all sweep lines can move to the first disk in their respective strip (or to the end, if there is no disk of OPT in the strip). Assume for a contradiction that later in time, at a configuration $v \neq t$, no sweep line can move to the right, i.e., every sweep line ℓ_i that is not right of all disks waits for another sweep line to move first. We say that the lines are in a *deadlock*.

Let S_{i^*} be the strip with the minimum index i, $i = 1, 2, \ldots, k$, such that a sweep line ℓ_i waits for another sweep line to move. Thus, from the minimality of i^*, the sweep line ℓ_{i^*} waits for a sweep line ℓ_{i^*+2} to move. As we assume the sweep lines are in a deadlock, sweep line ℓ_{i^*+2} waits for another sweep line – ℓ_{i^*+2} waits either for ℓ_{i^*} or for ℓ_{i^*+4}. We will later show that no two sweep lines ℓ_i and ℓ_{i+2} can mutually wait for each other. Therefore, ℓ_{i^*+2} does not wait for ℓ_{i^*}, and it thus waits for ℓ_{i^*+4}. Then as the lines are in deadlock, ℓ_{i^*+4} waits

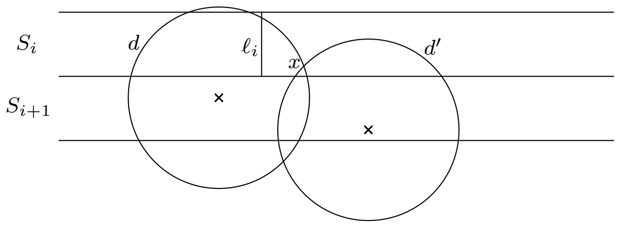


Fig. 3. Illustration for the proof of Lemma 3

either for ℓ_{i^*+2} or for ℓ_{i^*+6}. Using the same argument, ℓ_{i^*+4} waits for ℓ_{i^*+6}. Thus, using this argumentation iteratively, we end up claiming that ℓ_{i^*+2j} waits for $\ell_{i^*+2(j+1)}$, for any $j \geq 0$. This is not possible, as there are only k sweep lines.

We are left to show that the situation in which sweep lines ℓ_i and ℓ_{i+2} wait for each other does not occur. Assume such a situation. Sweep line ℓ_i waits for sweep line ℓ_{i+2} because ℓ_i wants to leave a disk d but the line ℓ_{i+2} did not pass the disk d in strip S_{i+2} yet. Similarly, line ℓ_{i+2} wants to leave a disk d' but the line ℓ_i did not pass the disk d' in strip S_i. We show that these assumptions give contradicting claims on the position of the disks d and d'. Consider now the disks d and d' alone, i.e., without the other disks of OPT. Now, as ℓ_i is currently at disk d and the line did not pass the disk d' yet, disk d' has to appear in S_i after d. This implies, however, that the center of d' is strictly right of the center of d (in terms of the x-coordinates). Fig. 3 illustrates this situation. Observe first that if disk d' (which appears right of d in S_i) intersects disk d, say at point x, then disk d' can be seen as a rotation of disk d around point x in counterclockwise direction. As the rotation leaves the center of the disk in the strip below, the rotation translates the center of the disk strictly to the right. If the disk d' does not intersect d, we can move the disk d to the right until the first moment when the translated disk d intersects d'. Repeating the argument we see that the center of d' is strictly right of the center of d.

Similarly, we can argue for the positions of disks in strip S_{i+2}, leading to the claim that the center of d' is strictly left of the center of d. This is a contradiction and the lemma follows. □

4 Conclusions

In this work we have presented a $(4+\epsilon)$-approximation algorithm for the problem of computing a minimum-weight dominating set in unit disk graphs. The main ingredient is a new 4-approximation algorithm for settings restricted to constant-size squares. This, in turn, uses a new 2-approximation algorithm for the problem of covering points in a constant-height strip only by disks from above or below. The 2-approximation algorithm finds a solution by computing a shortest path in an auxiliary graph that can be seen as mimicking a sweep-line approach with k sweep lines, which in turn mimic computing k parallel dynamic programs. This technique may be of independent interest. It remains open whether MWDS in unit disk graphs admits a PTAS.

References

1. Ambühl, C., Erlebach, T., Mihalák, M., Nunkesser, M.: Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 3–14. Springer, Heidelberg (2006)
2. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. ACM 41(1), 153–180 (1994)
3. Cheng, X., Huang, X., Li, D., Wu, W., Du., D.-Z.: A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. Networks 42(4), 202–208 (2003)
4. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discrete Math. 86(1-3), 165–177 (1990)
5. Dai, D., Yu, C.: A $5+\epsilon$-approximation algorithm for minimum weighted dominating set in unit disk graph. Theoret. Comput. Sci. 410(8-10), 756–765 (2009)
6. Feige, U.: A threshold of ln n for approximating set cover. J. ACM 45(4), 634–652 (1998)
7. Garey, M.R., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
8. Guha, S., Khuller, S.: Improved methods for approximating node weighted Steiner trees and connected dominating sets. Inform. and Comput. 150(1), 57–74 (1999)
9. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. J. ACM 32(1), 130–136 (1985)
10. Huang, Y., Gao, X., Zhang, Z., Wu, W.: A better constant-factor approximation for weighted dominating set in unit disk graph. J. Comb. Optim. (2008)
11. Hunt III, H.B., Marathe, M.V., Radhakrishnan, V., Ravi, S.S., Rosenkrantz, D.J., Stearns, R.E.: NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. J. Algorithms 26(2), 238–274 (1998)
12. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. 11(2), 329–343 (1982)
13. 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)
14. Robins, G., Zelikovsky, A.: Improved Steiner tree approximation in graphs. In: Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 770–779 (2000)
15. van Leeuwen, E.J.: Approximation algorithms for unit disk graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 351–361. Springer, Heidelberg (2005)
16. Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2001)
17. Zou, F., Li, X., Gao, S., Wu, W.: Node-weighted Steiner tree approximation in unit disk graphs. Theoret. Comput. Sci. 18(4), 342–349 (2009)
18. 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. Theoret. Comput. Sci (2009) (Article in Press)

Guard Games on Graphs: Keep the Intruder Out!

Fedor V. Fomin, Petr A. Golovach, and Daniel Lokshtanov

Department of Informatics, University of Bergen, PB 7803, 5020 Bergen, Norway
{Fedor.Fomin,Peter.Golovach,daniello}@ii.uib.no

Abstract. A team of mobile agents, called guards, tries to keep an intruder out of an assigned area by blocking all possible attacks. In a graph model for this setting, the agents and the intruder are located on the vertices of a graph, and they move from node to node via connecting edges. The area protected by the guards is a subgraph of the given graph. We investigate the algorithmic aspects of finding the minimum number of guards sufficient to patrol the area. We show that this problem is PSPACE-hard in general and proceed to investigate a variant of the game where the intruder must reach the guarded area in a single step in order to win. We show that this case approximates the general problem, and that for graphs without cycles of length 5 the minimum number of required guards in both games coincides. We give a polynomial time algorithm for solving the one-step guarding problem in graphs of bounded treewidth, and complement this result by showing that the problem is $W[1]$-hard parameterized by the treewidth of the input graph. We conclude the study of the one-step guarding problem in bounded treewidth graphs by showing that the problem is fixed parameter tractable (FPT) parameterized by the treewidth and maximum degree of the input graph. Finally, we turn our attention to a large class of sparse graphs, including planar graphs and graphs of bounded genus, namely graphs excluding some fixed apex graph as a minor. We prove that the problem is FPT and give a PTAS on apex-minor-free graphs.

1 Introduction

An intruder is trying to enter an area patrolled by a team of mobile units, for example robots. The goal of the robots is to protect the assigned area by blocking all possible attacks. We model this problem as a variant of the classical Cops and Robbers game [1] and we borrow the Cops and Robbers terminology, calling the guarding agents *cops* and the intruder a *robber*. The game of Cops and Robbers is a pursuit-evasion game played on a graph, see [3,16] for references on different pursuit-evasion and search games on graphs.

The study of *cop-robber guard games* was initiated by Fomin et al. [14]. The guard game is played on a graph G by two players, the *cop-player* and the *robber-player*. The graph G can be directed or undirected, but we only consider undirected graphs in this paper. Each player has *pawns*, the cop-player has *cops* and the robber-player has a *robber*, placed on the vertices of G. The aim of the cop-player is to prevent the robber from entering the *protected region* $C \subsetneq V$, also called the *cop-region*, and correspondingly the aim of the robber is to penetrate the protected region. The robber can not enter a vertex if it is occupied by a cop, and the cops guard the protected region C by blocking all vertices which the robber can use as entry points to C. We say that a cop *guards* the vertex v which he occupies.

The game is played in alternating turns. In their first move players choose their initial positions. The cops choose vertices inside C to occupy, and the robber chooses some vertex outside C to start in. In each subsequent turn the respective player can *move* each of his pawns to a vertex adjacent to the vertex the pawn occupies or leave the pawn in its current position. The cops are only allowed to move within the protected region C, and the robber can only move onto a vertex with no cops on it. At any time of the game both players know the positions of the cops and the robber in G. The guard game is a *robber-win* game if the robber-player can at some turn move the robber onto a vertex within C with no cop on it. In this case we say that the robber-player *wins* the game. Otherwise the cop-player can forever prevent the robber-player from winning. In this case we say that the game is a *cop-win* game, that the cop-player *wins* the game and that the cop-player can *guard* C.

The only difference between the game considered in this article and the game studied in [14] is the order of turns. In [14] the robber had to make the first move while in the problem studied here the cop-player starts the game. Despite the similar settings, the difference between the two games can be tremendous even for very simple examples. For instance consider the graph G in Figure 1 consisting of two paths P_R and P_C of same length connected by a perfect matching. The path P_C is the cop-region, and the task of the robber to enter P_C from P_R. If the robber starts first, then one cop is sufficient to guard C since the cop only needs to occupy the vertex in P_C which is matched to the vertex occupied by the robber after the robber-players move. If cops start first, their initial positions should form a dominating set of P_C because otherwise the robber player can start in a vertex adjacent to an undominated vertex in C and enter C on his next turn. Thus, to protect P_C in the "cops-first" variant of the game we need at least $\lceil (V(P_C) - 2)/3 \rceil$ cops. The algorithmic behavior of the two problems is also quite different. It was proved in [14] that when the robber's territory is a path, the "robber-first" variant of the game is solvable in polynomial time. In contrast, a simple reduction from the minimum dominating set problem shows that "cops-first" variant is NP-hard, see Proposition 2.

A different well-studied problem, the Eternal Domination problem which also is known as Eternal Security is strongly related to the guard game. In the Eternal Domination the objective is to place the minimum number of guards on the vertices of a graph G such that the guards can protect the vertices of G from an infinite sequence of attacks. In response to an attack of an unguarded vertex v, at least one guard must move to v and the other guards can either stay put, or move to adjacent vertices. Different variants of this problem

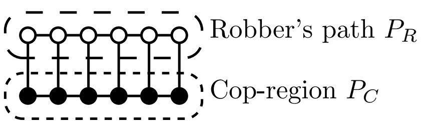


Fig. 1. Paths P_C and P_R connected by a matching

have been considered in [4,9,8,18,20,23,22]. The Eternal Domination problem is a special case of our game. This can be seen as follows. Let G be a graph on n vertices, we construct a graph H from G by adding a clique K on n vertices and connecting the clique and G by n edges which form a perfect matching. If the cop-region of H is $V(G)$ then G has an eternal dominating set of size k if and only if k cops can guard $V(G)$.

Our results. In this paper we prove a number of algorithmic and complexity results about the guarding problem. We start with a proof that the problem is PSPACE-hard on *undirected* graphs. While many games are known to be PSPACE-hard, all known PSPACE-hardness results for cops and robbers, or pursuit evasion games are for the directed graph variant of the games [14,19]. For example, the classical Cop and Robbers game was shown to be PSPACE-hard on directed graphs by Goldstein and Reingold in 1995 [19] whereas for undirected graphs, even an NP-hardness result was not known until very recently [15].

We show that the number of cops required to guard a graph is at most twice the number of cops required to protect the graph in the one-step variant of the game, that is when all players only make one move after the initial placement step. We show that this game is not only a good approximation of the general problem, but that for many graph classes like graphs without cycles of length 5 the two games are equivalent. We provide a number of FPT algorithms and parameterized complexity results for the one-step guarding problem. Our results include a polynomial time algorithm for the problem in graphs of bounded treewidth, a complexity result showing that our algorithm is essentially optimal, and an FPT algorithm for the problem parameterized by the treewidth and maximum degree of the input graph. Finally we use our treewidth-based algorithms to show that on graphs excluding some fixed apex graph as a minor the one-step guarding problem is FPT and admits a PTAS.

2 Definitions and Preliminaries

We consider finite undirected graphs without loops or multiple edges. The vertex set of a graph G is denoted by $V(G)$ and its edge set by $E(G)$, or simply by V and E if this does not create confusion. If $U \subseteq V(G)$ then the subgraph of G induced by U is denoted by $G[U]$. For a vertex v, the set of vertices which are adjacent to v is called the *(open) neighborhood* of v and denoted by $N_G(v)$. The *closed neighborhood* of v is the set $N_G[v] = N_G(v) \cup \{v\}$. If $U \subseteq V(G)$ then $N_G[U] = \bigcup_{v \in U} N_G[v]$. The *distance* $\text{dist}_G(u,v)$ between a pair of vertices u and

v in a connected graph G is the number of edges on a shortest u, v-path in G. For a positive integer r, $N_G^r[v] = \{u \in V(G) \colon \mathrm{dist}_G(u, v) \le r\}$. Whenever there is no ambiguity we omit the subscripts.

The length of a shortest cycle in G is called the *girth* of G and denoted by $g(G)$. If G is an acyclic graph then $g(G) = +\infty$. We use $\Delta(G)$ for the maximum degree of a vertex in G. Let $C \subsetneq V(G)$, and $R = V(G) \setminus C$. We call the set R where the robber moves while trying to enter C the *robber-region.* A triple $[G;\, C, R]$ is called the *board* of the game. For convenience, we keep both sets C and R in our notation despite the fact that they define each other. Clearly, the game is fully specified by the number of cops c and the board. We call the set $\delta[G;\, C, R] = \{v \in C \colon N(v) \cap R \neq \emptyset\}$ the *boundary* of the board.

Since the game is played in alternating turns starting at turn 1, the cop-player moves his cops at odd turns, and robber-player moves the robber at even turns. Two consecutive turns $2 \cdot i - 1$ and $2 \cdot i$ are jointly referred to as a *round* i, $i \ge 1$.

A *state* of the game at *time* i is given by the positions of all cops and robbers on the board after $i-1$ turns. A *strategy of a cop-player* (*strategy of a robber-player*) is a function $\mathcal{X}$ which, given the state of the game, determines the movements of the cops (the robber) in the current turn. If there are no cops (no robber) on the board, the function determines the initial positions of the cops (the robber).

The Guarding problem is, given a board $[G;\, R, C]$, to compute the minimum number of cops that can guard the protected region C. We call this number the *guard number* of the board and denote it by $\mathbf{gn}(G; C, R)$. Despite the differences between the robber-first and cops-first games, some of the results established in [14] carry over to the cops-first game. In particular, the following claim holds (see also [6,19,21]).

Proposition 1 ([14, Proposition 1]). *There is an algorithm that given an integer* $c \ge 1$ *and a board* $[G;\, C, R]$ *with the* n*-vertex graph* G *determines whether* c *cops can guard* C *in time* $\binom{|C|+c-1}{c}^2 \cdot |R|^2 \cdot n^{O(1)} = n^{O(c)}$.

Thus for every fixed c, one can decide in polynomial time whether c cops can guard the protected region against the robber on a given graph G. The running time $n^{O(c)}$ cannot be improved to an FPT running time unless $FPT = W[2]$. We refer to the book of Downey and Fellows [11] for an introduction to parameterized complexity. A reduction from the Dominating Set problem yields the following proposition.

Proposition 2. $[\star]$[1] *The* Guarding *problem is NP-hard. The* Guarding Decision *problem parameterized by the number* c *of guards is* W[2]*-hard. Finally, there is a constant* $\rho > 0$ *such that there is no polynomial time algorithm that, for every instance, approximates the guard number within a multiplicative factor* $\rho \log n$*, unless* P = NP*. Both the hardness results and the inapproximability result hold even when the robber territory is an independent set.*

[1] Proofs of results marked with $[\star]$ are omitted due the space restrictions an will appear in the journal paper.

3 Hardness of Guarding

Fomin et al. proved in [14] that the robber-first variant of GUARDING problem is PSPACE-hard for *directed* graphs. For undirected graphs only NP-hardness was proved and PSPACE-hardness was left as an open question. In this section we prove that the cops-first game is PSPACE-hard both for undirected and directed graphs. It should be noted that the complexity analysis for Cops and Robbers games for undirected graphs is much more complicated than for directed graphs (see e.g. [19]).

Theorem 1. *The* GUARDING *problem is* PSPACE*-hard on undirected graphs.*

Proof. We reduce the PSPACE-complete QUANTIFIED BOOLEAN FORMULA IN CONJUNCTIVE NORMAL FORM (QBF) problem [17] to the decision variant of the GUARDING problem. For a set of Boolean variables $x_1, x_2, \ldots, x_n$ and a Boolean formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where C_j is a clause, the QBF problem asks whether the expression $\phi = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n F$ is *true*, where for every i, Q_i is either $\forall$ or $\exists$.

Due the space restrictions we present here only the sketch of the proof. Given a quantified Boolean formula ϕ, we construct an instance of a guard game in several steps. We first construct a board $[G; C, R]$ and show that if the robber strategy is restricted to some specific conditions, then ϕ is true if and only if the cop player can win on this board with a a specific number of cops.

Constructing $[G; C, R]$. For every $Q_i x_i$ we introduce a gadget graph G_i. For $Q_i = \forall$, we define the graph $G_i(\forall)$ with vertex set $\{u_{i-1}, u_i, x_i, \overline{x}_i, y_i, \overline{y}_i, z_i, \overline{z}_i, a_i, \overline{a}_i, s_i, t_i\}$ and edge set $\{u_{i-1}y_i, y_iu_i, u_{i-1}\overline{y}_i, \overline{y}_iu_i, y_ia_i, a_iz_i, x_iz_i, \overline{y}_i\overline{a}_i, \overline{a}_i\overline{z}_i, \overline{x}_i\overline{z}_i, x_is_i, x_it_i, \overline{x}_is_i, \overline{x}_it_i\}$. Let $S_i = \{x_i, \overline{x}_i, z_i, \overline{z}_i, s_i, t_i\}$. For $Q_i = \exists$, we define $G_i(\exists)$ as the graph with vertex set $\{u_{i-1}, u_i, x_i, \overline{x}_i, y_i, z_i, a_i, s_i, t_i\}$ and edge set $\{u_{i-1}y_i, y_iu_i, y_ia_i, a_iz_i, x_iz_i, \overline{x}_iz_i, x_is_i, x_it_i, \overline{x}_is_i, \overline{x}_it_i\}$, and $S_i = \{x_i, \overline{x}_i, z_i, s_i, t_i\}$. The graphs $G_i(\forall)$ and $G_i(\exists)$ are shown in Figure 2. Observe that the vertex u_i appears both in the gadget graph G_i and in the gadget G_{i+1} for $i \in \{1, 2, \ldots, n-1\}$.

The graph G also has vertices $C_1, C_2, \ldots, C_m$ corresponding to clauses. The vertex x_i is joined with C_j by an edge if C_j contains the literal x_i, and $\overline{x}_i$ is joined with C_j if C_j contains the literal $\overline{x}_i$. The vertex u_n is connected with all vertices $C_1, C_2, \ldots, C_m$ by paths of length two with middle vertices $w_1, w_2, \ldots, w_m$. For every $i \in \{1, 2, \ldots, n\}$, the vertex s_i is joined by edges with all vertices u_j, y_j and $\overline{y}_j$ for $0 \leq j < i$, and the vertices s_i and t_i are connected by paths of length two with u_i and with all vertices u_j, y_j and $\overline{y}_j$ for $i < j \leq n$. Let W be the set of middle vertices of these paths. This completes the construction of G.

Let $C = S_1 \cup S_2 \cup \cdots \cup S_n \cup \{C_1, C_2, \ldots, C_m\}$ be the cop-region of G, and $R = V(G) \setminus C'$ be the robber-region. An example of the board $[G; C, R]$ for $\phi = \exists x_1 \forall x_2 \; x_1 \vee \overline{x}_2$ is shown in Figure 2. The paths added in the last stage of the construction are shown by dashed lines and the vertices in W are not shown.

We proceed to prove several properties of this board.

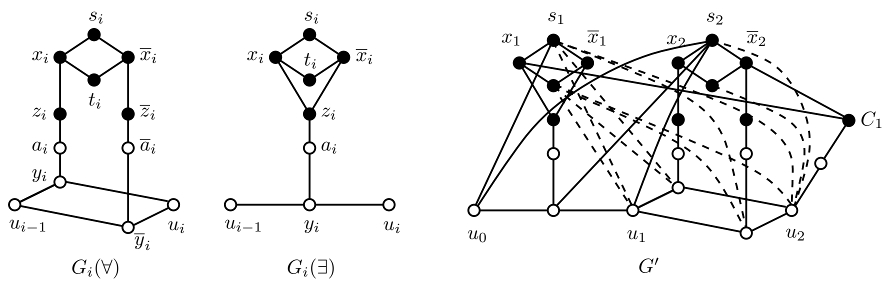


Fig. 2. Graphs $G_i(\forall)$, $G_i(\exists)$ (vertices of S_i are shown by the black color) and the board $[G;\, C, R]$

Lemma 1. [$\star$] *If $\phi = false$, then the robber-player has a winning strategy on the board $[G;\, C, R]$ against n cops. Moreover, suppose that the robber can use only strategies with the following properties:*

- *he starts from u_0,*
- *he moves along edges $u_{i-1}y_i, y_iu_i, u_{i-1}\overline{y}_i, \overline{y}_iu_i$ only in the direction induced by this ordering, i.e. these edges are "directed" for him.*

Then n cops can win on $[G;\, C, R]$ if $\phi = true$.

Then we add gadgets which force the robber to choose a particular vertex as starting vertex and to follow the restricted strategy described in Lemma 1 (otherwise he looses). These steps of reduction are omitted due the space restrictions.

The statement of Theorem 1 also holds for directed graphs since we can model an edge with two arcs, one going in each direction. Moreover, by using a simplified variant of our reduction, it can be proved that the Guarding problem is PSPACE-hard even on directed acyclic graphs.

4 One-Step Guarding

4.1 The One-Step Guard Number

In any cop-winning strategy, when the robber occupies some vertex $u \in R$, the cops should prevent him from entering C by blocking all vertices of $C \cap N(u)$. Since the robber makes his first move after the cops have chosen their initial positions, the cops have to start from an initial position such that for every vertex $u \in R$ they can occupy all vertices of $C \cap N(u)$ in one step. Thus it is not unreasonable that the number of cops needed to protect C from a robber that is only allowed to make one move after the initial step approximates the guard number of the board. Consider the variant of the game, where the robber is allowed to make only one move after the placement step. We call this variant of the game the *one-step* game. Then the minimum number of cops sufficient to guard the graph in this game is called the *one-step guard number*, and we denote the one-step guard number for the board $[G;\, C, R]$ by $\mathbf{gn}_1(G; C, R)$. We call the problem of computing the *one-step guard number* of a graph by the One-Step Guarding problem.

In the One-Step Guarding problem, a strategy for the cop-player on the board $[G;\, C, R]$ is defined as a pair $\mathcal{S} = (s, \mathcal{F})$ where

– s is a mapping assigning to every vertex v of C a non-negative integer $s(v)$ — the number of cops in v.
– $\mathcal{F} = \{f_u\}_{u \in R}$ is a family of functions $f_u \colon C \cap N(u) \to C$ defining moves of cops if the robber occupies u (a cop moves to $w \in C \cap N(u)$ from $f_u(w)$), such that for every $w \in C \cap N(u)$, $f_u(w) \in N[w]$, and for every $v \in C$, $|\{w \in C \cap N(u) \colon f_u(w) = v\}| \leq s(v)$.

If $X \subseteq C$, then $s(X) = \sum_{v \in X} s(v)$. We say that $\mathcal{S}$ is a winning strategy for c cops if $s(C) \leq c$. The simple but useful property of the one-step guard number is that it depends only on the local structure of the border neighborhood. We formalize this property in the following proposition, whose proof follows directly from the definition of one-step guarding.

Proposition 3. *For every board* $[G;\, C, R]$, $\mathbf{gn}_1(G; C, R) = \mathbf{gn}_1(G'; C', R')$ *for* $G' = G[N_G[\delta[G;\, C, R]]]$, $C' = C \cap N_G[\delta[G;\, C, R]]$ *and* $R' = R \cap N_G[\delta[G;\, C, R]]$.

The one-step guard number gives the following approximation of the guard number.

Theorem 2. [$\star$] *For any board* $[G;\, C, R]$, $\mathbf{gn}_1(G; C, R) \leq \mathbf{gn}(G; C, R) \leq 2 \cdot \mathbf{gn}_1(G; C, R)$.

A tightness of the upper bound can be seen from the following example. Let G be a graph with vertices x, y, z, such that the vertices x and y are adjacent. The vertices x and z, and the vertices y and z are joined by k paths of length two. Let $R = \{x, y\}$, and $C = V(G) \setminus R$. It can be easily shown that $\mathbf{gn}_1(G; C, R) = k$ and $\mathbf{gn}(G; C, R) = 2k$. We now show that the lower bound is tight for a large collection of boards.

Theorem 3. [$\star$] *Let* $[G;\, C, R]$ *be a board such that for every cycle* C_5 *of length* 5 *in* G, $|E(C_5) \cap E(G[R])| \neq 1$. *Then* $\mathbf{gn}_1(G; C, R) = \mathbf{gn}(G; C, R)$.

Since the Guarding Decision problem remains difficult even in the case when R is an independent set, it follows from Proposition 2 and Theorem 3 that the computation of the one step guard number is difficult too. .

Corollary 1. *The decision version of the* One-Step Guarding *problem is* NP*-complete and it remains* NP*-complete for planar graphs. Moreover, the parameterized version of the problem with k being a parameter is* W[2]*-hard. Also, there is a constant* $\rho > 0$ *such that there is no polynomial time algorithm that, for every instance, approximate the border dominating number within a multiplicative factor* $\rho \log n$, *unless* $\mathrm{P} = \mathrm{NP}$.

Despite the algorithmic lower bounds in Corollary 1, it is sometimes possible to use the one-step guard number for an approximation of the guard number.

Let us consider a generalization of the Dominating Set problem called Black and White Dominating Set problem (see e.g. [2]). The input is a *black and white* graph, which simply means that the vertex set of the graph G has been partitioned into two disjoint sets B and W of black and white vertices. Given a black and white graph G, the problem is to find a *dominating* set $X \subset V(G)$ of the minimum cardinality which dominates B, i.e. such that for each vertex $v \in B$, $N_G[v] \cap X \neq \emptyset$. We call the cardinality of such a set the *black and white domination number* and denote it by $\gamma(G; B, W)$. Observe for any cop-winning strategy the set of vertices occupied by the cops in the beginning of the game has to dominate the boundary $\delta[G;\, C, R]$. This yields the following proposition about the relationship between black and white domination and one-step graph guarding.

Proposition 4. *For any board* $[G;\, C, R]$, $\gamma(G[C]; \delta[G;\, C, R], C \setminus \delta[G;\, C, R]) \leq \mathbf{gn}_1(G; C, R)$.

These two parameters can be arbitrarily far apart. Consider the graph G constructed from two vertices u and v by joining them by k paths of length two with middle vertices $w_1, \ldots, w_k$, and let $C = \{v, w_1, \ldots, w_k\}$. Obviously, $\mathbf{gn}_1(G; C, R) = k$ and $\gamma(G[C]; \delta[G;\, C, R], C \setminus \delta[G;\, C, R]) = 1$. Still there are cases when these parameters coincide.

Proposition 5. [$\star$] *Let* $[G;\, C, R]$ *be a board such that* $g(G) \geq 5$. *Then* $\gamma(G[C]; \delta[G;\, C, R], C \setminus \delta[G;\, C, R]) = \mathbf{gn}_1(G; C, R)$.

Combining Theorem 3 and Proposition 5, we obtain the next corollary.

Corollary 2. *Let* $[G;\, C, R]$ *be a board such that* $g(G) \geq 6$. *Then* $\gamma(G[C]; \delta[G;\, C, R], C \setminus \delta[G;\, C, R]) = \mathbf{gn}_1(G; C, R) = \mathbf{gn}(G; C, R)$.

It is known [24] that the parameterized variant of the Black and White Dominating Set problem with the cardinality of dominating set being the parameter is FPT for graphs of girth at least 5. Together with Theorem 3 this yields the following corollary.

Corollary 3. *The* (One-step) Guarding *and* Guarding *problems are* FPT *when parameterized by the number of cops for boards* $[G;\, C, R]$ *with* $g(G) \geq 6$.

4.2 One-Step Guarding for Sparse Graphs

In this section we consider the One-Step Guarding problem in graphs of bounded treewidth. We denote the treewidth of G by $\mathbf{tw}(G)$ (see e.g. [7] for the definition of the treewidth). It is well known that many problems, which are difficult for generals graphs, can be solved in polynomial time in graphs of bounded treewidth. We show here that this is also the case for the computation of the one-step guard number. We construct a dynamic programming algorithm for this problem.

Theorem 4. [$\star$] *Let G be an n vertex graph given with its tree decomposition of width t. Then $\mathbf{gn}_1(G; C, R)$ can be computed in time $h(t)n^{O(t^2)}$, where h is some function of t.*

Note that this algorithm is polynomial if the treewidth if fixed, but it is not an FPT algorithm when t is the parameter. In what follows, we show that (up to widely believed assumption that FPT $\neq$ W[1]) the One-Step Guarding problem parameterized by the treewidth of the input graph is not FPT.

Theorem 5. [$\star$] *The* One-Step Guarding *problem is* W[1]*-hard when parameterized by the treewidth of the input graph.*

Using Theorem 3, we have the following corollary.

Corollary 4. *The parameterized version of the* Guarding *problem with the treewidth of the input graph being a parameter is* W[1]*-hard.*

In the following theorem we show that with some additional restrictions on graphs the One-step Guarding problem become fixed parameter tractable.

Theorem 6. [$\star$] *For any positive integers t and d, $\mathbf{gn}_1(G; C, R)$ can be computed in linear time for boards $[G;\, C, R]$, with $\mathbf{tw}(G) \leq t$ and $\Delta(G) \leq d$.*

4.3 One-Step Guarding in Apex-Minor-Free Graphs

Our results for graphs of bounded treewidth can be used for approximation of the one-step guard number for some graph classes.

It is said that a graph class $\mathcal{G}$ has *bounded local treewidth with bounding function f* if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for every graph $G \in \mathcal{G}$, every $v \in V(G)$, and every positive integer r it holds that $\mathbf{tw}(G[N^r[v]]) \leq f(r)$. An *apex graph* is a graph obtained from a planar graph G by adding a vertex and making it adjacent to some vertices of G. A graph class is *apex-minor-free* if it does not contain any graph with some fixed apex graph as a minor. For example, planar graphs (and bounded-genus graphs) are apex-minor-free graphs.

Eppstein [12,13] characterized all minor-closed graph classes that have bounded local treewidth. It was proved that they are exactly apex-minor-free graphs. These results were improved by Demaine and Hajiaghayi [10]. They proved that all apex-minor-free graphs have linear local treewidth. We show that there is a polynomial time approximation scheme (PTAS) on the class of apex-minor-free graphs for the computation of the one-step guard number. To do this we use the well known technique for solving NP-hard problems on planar graphs proposed by Baker [5] and generalized by Eppstein [12,13] (see also [10]) to minor-closed graph classes with bounded local treewidth. We start with a simple observation.

Lemma 2. [$\star$] *Let $[G_1;\, C_1, R_2]$ and $[G_2;\, C_2, R_2]$ be two boards such that $C_1 \cap R_2 = C_2 \cap R_1 = \emptyset$. Then $\mathbf{gn}_1(G; C, R) \leq \mathbf{gn}_1(G_1; C_1, R_1) + \mathbf{gn}_1(G_2, C_2, R_2)$, where $G = G_1 \cup G_2$, $C = C_1 \cup C_2$ and $R = R_1 \cup R_2$.*

Let u be a vertex of a graph G. For $i \geq 0$ we denote by L_i the i-th level of breadth first search from u, i.e. the set of vertices at distance i from u. We call the partition of the vertex set $V(G)$ $\mathcal{L}(G,u) = \{L_0, L_1, \ldots, L_r\}$ *breadth first search (BFS) decomposition* of G. We assume for convenience that for a BFS decomposition $\mathcal{L}, (G,u)$ $L_i = \emptyset$ whenever $i < 0$ or $i > r$, The BFS decomposition can be constructed using a breadth first search in a linear time.

Let $[G;\, C, R]$ be a board, and let G be a graph with BFS decomposition $\mathcal{L}(G,u) = (L_0, L_1, \ldots, L_r)$, and t be a positive integer. Suppose that $i \leq j$ are integers. For $i \leq j$, we define $G_{ij} = G[\bigcup_{p=i}^{j} L_p]$. For all $i \leq j$, we set $C_{i,j} = C \cap G_{i-2,j+2}$, $R_{i,j} = R \cap G_{i,j}$ and $F_{i,j} = G[C_{i,j} \cup R_{i,j}]$.

The following result is due to Demaine and Hajiaghayi [10] (see also the paper by Eppstein [13]),

Lemma 3 ([10]). *Let G be an apex-minor-free graph. Then $\mathbf{tw}(F_{ij}) = O(j-i)$.*

Now we are ready to describe our algorithm. Let $k \geq 4$ be an integer. For a given board $[G;\, C, R]$ for an apex-minor-free graph G, the BFS decomposition $\mathcal{L}(G,u) = (L_0, L_1, \ldots, L_r)$ is constructed for some vertex u.

If $r \leq k$ then $\mathbf{gn}_1(G; C, R)$ is computed directly. We use the fact that $\mathbf{tw}(G) = O(k)$ and, for example, use Bodlaender's Algorithm [7] to construct in linear time a suitable tree decomposition of G. Then, by Theorem 4, $\mathbf{gn}_1(G; C, R)$ can be computed in a polynomial time.

Suppose now that $r > k$. Let $F_i = F_{i,i+k-1}$, $C_i = C_{i,i+k-1}$ and $R_i = R_{i,i+k-1}$. For $i = 1, \ldots, k$, we construct boards $[F_{i+(j-1)\cdot k};\, C_{i+(j-1)\cdot k}, R_{i+(j-1)\cdot k}]$ for $0 \leq j \leq p = \lceil \frac{r-i+1}{k} \rceil + 1$, and compute

$$c_i = \sum_{j=0}^{p} \mathbf{gn}_1(F_{i+(j-1)\cdot k}; C_{i+(j-1)\cdot k}, R_{i+(j-1)\cdot k}).$$

We approximate $\mathbf{gn}_1(G; C, R)$ by the value $\mathbf{gn}'_1(G; C, R) = \min\{c_i \colon i \in \{1, \ldots, k\}\}$.

The following lemma gives properties of the algorithm.

Lemma 4. [$\star$] *For any board $[G;\, C, R]$ for an apex-minor-free graph G and for each fixed positive integer k,*

1. $\mathbf{gn}'_1(G; C, R)$ *can be computed in a polynomial time.*
2. $\mathbf{gn}_1(G; C, R) \leq \mathbf{gn}'_1(G; C, R) \leq (1 + \frac{4}{k}) \cdot \mathbf{gn}_1(G; C, R)$.

Finally, we have the following claim.

Theorem 7. *The problem of computation of the one-step guard number admits a PTAS for classes of apex-minor-free graphs and hence, for planar graphs and for graphs with bounded genus.*

Using Theorem 2, we get an approximation for the guard number.

Corollary 5. *For any $\varepsilon > 0$, the guard number can be approximated within the factor $2 + \epsilon$ in a polynomial time for apex-minor-free graph classes.*

Notice that Corollary 5 together with Proposition 3 yields a PTAS for the guard number of a restricted class of apex-minor-free graphs. This class includes, but is not limited to the apex-minor-free graphs with girth at least 6. For some special cases it is possible to get better results for planar graphs.

Theorem 8. [$\star$] *Let $[G; C, R]$ be a board such that G is a planar graph which is embedded in such a way that all vertices of $R \cap N[C]$ lay on the boundary of the external face of $G[N[C]]$. Then $\mathbf{gn}_1(G; C, R)$ can be computed polynomially.*

This theorem means that in this case the guard number can be approximated polynomially within the factor 2. Moreover, for some cases (see Proposition 3) the guard number itself can be computed polynomially. For example, when G is bipartite. Note also that it is possible to give a "symmetric" sufficient conditions for the board.

References

1. Aigner, M., Fromme, M.: A game of cops and robbers. Discrete Appl. Math. 8, 1–11 (1984)
2. Alber, J., Fan, H., Fellows, M.R., Fernau, H., Niedermeier, R., Rosamond, F., Stege, U.: A refined search tree technique for dominating set on planar graphs. J. Comput. System Sci. 71, 385–405 (2005)
3. Alspach, B.: Searching and sweeping graphs: a brief survey. Matematiche (Catania) 59, 5–37 (2006)
4. Anderson, M., Barrientos, C., Brigham, R.C., Carrington, J.R., Vitray, R.P., Yellen, J.: Maximum-demand graphs for eternal security. J. Combin. Math. Combin. Comput. 61, 111–127 (2007)
5. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. Assoc. Comput. Mach. 41, 153–180 (1994)
6. Berarducci, A., Intrigila, B.: On the cop number of a graph. Adv. in Appl. Math. 14, 389–403 (1993)
7. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
8. Burger, A.P., Cockayne, E.J., Gründlingh, W.R., Mynhardt, C.M., van Vuuren, J.H., Winterbach, W.: Finite order domination in graphs. J. Combin. Math. Combin. Comput. 49, 159–175 (2004)
9. Burger, A.P., Cockayne, E.J., Gründlingh, W.R., Mynhardt, C.M., van Vuuren, J.H., Winterbach, W.: Infinite order domination in graphs. J. Combin. Math. Combin. Comput. 50, 179–194 (2004)
10. Demaine, E.D., Hajiaghayi, M.T.: Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In: Munro, J.I. (ed.) SODA, pp. 840–849. SIAM, Philadelphia (2004)
11. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer, New York (1999)
12. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. In: SODA, pp. 632–640 (1995)

13. Eppstein, D.: Diameter and treewidth in minor-closed graph families. Diameter and treewidth in minor-closed graph families 27, 275–291 (2000)
14. Fomin, F.V., Golovach, P.A., Hall, A., Mihalák, M., Vicari, E., Widmayer, P.: How to guard a graph? In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 318–329. Springer, Heidelberg (2008)
15. Fomin, F.V., Golovach, P.A., Kratochvíl, J.: On tractability of cops and robbers game. In: IFIP, vol. 273, pp. 171–185 (2008)
16. Fomin, F.V., Thilikos, D.M.: An annotated bibliography on guaranteed graph searching. Theor. Comput. Sci. 399, 236–245 (2008)
17. Garey, M.R., Johnson, D.S.: Computers and intractability. W. H. Freeman and Co, San Francisco (1979)
18. Goddard, W., Hedetniemi, S.M., Hedetniemi, S.T.: Eternal security in graphs. J. Combin. Math. Combin. Comput. 52, 169–180 (2005)
19. Goldstein, A.S., Reingold, E.M.: The complexity of pursuit on a graph. Theoret. Comput. Sci. 143, 93–112 (1995)
20. Goldwasser, J.L., Klostermeyer, W.F.: Tight bounds for eternal dominating sets in graphs. Discrete Math. 308, 2589–2593 (2008)
21. Hahn, G., MacGillivray, G.: A note on k-cop, l-robber games on graphs. Discrete Math. 306, 2492–2497 (2006)
22. Klostermeyer, W.F.: Complexity of eternal security. J. Combin. Math. Combin. Comput. 61, 135–140 (2007)
23. Klostermeyer, W.F., MacGillivray, G.: Eternally secure sets, independence sets and cliques. AKCE Int. J. Graphs Comb. 2, 119–122 (2005)
24. Raman, V., Saurabh, S.: Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles. Algorithmica 52, 203–225 (2008)

Between a Rock and a Hard Place: The Two-to-One Assignment Problem[⋆]

Dries Goossens[1], Sergey Polyakovskiy[1], Frits C.R. Spieksma[1], and Gerhard J. Woeginger[2]

[1] Operations Research Group, Katholieke Universiteit Leuven, Naamsestraat 69, B-3000 Leuven, Belgium
[2] Department of Mathematics, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. We describe the two-to-one assignment problem, a problem in between the axial three-index assignment problem and the three-dimensional matching problem, having applications in various domains. For the (relevant) case of decomposable costs satisfying the triangle inequality we provide, on the positive side, two constant factor approximation algorithms. These algorithms involve solving minimum weight matching problems and transportation problems, leading to a 2-approximation, and a $\frac{3}{2}$-approximation. Moreover, we further show that the best of these two solutions is a $\frac{4}{3}$-approximation for our problem. On the negative side, we show that the existence of a polynomial time approximation scheme for our problem would imply P=NP.

Keywords: Assignment problem, matching problem, efficient algorithm, approximation.

1 Introduction

The two-to-one assignment problem (2-1-AP) in the title of this paper is defined as follows: Given are a set R of $2n$ red elements and a set G of n green elements. A *feasible triple* (or just *triple*, for short) consists of two distinct elements from R and of a single element from G. There is a cost-coefficient c_{ijk} given for each feasible triple, where the indices i and j run over the set R, and the index k runs over the set G. The goal is to select n triples such that each element from the ground set $R \cup G$ is used exactly once, and such that the sum of all triple costs is minimized.

The two-to-one assignment problem is closely related to the three-dimensional matching problem (3DM) and to the axial three-index assignment problem (3AP), which play the role of rock and hard place in the title. In the 3DM a single set of $3n$ elements is given, and any three elements constitute a feasible triple. In the

⋆ This research has been supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.033.403, by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society), and by OT Grant OT/07/015.

3AP three n-element sets are given, and a feasible triple consists of one element from each of the three sets. We observe that problem 2-1-AP (i) is a special case of 3DM, and (ii) does contain 3AP as a special case. To see (i), observe that adding the 'missing' cost-coefficients with a sufficiently high cost to an instance of 2-1-AP produces an equivalent instance of 3DM. An analogous observation turns any instance of 3AP into an equivalent instance of 2-1-AP, and thus yields (ii). Hence, difficulty-wise problem the two-to-one assignment problem 2-1-AP is sandwiched between the rock 3AP and the hard place 3DM.

From the practical point of view, any setting where one selects triples that consist of two elements from one set and a single element from another set can be modeled as problem 2-1-AP. We list several applications from diverse areas:

Satellite refueling. Servicing and refueling spacecraft in orbit extends the lifetime of the spacecraft, reduces launching and insurance cost, and increases operational flexibility and robustness. Dutta & Tsiotras [4] investigate a scenario where satellites exchange fuel amongst themselves in pairs. This amounts to pairing up $2n$ satellites and to assigning these pairs to n locations in orbit.

Chromosome pairing. A human cell contains two homologous chromosomes from each of the 22 chromosome classes known as the autosomes, and two sex chromosomes from the X and the Y class depending on the gender of the individual. Biyani, Wu & Sinha [1] investigate a joint classification and pairing problem, where $2n$ chromosomes have to be paired in homologues and then assigned to n autosome classes. The cost-coefficients rely on statistical properties of chromosome data and encode certain maximum likelihood estimates.

Sports scheduling. Urban & Russel [11] discuss the scheduling of football competitions that take place on n venues that are not associated with any of the $2n$ participating teams. Apart from financial constraints, the cost-coefficients of triples also encode travel distances and fan-related issues.

Gender matching. Back in 1963, Brian Wilson [12] wrote the song *"Surf City"*, which deals with a community of surfers in the California of the 1960s. The song text contains the well-known line *"Two girls for every boy!"*. A recent cover version by the Go-Gos changed the lyrics into *"Two boys for every girl!"*. Problem 2-1-AP covers a variety of scenarios in similar non-monogamous societies.

In the special case DECOM of 2-1-AP, there is a non-negative symmetric distance d_{ij} specified for every pair (i,j) of elements in $R \cup G$. The cost-coefficients of feasible triples (i,j,k) are *decomposable*, which means that they are defined as

$$c_{ijk} \;=\; d_{ij} + d_{ik} + d_{jk}. \tag{1}$$

If additionally these distances d_{ij} satisfy the triangle inequality

$$d_{ij} \;\le\; d_{ik} + d_{jk} \qquad \text{for all } i,j,k \tag{2}$$

then we denote the resulting special case of 2-1-AP as problem Δ-DECOM.

Let us first recall several special cases of 3DM and 3AP from the literature, where the underlying cost-coefficients are decomposable in a similar fashion; notice however, that in the case of cost-coefficients satisfying (1) and (2), the reductions from 3AP to 2-1-AP, and from 2-1-AP to 3DM sketched above no longer work. Crama & Spieksma [3] showed that 3AP with decomposable costs (1) does not admit any polynomial time constant factor approximation algorithm, unless P=NP. However if the distances also satisfy the triangle inequality (2), then a polynomial time 4/3-approximation algorithm becomes possible. Spieksma & Woeginger [10] prove that 3AP with decomposable costs remains NP-hard when the distances d_{ij} are the Euclidean distances of a set of points in the Euclidean plane. Magyar, Johnsson & Nevalainen [7] describe genetic algorithms for 3DM with decomposable costs of the form $c_{ijk} = \min\{d_{ij} + d_{ik}, d_{ij} + d_{jk}, d_{ik} + d_{jk}\}$. Burkard, Rudolf & Woeginger [2] deal with a variant of 3AP with decomposable costs of the form $c_{ijk} = a_i b_j c_k$; this special case is NP-hard and does not admit any polynomial time constant factor approximation algorithm unless P=NP.

Our Results

The results of Crama & Spieksma [3] for 3AP easily imply that DECOM is NP-hard, and that DECOM does not admit any polynomial time constant factor approximation algorithm unless P=NP. Therefore, we will mainly concentrate on the approximation of the more tractable variant Δ-DECOM. We will derive the following results.

– Problem Δ-DECOM is APX-hard, and thus cannot possess a PTAS unless P=NP.
– We exhibit a polynomial time 4/3-approximation algorithm for Δ-DECOM. This extends and generalizes the results of Crama & Spieksma [3].

2 APX-Hardness of Δ-DECOM

In this section we prove that Δ-DECOM is APX-hard. Our proof is through an approximation preserving reduction from the following matching problem.

Problem: Maximum bounded 3-dimensional matching (Max-3DM-B)

Instance: Three sets $X = \{x_1, \dots, x_q\}$, $Y = \{y_1, \dots, y_q\}$, and $Z = \{z_1, \dots, z_q\}$. A subset $T \subseteq X \times Y \times Z$ such that any element in X, Y, Z occurs in one, two or three triples in T; note that this implies $q \leq |T| \leq 3q$.

Goal: Find a subset T' of T of maximum cardinality such that no two triples of T' agree in any coordinate; such sets of triples are called *matchings*.

Kann [6] established that Max-3DM-B is APX-hard. An instance of this problem is called a *perfect* instance, if its optimal solution consists of q triples that

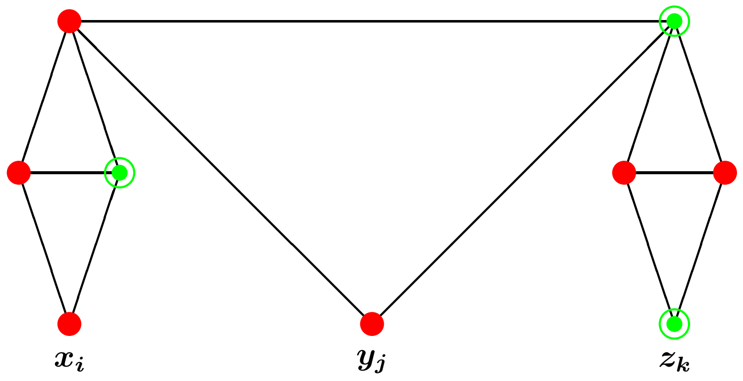


Fig. 1. The gadget in the APX-hardness proof of Δ-DECOM

cover all elements in $X \cup Y \cup Z$. Petrank [8] proved that MAX-3DM-B has a hard gap at location 1: This means that even perfect instances of MAX-3DM-B are hard to approximate, and that the existence of a PTAS for perfect instances would imply P=NP.

The rest of this section is dedicated to the APX-hardness argument for Δ-DECOM. Starting from an arbitrary instance I' of MAX-3DM-B, we will build a corresponding instance I of Δ-DECOM by using the gadget depicted in Figure 1. We note that this gadget is a simplification of another gadget that has been designed by Garey & Johnson [5] to establish NP-hardness of the problem PARTITION INTO TRIANGLES.

– For each element of $X \cup Y \cup Z$ in instance I' of MAX-3DM-B, there is a corresponding point in instance I of Δ-DECOM; these points are called *element points*.
– For each triple (x_i, y_j, z_k) in T, there are six corresponding points in instance I. These six points are called *gadget points*, and they are connected through auxiliary edges to the three element points corresponding to x_i, y_j, z_k as indicated in Figure 1.

The sets R and G in instance I are defined according to the colors in Figure 1; dark-grey points are in R, light-grey points are in G. In particular, element points that correspond to elements of $X \cup Y$ are in the set R, and element points that correspond to elements of Z are in the set G. What about the distances $d(\cdot,\cdot)$? If two points are connected by an auxiliary edge in the gadget in Figure 1, then they are at distance 1; otherwise their distance equals 2. Note that these distances satisfy the triangle inequality, and note that any two gadget points from different gadgets are at distance 2. This completes the description of the instance I of Δ-DECOM.

Note that instance I contains $3q + 6|T|$ points. Any feasible solution partitions the points into $q + 2|T|$ triangles of perimeters 3, 4, 5, and 6. Triangles of perimeter 3 are called *good*, and triangles of perimeter 4 or more are called *bad*.

The following observation will be useful.

Observation 1. *Consider a feasible solution, in which all nine points in Figure 1 belong to good triangles. Then either the three element points are all matched with two gadget points from this gadget, or none of them is matched with a point from this gadget.*

The following two lemmas are easy consequences of Observation 1.

Lemma 1. *Instance I' of* MAX-3DM-B *possesses a matching of size q, if and only if the constructed instance I of Δ-DECOM satisfies* $\mathrm{OPT}(I) = 3q + 6|T|$.

Lemma 2. *Let $\delta \geq 0$ be a real number. If instance I has a feasible solution of cost at most $3q + 6|T| + \delta q$, then instance I' possesses a matching of size at least $(1 - 4\delta)q$.*

Proof. Fix some feasible solution for instance I with cost at most $3q + 6|T| + \delta q$. Note that at most δq of the triangles in this feasible solution are bad. We call a gadget *damaged*, if at least one of its six gadget points lies in a bad triangle. We call an element point *damaged*, if (i) it is in a bad triangle, or if (ii) it is in a good triangle but together with a gadget point from a damaged gadget.

There are at most $3\delta q$ damaged element points of type (i). Furthermore, there are at most $3\delta q$ damaged gadgets, each of which may yield at most three damaged element points of type (ii). Hence altogether there are at most $12\delta q$ damaged element points, which leads to at least $3(1 - 4\delta)q$ undamaged element points. Every undamaged element point is in a triangle with two gadget points from the same undamaged gadget, and there are two other undamaged element points that are in triangles with points from the very same gadget. Hence the $3(1 - 4\delta)q$ undamaged element points can be divided into groups of three that correspond to $(1 - 4\delta)q$ undamaged gadgets. Then the corresponding $(1 - 4\delta)q$ triples in instance I' form a matching.

In case we start the construction from a perfect instance I' of MAX-3DM-B, Lemma 1 yields $\mathrm{OPT}(I) = 3q + 6|T|$ for the resulting instance I. A $(1 + \varepsilon)$-approximation algorithm for Δ-DECOM would yield an approximate objective value of at most

$$(1 + \varepsilon) \cdot \mathrm{OPT}(I) \;\leq\; 3q + 6|T| + 21\varepsilon \cdot q.$$

Here we have applied $|T| \leq 3q$. Then Lemma 2 yields a matching of size at least $(1 - 84\epsilon)q$ for instance I'. Hence, a PTAS for Δ-DECOM would imply a PTAS for perfect instance of MAX-3DM-B.

Theorem 1. *Δ-DECOM is APX-hard.*

Similar arguments can be used to show that also the special cases of problems 3AP and 3DM with decomposable costs with $d_{ij} \in \{1, 2\}$ are APX-hard.

3 Approximation Results for Δ-DECOM

We formulate and analyze three polynomial time approximation algorithms for Δ-DECOM: We first design a 2-approximation algorithm, then a

3/2-approximation algorithm, and finally combine these two algorithms to get a 4/3-approximation. Our approach adds a number of new ideas to the work of Crama & Spieksma [3]. Our methods involve solving assignment problems, weighted matching problems, and transportation problems; we refer to Schrijver [9] for an overview of methods and achievable time complexities for these problems.

3.1 The Transportation Heuristic

Consider some instance I of Δ-DECOM. Our first heuristic TP uses the following transportation problem as a main ingredient:

$$\begin{array}{lll} \min \sum_{i\in R}\sum_{k\in G} d_{ik}x_{ik} & & \\ \text{s.t.} \quad \sum_{i\in R} x_{ik} = 2 & & \text{for all } k \in G \\ \qquad \sum_{k\in G} x_{ik} = 1 & & \text{for all } i \in R \\ \qquad x_{ik} \in \{0,1\} & & \text{for all } i \in R, \text{for all } k \in G. \end{array}$$

Let x^* denote an optimal solution to this transportation problem, which assigns to every element of G two distinct elements from R. Then heuristic TP returns the corresponding feasible solution $X = \{(i,j,k) : \ x^*_{ik} = 1, x^*_{jk} = 1\}$. For the analysis of TP we fix an optimal set Z of feasible triples for instance I. We deduce successively:

$$\text{TP}(I) = \sum_{(i,j,k)\in X} (d_{ij} + d_{ik} + d_{jk}) \ \le \ 2 \sum_{(i,j,k)\in X} (d_{ik} + d_{jk}) \tag{3}$$

$$\le 2 \sum_{(i,j,k)\in Z} (d_{ik} + d_{jk}) \ \le \ 2\cdot \text{OPT}(I). \tag{4}$$

Here the inequality in (3) follows by applying the triangle inequality in order to bound d_{ij}. The first inequality in (4) holds since the transportation problem minimizes the underlying value $\sum d_{ik} + d_{jk}$. The second inequality in (4) is trivial.

To see that equality may be attained in (4), consider the instance I depicted in Figure 2. This instance has $R = \{r_1, r_2, r_3, r_4\}$ and $G = \{g_1, g_2\}$ with distances as indicated in the picture. The optimal solution $Z = \{(r_1, r_2, g_1), (r_3, r_4, g_2)\}$ has cost $\text{OPT}(I) = 4$. If the transportation problem assigns r_1 and r_3 both to g_1 and assigns r_2 and r_4 both to g_2, then TP ends up with $X = \{(r_1, r_3, g_1), (r_2, r_4, g_2)\}$, and hence $\text{TP}(I) = 8$. All in all, this yields the following theorem.

Theorem 2. *Heuristic TP is a polynomial time 2-approximation algorithm for problem Δ-DECOM. Moreover, there exist instances I for which $TP(I) = 2\,\text{OPT}(I)$.*

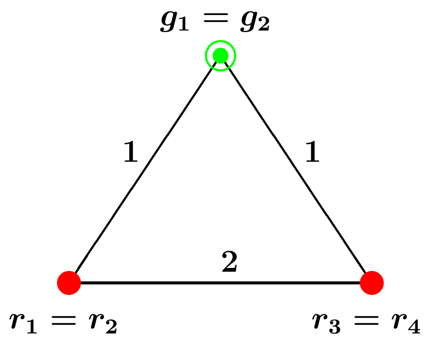


Fig. 2. A worst case instance for heuristic TP

3.2 The Match-and-Assign Heuristic

The Match-and-Assign heuristic MA for Δ-DECOM goes through two stages that both solve a minimum weight matching problem: In the first stage, MA computes a minimum weight matching M for the $2n$ elements in R under the distances d_{ij} with $i, j \in R$. In the second stage, MA computes a minimum weight assignment of the pairs (i, j) in M to the elements k in G under the costs c_{ijk}. This second stage can be described by the following integer program:

$$\begin{array}{lll} \min \sum_{(i,j)\in M} \sum_{k\in G} c_{ijk} x_{ijk} & & \\ \text{s.t.} \quad \sum_{(i,j)\in M} x_{ijk} = 1 & & \text{for all } k \in G \\ \qquad \sum_{k\in G} x_{ijk} = 1 & & \text{for all } (i,j) \in M \\ \qquad x_{ijk} \in \{0,1\} & & \text{for all } (i,j) \in M, k \in G. \end{array}$$

For an optimal solution x^* of this assignment problem, heuristic MA returns the feasible solution $X = \{(i, j, k) : x^*_{ijk} = 1\}$. For the analysis of MA we fix an optimal solution Z of I.

Lemma 3. *There exists a partition of R into two subsets R_1 and R_2 with $|R_1| = |R_2| = n$ that has the following properties.*

- *Every pair $(i, j) \in M$ has one element in R_1, and the other element in R_2.*
- *In the optimal solution Z, every element $k \in G$ is in a triple with one element $Z_1(k)$ in R_1 and one element $Z_2(k)$ in R_2.*

Proof. We construct a multi-graph with $2n$ edges on the vertex set R. The edge set contains all n edges in M; these edges are called matching-edges. Furthermore, for every triple (i, j, k) in the optimal solution Z there is a corresponding edge between i and j; these edges are called triple-edges. Since every vertex in this multi-graph is incident to exactly one matching-edge and one triple-edge, the multi-graph is a collection of even cycles.

We consistently orient the edges along every cycle, so that every vertex has precisely one in-going and one out-going arc. If a triple-edge corresponding to triple (i, j, k) is oriented from i to j, then we define $Z_1(k) = i$ and $Z_2(k) = j$, and we put i into R_1 and j into R_2. This construction satisfies all desired properties.

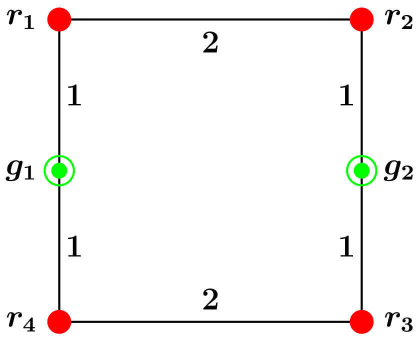


Fig. 3. A worst case instance for heuristic MA

We define another feasible solution Y that consists of n triples from $R_1 \times R_2 \times G$: A triple (i, j, k) is in Y, if and only if $(i, j) \in M$ and $Z_2(k) = j$. This yields

$$\begin{aligned}
\mathrm{MA}(I) = \sum_{(i,j,k)\in X} c_{ijk} \ &\leq \sum_{(i,j,k)\in Y} c_{ijk} && (5)\\
= \sum_{(i,j,k)\in Y} (d_{ij} + d_{ik} + d_{jk}) \ &\leq \ 2 \sum_{(i,j,k)\in Y} (d_{ij} + d_{jk}) && (6)\\
= 2 \sum_{(i,j,k)\in Y} d_{ij} + 2 \sum_{(i,j,k)\in Z} d_{jk} \ &\leq \ 2 \sum_{(i,j,k)\in Z} (d_{ij} + d_{jk}). && (7)
\end{aligned}$$

Here the inequality in (5) follows, since the second stage of MA matches the pairs in M at minimum cost with the elements of G, whereas Y matches them according to $Z_2(\cdot)$. The inequality in (6) follows by applying the triangle inequality in order to bound d_{ik}. Finally the inequality in (7) follows from the fact that matching M is the minimum cost matching for the set R.

Next, by a similar argument we get the following inequality that is perfectly symmetric to inequality (7):

$$\mathrm{MA}(I) \ \leq \ 2 \sum_{(i,j,k)\in Z} (d_{ij} + d_{ik}). \tag{8}$$

Adding (7) to (8) and using the triangle inequality entails

$$\begin{aligned}
\mathrm{MA}(I) &\leq \sum_{(i,j,k)\in Z} (2d_{ij} + d_{ik} + d_{jk})\\
&\leq \sum_{(i,j,k)\in Z} \frac{3}{2}(d_{ij} + d_{ik} + d_{jk}) \ = \ \frac{3}{2} \cdot \mathrm{OPT}(I). && (9)
\end{aligned}$$

To see that equality may hold in (9), consider the instance I in Figure 3. This instance has $R = \{r_1, r_2, r_3, r_4\}$ and $G = \{g_1, g_2\}$. The distances are as indicated in the picture; whenever two elements are not connected by an edge, their distance equals 2. An optimal solution is $Z = \{(r_1, r_4, g_1), (r_2, r_3, g_2)\}$ with cost $\mathrm{OPT}(I) = 8$. In the first stage, heuristic MA may find the matching $M = \{(r_1, r_2), (r_3, r_4)\}$. Then the second stage yields $X = \{(r_1, r_2, g_1), (r_3, r_4, g_2)\}$, and $\mathrm{MA}(I) = 12$. We summarize our results in the following theorem.

Theorem 3. *Heuristic MA is a polynomial time* 3/2*-approximation algorithm for problem Δ-DECOM. Moreover, there exist instances I for which* $MA(I) = (3/2)\,\mathrm{OPT}(I)$.

3.3 The Combined Heuristic

Do there exist instances of Δ-DECOM on which both TP and MA perform poorly (that is, close to their worst case performance guarantees)? We will demonstrate that the answer is actually no. The combined heuristic COMB runs TP and MA on instance I, and then outputs the better of the two solutions.

In the analysis of COMB we use the same notation as above. Let I be an instance of Δ-DECOM, and let Z be an optimal solution of I. We add (4), (7), and (8) to get

$$\begin{aligned} 3 \cdot \mathrm{COMB}(I) &\le \mathrm{TP}(I) + \mathrm{MA}(I) + \mathrm{MA}(I) \\ &\le 2 \sum_{(i,j,k)\in Z} (d_{ik} + d_{jk}) + 2 \sum_{(i,j,k)\in Z} (d_{ij} + d_{ik}) + 2 \sum_{(i,j,k)\in Z} (d_{ij} + d_{jk}) \\ &= 4 \sum_{(i,j,k)\in Z} (d_{ij} + d_{ik} + d_{jk}) \quad = \quad 4 \cdot \mathrm{OPT}(I). \end{aligned} \tag{10}$$

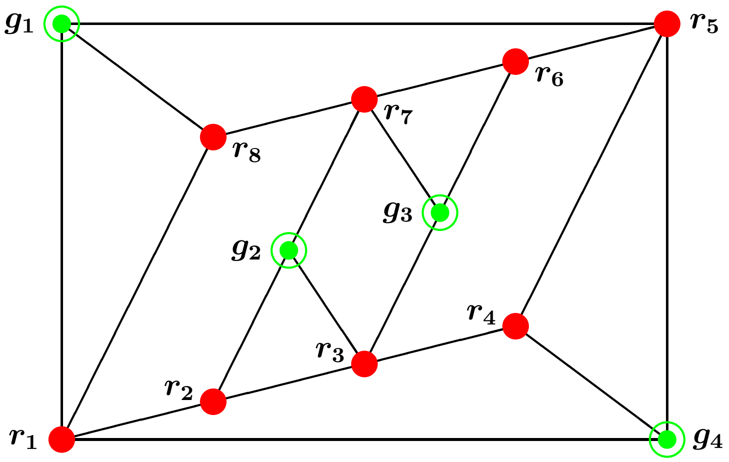


Fig. 4. A worst case instance for heuristic COMB

To see that equality may be attained in (10), consider the instance I depicted in Figure 4. This instance has $|R| = 8$ and $|G| = 4$. All edges in the picture correspond to distance 1, and all non-edges in the picture correspond to distance 2. Observe the following: If a triple (i, j, k) forms a triangle in the picture (with all three edges present), then its cost is 3. If a triple does not form a triangle, then its cost is at least 4. Now an optimal solution consists of the four triangles (r_1, r_8, g_1), (r_2, r_3, g_2), (r_6, r_7, g_3), and (r_4, r_5, g_4), and the corresponding optimal cost is $\mathrm{OPT}(I) = 12$. How do our heuristics TP and MA behave on instance I?

- Suppose that the transportation problem in heuristic TP assigns r_5 and r_8 to g_1; assigns r_2 and r_7 to g_2; assigns r_3 and r_6 to g_3; and assigns r_1 and r_4 to g_4. Then each of the four resulting triples has cost 4, and hence $\mathrm{TP}(I) = 16$.
- Suppose that the first stage of heuristic MA finds the matching (r_1, r_2), (r_3, r_4), (r_5, r_6), and (r_7, r_8). Note that none of these four pairs belongs to any triangle in Figure 4. Hence, no matter how the second stage matches the pairs to elements of G, every resulting triple will incur a cost of at least 4. This leads to $\mathrm{MA}(I) = 16$.

To summarize, the instance in Figure 4 is a worst case instance for heuristic COMB with $\mathrm{COMB}(I) = (4/3)\,\mathrm{OPT}(I)$. We formulate the following theorem.

Theorem 4. *COMB is a polynomial time* $4/3$*-approximation algorithm for* Δ*-DECOM. Moreover, there exist instances I for which COMB*$(I) = (4/3)\,\mathrm{OPT}(I)$.

4 Conclusion

We introduced the two-to-one assignment problem, and investigated the approximability of a special case of this problem, called Δ-DECOM. Possible avenues for further research include

- finding approximation algorithms for the corresponding special case of 3DM,
- investigating the geometric case of Δ-DECOM, and
- decreasing the gap between the current lower and upper bound of what is achievable (in terms of approximation)) by polynomial time algorithms.

References

1. Biyani, P., Wu, X., Sinha, A.: Joint classification and pairing of human chromosomes. IEEE/ACM Trans. on Comp. Biol. and Bioinf. 2, 102–109 (2005)
2. Burkard, R.E., Rudolf, R., Woeginger, G.J.: Three-dimensional axial assignment problems with decomposable cost coefficients. Discr. Appl. Math. 65, 123–140 (1996)
3. Crama, Y., Spieksma, F.C.R.: Approximation algorithms for three-dimensional assignment problems with triangle inequalities. Eur. J. of Oper. Res. 60, 273–279 (1992)

4. Dutta, A., Tsiotras, P.: Egalitarian peer-to-peer satellite refueling strategy. J. of Spacecraft and Rockets 45, 608–618 (2008)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
6. Kann, V.: Maximum bounded 3-dimensional matching is MAX SNP-complete. Inf. Proc. Let. 37, 27–35 (1991)
7. Magyar, G., Johnsson, M., Nevalainen, O.: An adaptive hybrid genetic algorithm for the three-matching problem. IEEE Trans. on Evol. Comp. 4, 135–146 (2000)
8. Petrank, E.: The hardness of approximation: Gap location. Comp. Compl. 4, 133–157 (1994)
9. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer, Berlin (2003)
10. Spieksma, F.C.R., Woeginger, G.J.: Geometric three-dimensional assignment problems. Eur. J. of Oper. Res. 91, 611–618 (1996)
11. Urban, T., Russel, R.: Scheduling sport competitions on multiple venues. Eur. J. of Oper. Res. 148, 302–311 (2003)
12. Wilson, B.: Surf City. Liberty Records (1963)

Scheduling and Packing Malleable Tasks with Precedence Constraints of Bounded Width

Elisabeth Günther[1], Felix G. König[1], and Nicole Megow[2]

[1] Technische Universität Berlin, Institut für Mathematik, Germany
{eguenth,fkoenig}@math.tu-berlin.de
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
nmegow@mpi-inf.mpg.de

Abstract. We study two related problems in non-preemptive scheduling and packing of malleable tasks with precedence constraints to minimize the makespan. We distinguish the scheduling variant, in which we allow the free choice of processors, and the packing variant, in which a task must be assigned to a contiguous subset of processors.

For precedence constraints of bounded width, we completely resolve the complexity status for any particular problem setting concerning width bound and number of processors, and give polynomial-time algorithms with best possible performance. For both, scheduling and packing malleable tasks, we present an FPTAS for the NP-hard problem variants and exact algorithms for all remaining special cases. To obtain the positive results, we do not require the common monotonous penalty assumption on processing times, whereas our hardness results hold even when assuming this restriction.

With the close relation between contiguous scheduling and strip packing, our FPTAS is the first (and best possible) constant factor approximation for (malleable) strip packing under special precedence constraints.

1 Introduction

Parallelism plays a key role in high performance computing. The apparent need for adequate models and algorithms for scheduling parallel task systems has attracted significant attention in scheduling theory over the past decade [4,14]. Several models have been proposed, among which scheduling malleable tasks as proposed in [17] is an important and promising model [11].

In the problem of scheduling malleable tasks, we are given a set $J = \{1,2,\ldots,n\}$ of tasks and m identical parallel processors. The tasks are *malleable*, which means that the processing time of a task $j \in J$ is a function $p_j(\alpha_j)$ depending on the number of processors $\alpha_j \in \mathbb{N}$ allotted to it. The tasks must be processed non-preemptively respecting precedence constraints given by a partial order $(J,\prec)$: For any $i, j \in J$, let $i \prec j$ denote that task i must be completed before task j starts processing. Two tasks are called *incomparable* if neither $i \prec j$ nor $j \prec i$, otherwise, they are called *comparable*. The *width* ω of a partial order is the maximum number of pairwise incomparable tasks.

An *allotment* $(\alpha_j)_{j\in J}$ and an assignment of start times $\sigma_j \geq 0$ for each task $j \in J$ establish a *feasible schedule* if the precedence constraints are respected and at no point in time the number of required processors exceeds the number of available processors m. The goal is to find a feasible schedule of minimum total length, called makespan.

Quite some research has been dedicated to malleable task scheduling since its introduction in [17]. For the problem with general precedence constraints among tasks, Lepère et al. [13] provide an approximation algorithm with performance guarantee $3+\sqrt{5} \approx 5.236$. For special cases, such as series-parallel precedence constraints and precedence constraints of bounded width, they prove a ratio of $(3+\sqrt{5})/2 \approx 2.618$ in the same paper, improving on an earlier factor $4+\varepsilon$ approximation for trees by [12]. Jansen and Zhang [11] consider the case of general precedence constraints and provide an algorithm with performance guarantee ≈ 4.730598, which they show to be asymptotically tight.

All these results crucially require the *monotonous penalty assumption*, which ensures that for any malleable task j, its processing time function $p_j(\alpha_j)$ is non-increasing, and its work function $\alpha_j p_j(\alpha_j)$ is non-decreasing. In this work, we abandon this restriction, such that an arbitrary set of feasible allotments can be prescribed (simply set the task duration for forbidden allotments to some large constant). Notice that scheduling *parallel tasks*, for which the number of allotted processors is already given, is a special case in our model if m is polynomially bounded in the input. This assumption is reasonable and quite common, see [10]. It is necessary at this point, because the input encoding of the malleable task scheduling problem is polynomial in m, while the input of a parallel task scheduling problem is polynomial in $\log m$.

A remarkable amount of literature deals with scheduling independent parallel tasks. The only work concerning precedence constraints, that we are aware of in this setting, investigates the special case of chains [2]. Therein, Błażewicz and Liu show that scheduling unit size parallel tasks with precedence constraints that form chains is NP-hard already for three processors. Assuming monotonously increasing (decreasing) processing times along chains, they give polynomial-time algorithms.

We also consider the *contiguous* variant of the malleable scheduling problem in which we require that each task is processed on a subset of processors with consecutive indices. That such a schedule is desirable in certain applications is mentioned already in [17]. Duin and van der Sluis [5] investigate contiguous scheduling of parallel tasks in the context of assigning check-in counters at airports; they call it *adjacent scheduling*. Another problem closely related to contiguous scheduling is *strip packing*, i.e., the problem of packing rectangles in a strip of width 1 such that the packing height is minimized. More generally, we define the *discrete malleable strip packing problem* as strip packing with malleable rectangles: For strip width m, each rectangle j may have a width $\alpha_j \in \{1,\ldots,m\}$, and the height of j is a function depending on α_j.

The classical strip packing problem with arbitrary precedence constraints has been investigated by Augustine et al. [1]; they present a factor $\Theta(\log n)$ approximation. As lower bounds, they use the longest chain, i.e., the largest set of pairwise comparable tasks, and the total volume to be packed. Since these bounds immediately apply for scheduling parallel tasks as well, the approximation guarantee carries over. Moreover, with the techniques in [13] or [11], the result can be transferred to malleable task scheduling with arbitrary precedence constraints if the monotonous penalty assumption holds, see [9]. Augustine et al. [1] also provide a packing instance for which the gap between the optimal value and both lower bounds is indeed $\log n$, which indicates that new ideas and bounds are necessary for improving on this performance guarantee.

Our results. We derive a fully polynomial-time approximation scheme (FPTAS) for scheduling malleable tasks under precedence constraints of bounded width, ω. This significantly improves on the previously best known approximation ratio of $(3+\sqrt{5})/2 \approx 2.618$ in [13] under the restriction to monotonous penalty functions. To the best of our knowledge, our FPTAS also constitutes the first constant factor approximation for scheduling parallel tasks with precedence constraints. The algorithm is a dynamic programming scheme based on Dilworth's well-known decomposition theorem [3] which has been widely used to solve related scheduling problems [8,13,16,18].

For the special case $\omega = m = 2$, we provide an efficient algorithm that solves the problem to optimality. We complement our positive results by showing that the problem becomes NP-hard for $\omega \geq 3$ or $m \geq 3$. Thus, our algorithms are best possible, unless P=NP.

When scheduling parallel tasks, our positive results can be extended even further, yielding an efficient exact algorithm for $\omega \geq 2$ and any m. All other cases are shown to be NP-hard. Furthermore, we resolve the complexity question for precedence constraints which form caterpillars, a special case of trees, by showing that these problems are also NP-hard for malleable tasks and even parallel tasks, for any m.

Regarding contiguous scheduling, or discrete malleable strip packing, all of our hardness results carry over. Also, the FPTAS can be adapted naturally. For the special case of $\omega = 2$, our algorithm efficiently computes optimal solutions for classical (non-malleable) strip packing. Similarly, we efficiently solve discrete malleable strip packing for $\omega = m = 2$ to optimality. Under the assumption that the width of the strip m is polynomially bounded (see e.g. also [10]), our FPTAS is the first constant factor approximation for classical strip packing under special precedence constraints. The best previous result is a general factor $\Theta(\log n)$ approximation for arbitrary precedence constraints in [1,9].

Quite notably, unlike most previous algorithms for malleable scheduling, none of our algorithms requires the monotonous penalty assumption on processing times. On the other hand, our hardness results hold even when assuming this restriction.

2 A Fully Polynomial-Time Approximation Scheme (FPTAS)

Given a scheduling instance with precedence constraints of width bounded by a constant ω, the number of tasks processed concurrently in a feasible schedule can never exceed ω . On the other hand, any maximal set A of incomparable tasks partitions the set of all tasks into two subsets containing tasks that must be processed before, respectively after, some task in A. We exploit this structure to obtain an exact dynamic programming algorithm with pseudo-polynomial running time. Then, we show how to turn this algorithm into an FPTAS.

2.1 Dynamic Programming Algorithm (DP)

The structure of our dynamic program is based on a correlation between feasible subschedules and *ideals* of orders as described in [15]. An ideal I of $(J, \prec)$ is a subset of J

such that every task of I implies all its predecessors to be elements of I. In order to respect precedence constraints, every initial part of a feasible schedule must consist of a subset of tasks fulfilling the ideal property. We will define our dynamic program in terms of finding paths in a directed graph based on ideals; reaching an ideal I' from $I \subset I'$ will correspond to feasibly extending a subschedule by the tasks in $I' \setminus I$.

Utilizing *Dilworth's Decomposition Theorem* [3] which states that for any partial order $(J, \prec)$ of width ω, there exists a partition of $(J, \prec)$ into ω chains $\mathbb{C}_1, \ldots, \mathbb{C}_\omega$, we can represent order ideals as follows. For a given chain decomposition $\mathbb{C}_1, \ldots, \mathbb{C}_\omega$, every ideal I of $(J, \prec)$ can be described by an ω-tuple $(I_i)_{i=1,\ldots,\omega}$, where component I_i indicates that the first I_i tasks of chain $\mathbb{C}_i$ are contained in I. Thus, the number of distinct ideals is bounded by n^ω. Such a chain decomposition can be found in polynomial time, see e.g. Fulkerson [6]. To simplify notation, we identify I_i with the I_i-th task of chain $\mathbb{C}_i$ and denote its processing time as $p_i(\cdot)$.

A *state* in our dynamic program is a triple $[I, \alpha, C]$ which specifies an ideal I, represented by its *front tasks* $I_1, \ldots, I_\omega$, as well as an allotment vector $\alpha = (\alpha_i)_{i=1,\ldots,\omega}$ and a vector of completion times $C = (C_i)_{i=1,\ldots,\omega}$ for the front tasks of I. This information also defines start times $\sigma_i := C_i - p_i(\alpha_i)$ for all front tasks $(I_i)_{i=1,\ldots,\omega}$. We call a state *valid*, if the number of processors used by its front tasks does not exceed the number of available processors m at any completion time C_i. If $I_i = 0$, no task of chain $\mathbb{C}_i$ is contained in I, and we set α_i and C_i to 0. We call the particular state $\oslash := [\emptyset, 0, 0]$ *start state* and every state $[I, \alpha, C]$ with $I = J$ *end state*.

Every feasible subschedule has a representation as a state defined by the allotment values and the completion times of its front tasks. We establish a state graph G by linking two valid states $F = [I, \alpha, C]$, $F' = [I', \alpha', C']$ by an arc (F, F'), if F' is an extension of F by one task j with feasible α_j and C_j. More formally, the conditions for inserting the arc are:

1. The ideals differ only in one component i, and $I'_i = I_i + 1$. All other components of I', α' and C' remain equal.
2. The start time σ'_i of I'_i in F' respects precedence constraints, i.e., $\sigma'_i \geq C_j$ for all front tasks $(I_j)_{j=1,\ldots,\omega}$ with $I_j \prec I'_i$
3. The new task starts no earlier than the other front tasks, i.e., $\sigma'_i \geq \sigma'_j$ for all $j = 1, \ldots, \omega$.

Note, that the validity of a state as well as conditions 1–3 can be checked in constant, i.e., $\mathcal{O}(\omega^2)$, time.

Condition 1 clearly ensures, that across a path P in G from $\oslash$ to an end state, every task in J is assigned exactly one α_j and σ_j. By conditions 2 and 3, and the ideal property of the states, these start times respect all precedence constraints. Finally, the number of available processors is never exceeded due to condition 3: When a new task j is added to F with start time σ_j and allotment α_j, all tasks in I active at or after σ_j are front tasks, thus they are all taken into account when determining α_j. Furthermore, the makespan of such schedule is given by the largest C_i in its end state.

We have hence established, that any path P in G corresponds to a feasible schedule with makespan determined by the end state of P. We will now prove that the converse holds as well.

Lemma 1. *Any feasible schedule S with makespan $C_{\max}$ corresponds to a path in G from $\oslash$ to an end state with latest completion time $C_{\max}$.*

Proof. Our argument is inductive, and we start with an arbitrary feasible schedule S containing all tasks in J. The graph G obviously contains an end state $F' = [I', \alpha', C']$, in which α' and C' respectively correspond to the allotment and completion times of the last tasks in the chains $\mathbb{C}_1, \ldots, \mathbb{C}_\omega$ of an appropriate chain decomposition of J. These tasks form the front tasks of F', defining ideal I' containing all tasks in J. Let j denote a front task of I' with the latest start time. Now $I := I' \setminus \{j\}$ is again an ideal. Thus, there is a valid state $F = [I, \alpha, C]$ in G with α and C corresponding to the allotment values and completion times of the front tasks of I in S. By construction and the feasibility of S the states F and F' fulfill conditions 1–3. Hence, G contains the edge (F, F'). By induction, this yields the desired result. □

With Lemma 1 we find an optimal schedule as follows: We search for an end state with minimum makespan reachable from the start state, and create the schedule by backtracking.

The number of distinct ideals I was already mentioned to be bounded by n^ω, whereas the number of feasible allotments for each ideal does not exceed m^ω. The optimal makespan is at most by $Z_{UB} = np_{\max}$ with $p_{\max} := \max\{p_j(m) \mid j \in J\}$. Thus, assuming w.l.o.g. that processing times are integral (standard scaling argument), the task completion time can attain up to Z_{UB} different values. Hence, the number of valid states is bounded by $n^\omega m^\omega Z_{UB}^\omega$ which gives a bound on the overall running time of the dynamic programming algorithm, $\mathcal{O}(n^{2\omega} m^{2\omega} Z_{UB}^{2\omega})$.

Theorem 1. *For a given scheduling instance with precedence constraints of width ω, algorithm DP finds a feasible solution with minimum makespan in time $\mathcal{O}(n^{2\omega} m^{2\omega} Z_{UB}^{2\omega})$.*

2.2 FPTAS

In a fully polynomial time algorithm, we cannot afford to consider all values in $[0, Z_{UB}]$ for possible completion times of front tasks. Using a standard rounding technique, this number can be reduced to be polynomially bounded in the input size and $1/\varepsilon$ at the cost of increasing the makespan by at most a factor $(1+\varepsilon)$ in the following way.

For a given parameter $\varepsilon > 0$, we partition the interval $[0, Z_{UB}]$ into subintervals of size $\varepsilon Z_{LB}/n$ and restrict the possible completion times to the set E of endpoints of the subintervals. This reduces the number of values for possible completion times to $nZ_{UB}/(\varepsilon Z_{LB}) \leq n^2/\varepsilon$, for some lower bound on the optimal value $Z_{LB} \geq p_{\max}$. Now we run algorithm DP' which is a slightly modified variant of algorithm DP in which we round the completion times in a state to the nearest value in E. This restriction increases an optimal solution value of DP by at most $\varepsilon Z_{LB}/n$ per task. Thus, DP' finds a schedule with a makespan that exceeds the optimal makespan found by DP by at most εZ_{LB} time units. We skip further details and refer to [19] for an overview of techniques for obtaining FPTASs.

Theorem 2. *There exists an FPTAS for scheduling malleable tasks with precedence constraints of bounded width with running time.*

In case a particular allotment is given, the number of states which need to be considered obviously reduces significantly. Here, the running time of DP drops to $\mathscr{O}(n^{2\omega}Z_{UB}^{2\omega})$, and we obtain an FPTAS for scheduling parallel tasks, even if m is not polynomially bounded.

3 Optimally Solvable Special Cases

The problem of scheduling malleable tasks is clearly optimally solvable in polynomial time if either the number of processors is $m = 1$ or the width of the partial order is $\omega = 1$. Thus, we assume $m, \omega \geq 2$ from now on. We provide an efficient algorithm solving the special case $m = \omega = 2$ to optimality. It is based on the the idea of the dynamic program in Sec. 2.1 and the following observations regarding optimal (sub)solutions for special allotments. We prove NP-hardness of all remaining cases of m and ω in Sec. 4.

Observation 1. *Given a subset of tasks $J' \subseteq J$ with an allotment $\alpha_j = 2$ for all $j \in J'$, an optimal schedule for J' can be found in polynomial time if $m = \omega = 2$.*

Observation 2. *Given a subset of tasks $J' \subseteq J$ with an allotment $\alpha_j = 1$ for all $j \in J'$, an optimal schedule for J' can be found in polynomial time if $m = \omega = 2$.*

While the former is obvious, the latter can be realized by list scheduling tasks j in topological order, and setting $\sigma_j = \max_{i \prec j}\{\sigma_i + p_i(1)\}$, or $\sigma_j = 0$ if no such i exists. The makespan obtained clearly coincides with a longest chain in $(J, \prec)$ w.r.t. processing times under $(\alpha_j)_{j\in J}$, a lower bound on the optimum. The resulting schedule is also feasible since $\omega = 2$.

Theorem 3. *The problem of scheduling malleable tasks with precedence constraints of width bounded by $\omega = 2$ on $m = 2$ processors can be solved optimally in polynomial time.*

Proof. For $m = 2$, any optimal solution can be split into maximal subsequent subschedules $S_{J'}$ for subsets $J' \subseteq J$, such that in any $S_{J'}$, either $\alpha_j = 1$ for all $j \in J'$, or $\alpha_j = 2$ for all $j \in J'$. We say these subschedules are of *type one* or *type two*, respectively. Their makespans simply add up to the makespan of the whole schedule.

We now define a graph G whose nodes correspond to the (polynomially many) ideals of $(J, \prec)$ as in Sec. 2.1, such that any solution of the above structure is represented as a path in G—a shortest path in G will correspond to an optimal schedule. There is an edge from I to I' in G, if and only if $I \subset I'$. Such an edge corresponds to a subschedule $S_{J'}$ for the tasks in $J' := (I' \setminus I) \subseteq J$ as follows. If J' forms a chain in $(J, \prec)$, in particular, if $|J'| = 1$, we set $\alpha_j = \arg\min\{p_j(\alpha) \mid \alpha \in \{1, 2\}\}$ for all $j \in J'$ and define the corresponding schedule by concatenating optimal schedules of type one and two for the cases $\alpha_j = 1$ and $\alpha_j = 2$, respectively. Otherwise, we define $S_{J'}$ to be an optimal subschedule of type two. Note that by Obs. 1 and 2, all of these $S_{J'}$ can be computed in polynomial time. The lengths of edges are set to the makespans of these optimal subschedules.

Clearly, paths in G from $\emptyset$ to J correspond to feasible schedules of the same length. Furthermore, any optimal schedule is represented as a path in G, since for each of its subsolutions $S_{J'}$ of type one, there are ideals I, I' with $J' = I' \setminus I$ connected by an edge, and for each $S_{J'}$ of type two, there is a path from I to I' across ideals only differing by one task and $J' = I' \setminus I$. $\square$

Using the same idea as in the proof above, we can state an even stronger positive result for the special case of parallel tasks.

Corollary 1. *The problem of scheduling parallel tasks with precedence constraints of width $\omega = 2$ can be solved optimally in polynomial time for any number of processors m.*

The result ensues when redefining the notions of subschedules of type one and type two in the proof of Thm. 3. We adapt the correspondence between edges in the ideal graph and subschedules accordingly: Define maximal subschedules containing no concurrent tasks to be of type one, and those in which the processing interval of any tasks overlaps with that of another to be of type two. Now note that an analog of Obs. 2 remains valid.

4 Hardness Results

In this section we show that the results in the previous section are the best that we can hope for, unless P=NP. We fully settle the complexity status of scheduling malleable tasks under precedence constraints of bounded width ω by showing that the problem is NP-hard even under the monotonous penalty assumption when $\omega \geq 3, m \geq 2$ (Thm. 4) or $\omega = 2, m \geq 3$ (Thm. 5), where the former result holds for parallel tasks as well. We complement these results with proving NP-hardness for precedence constraints that form a caterpillar, i.e., a special tree.

Theorem 4. *The problem of scheduling malleable tasks with precedence constraints of width bounded by a constant $\omega \geq 3$ on any fixed number of processors $m \geq 2$ is NP-hard, even under the monotonous penalty assumption.*

Proof. We give a reduction from the NP-complete PARTITION problem, see [7], to the scheduling problem with precedence constraints that form 3 independent chains. It is easy to see that this specific problem variant can be reduced to any other problem with $\omega \geq 3$ and $m \geq 2$ by adapting the number of processors needed by tasks to achieve the processing times used in the reduction.

Consider an instance P of PARTITION: Given a set of *values* $\{v_i\}_{i=1,\dots,n}$ with $V := \sum_{i=1}^{n} v_i$, does there exist a partition A_1, A_2 of $\{1,\dots,n\}$ such that $\sum_{i \in A_1} v_i = \sum_{i \in A_2} v_i = V/2$?

We construct an instance S of our scheduling problem by borrowing ideas from a reduction for scheduling with communication delays in [18]. Instance S consists of $3n$ tasks to be scheduled on $m = 2$ processors. The precedence relations form 3 chains $\{a_i\}_{i=1,\dots,n}$, $\{b_i\}_{i=1,\dots,n}$, and $\{c_i\}_{i=1,\dots,n}$ of n tasks each. Suppose that the tasks of each chain are ordered with respect to their indices, i.e., $a_i \prec a_j$ for all $i < j$. Each node in chains $\{a_i\}_{i=1,\dots,n}$ and $\{b_i\}_{i=1,\dots,n}$ has processing time V for any processor allotment. Each task i in chain $\{c_i\}_{i=1,\dots,n}$ corresponds to element i of instance P and has processing time v_i independently of the processor allotment. Note that all processing times obey the monotonous penalty assumption.

We prove that there is a feasible schedule for instance S with makespan at most $nV + V/2$ if and only if P is a yes-instance.

Let A_1, A_2 be a partition satisfying $\sum_{i \in A_1} v_i = \sum_{i \in A_2} v_i = V/2$, and let $\pi(i)$ denote the index of the subset containing i. Consider the schedule, in which all tasks a_i are processed on the first processor, all tasks b_i on the second processor, and every task c_i on processor $\pi(i)$ placed between task a_{i-1} and a_i respectively b_{i-1} and b_i. More formally, we define start times for all tasks $\sigma_{a_i} := \sum_{k \in A_1, k \le i} v_k + (i-1)V$, $\sigma_{b_i} := \sum_{k \in A_2, k \le i} v_k + (i-1)V$, and $\sigma_{c_i} := \sum_{k \in A_{\pi(i)}, k < i} v_k + (i-1)V$. Simple calculations show that no precedence relation is violated and that the number of used processors never exceeds 2. Thus, there exists a feasible schedule with makespan $nV + V/2$.

Consider now a schedule for instance S with makespan at most $nV + V/2$. Clearly, on each processor there are exactly n tasks with processing time V; these are the tasks of chains $\{a_i\}_{i=1,\dots,n}$ and $\{b_i\}_{i=1,\dots,n}$. Thus, on every processor there are $V/2$ time units left for processing the remaining tasks of chain $\{c_i\}_{i=1,\dots,n}$. And therefore, there must exist a partition for instance P. □

The reduction in the proof of Thm. 4 adapts naturally to the problem of scheduling parallel tasks.

Corollary 2. *The problem of scheduling parallel tasks, each using exactly one processor, under precedence constraints of width bounded by a constant $\omega \ge 3$ on any fixed number of processors $m \ge 2$ is NP-hard.*

Theorem 5. *The problem of scheduling malleable tasks with precedence constraints of width $\omega = 2$, is NP-hard on any fixed number of processors $m \ge 3$, even under the monotonous penalty assumption.*

Proof. We give a reduction from an arbitrary instance of the NP-complete KNAPSACK decision problem, see [7], to the problem of scheduling 2 independent chains of tasks on $m = 3$ processors. It is easy to see that this case can be reduced to any problem setting with $m > 3$ by adapting the number of processors needed by tasks to achieve the processing times used in the reduction.

Let K denote an instance of KNAPSACK in slightly modified formulation: Given a set of *values* $\{v_i\}_{i=1,\dots,n}$, a set of *weights* $\{w_i\}_{i=1,\dots,n}$ and numbers V and W, does there exist a set $A \subseteq \{1,\dots,n\}$ with total weight at most W, i.e., $\sum_{i \in A} w_i \le W$ and a complement valued at most V, i.e., $\sum_{i \notin A} v_i \le V$? By a standard scaling argument we may assume w.l.o.g. that $V = W$.

We construct a corresponding instance S of our scheduling problem as follows: For each item $i = 1,\dots,n$, we introduce tasks j_i and $\bar{j}_i$. All j_i, and all $\bar{j}_i$ respectively, form a chain in the order of their indices. In each chain, between any two tasks, there is an additional task h_i, respectively $\bar{h}_i$, which has processing time $p_h := n \max_i\{v_i, w_i\}$ for any allotment of processors. All tasks j_i, $\bar{j}_i$ have the same processing time $p_b := 2p_h$ on 2 and 3 processors; when processed by a single processor, the processing time of task j_i increases by v_i, and the processing time of task $\bar{j}_i$ increases by w_i, respectively. These processing times clearly obey the monotonous penalty assumption.

We prove that the KNAPSACK instance K has a solution (yes-instance) if and only if there is a schedule with makespan at most $(n-1)p_h + np_b + V$.

Given a feasible solution set A for K we can construct a feasible scheduling solution for the corresponding instance S as follows: For every item $i \in A$ we allot 2 processors to task j_i and 1 processor to task $\bar{j}_i$; for every item $i \notin A$ vice versa. The remaining tasks get 1 processor. We schedule every task directly after the completion of its predecessor, i.e., $\sigma_{j_i} := (i-1)p_h + \sum_{k<i} p_{j_k}(\alpha_{j_k})$, $\sigma_{h_i} := (i-1)p_h + \sum_{k\leq i} p_{j_k}(\alpha_{j_k})$; the start times $\sigma_{\bar{j}_i}$ and $\sigma_{\bar{h}_i}$ are defined the same way. In this schedule, the processing of task j_{i-1} is completed before the processing of task $\bar{j}_i$ starts, i.e.,

$$\begin{aligned}\sigma_{j_{i-1}} + p_{j_{i-1}}(\alpha_{j_{i-1}}) = (i-2)p_h + \sum_{k\leq i-1} p_{j_k}(\alpha_{j_k}) &= (i-2)p_h + \sum_{k\leq i-1} p_b + \sum_{k\leq i-1, k\notin A} v_k \\ &\leq (i-1)p_h + \sum_{k<i} p_b \leq \sigma_{\bar{j}_i}.\end{aligned}$$

This holds analogously for $\bar{j}_{i-1}$ and j_i. Thus, there are never more than 3 processors in use and the schedule is feasible. Moreover, its makespan is $(n-1)p_h + np_b + \max\{\sum_{i\notin A} v_i, \sum_{i\in A} w_i\}$ which is at most $(n-1)p_h + np_b + V$.

For a given schedule with makespan at most $(n-1)p_h + np_b + V$ we can construct a feasible set A for the KNAPSACK instance K. Suppose there are two tasks j_i and $\bar{j}_i$ with disjoint processing intervals; w.l.o.g. let $\bar{j}_i$ be processed before j_i. Then all predecessors of $\bar{j}_i$ must be processed before $\bar{j}_i$ and all successors of j_i must be processed after task j_i. Thus, the makespan of the schedule is at least $np_b + (n-1)p_h + p_b > (n-1)p_h + np_b + V$. Thus, for any two tasks j_i or $\bar{j}_i$ at least one of them must be processed on a single processor.

Let A contain all items i that correspond to tasks $\bar{j}_i$ using a single processor. It is easy to verify that this set is a feasible solution to K. Given the makespan of the schedule, we have that $(n-1)p_h + np_b + \sum_{i\in A} w_i \leq (n-1)p_h + np_b + V$ which implies $\sum_{i\in A} w_i \leq V$. On the other hand, $(n-1)p_h + np_b + \sum_{i\notin A} v_i \leq (n-1)p_h + np_b + V$ which satisfies $\sum_{i\notin A} v_i \leq V$. □

Note that the use of malleable tasks in the reduction above is imperative—thus, the proof does not carry over to the case of parallel tasks. However, in Cor. 1 we have already shown this case to be tractable for $\omega = 2$ and any m.

We conclude our complexity investigations by showing that the scheduling problem is also NP-hard under precedence constraints that form a caterpillar, i.e., a special tree composed of a path and leaves only. This result still holds for parallel tasks or when assuming monotonous penalties. Recall that trees are a special case of series-parallel orders. This complexity status was left open in previous work presenting approximation algorithms for trees and generally series-parallel precedence constraints for malleable tasks in [12,13].

Theorem 6. *The problem of scheduling malleable tasks with precedence constraints that form a caterpillar is NP-hard for every fixed $m \geq 2$, even under the monotonous penalty assumption.*

Proof. It suffices to consider the case $m = 2$, since again for any greater number of processors, we can adapt the number of processors needed by tasks to achieve the processing times used below. We give a reduction from an arbitrary instance P of the NP-complete 3-PARTITION decision problem, see [7]. Such instance consists of natural numbers z, B and a_i with $\sum_{i=1}^{3z} a_i = zB$ and $B/4 < a_i < B/3$ for all $i = 1, \dots, 3z$, and the question is whether there exists a partition of $\{1, \dots, 3z\}$ into z disjoint sets $A_1, \dots, A_z$ such that $\sum_{i \in A_j} a_i = B$ for all $j = 1, \dots, z$.

We construct the following instance S of malleable scheduling on two processors: The set of tasks J contains two tasks j_i, k_i for all $i = 1, \dots, 3z$ and two tasks g_i, h_i for all $i = 1, \dots, z$ with the following processing times with monotonous penalties:

$$\begin{aligned} &p_{j_i}(1) = a_i, \quad && p_{j_i}(2) = a_i; \quad && p_{k_i}(1) = 2B, \quad && p_{k_i}(2) = B; \\ &p_{g_i}(1) = B, && p_{g_i}(2) = B; && p_{h_i}(1) = 2B, && p_{h_i}(2) = B. \end{aligned}$$

We introduce the following precedence constraints, which obviously form a caterpillar:

$$k_1 \prec \cdots \prec k_{3z} \prec g_1 \prec h_1 \prec \cdots \prec g_z \prec h_z \tag{1}$$

$$k_1 \prec j_1, \dots, k_{3z} \prec j_{3z} \tag{2}$$

We now argue that P is a yes-instance if and only if S has a solution with makespan at most $5zB$. Suppose there exists a partition of the a_i as required. To obtain a solution to S, we can first process all tasks k_i on two processors each. Then, we alternately schedule one task h_i on two processors, and one task g_i on one processor. In parallel to each g_i we schedule three tasks a_i belonging to the same subset A_x in the partition. This results in a schedule with makespan

$$3z \cdot p_{k_i}(2) + z \cdot p_{h_i}(2) + \sum_{i=1,\dots,z} \max\{B, \sum_{k \in A_i} a_k\} = 3zB + zB + zB = 5zB.$$

If there exists a solution to S with makespan at most $5zb$, we know that all tasks k_i and h_i must run on two processors, already accounting for time $4zB$ during which all processors are busy. The remaining tasks j_i and g_i must thus be scheduled within time zB, and this can only be achieved when all of these tasks are allotted only one processor and there are no gaps in their schedule.

Due to precedence constraints (1), tasks g_i need to be scheduled alternately with tasks h_i. Since the latter run on two processors each, the only way to avoid gaps is to divide tasks a_i into B triplets of length z each, and to process each triplet in parallel to one task g_i. □

This reduction can be adapted to suit the case of parallel tasks.

Corollary 3. *The problem of scheduling parallel tasks with precedence constraints that form a tree is NP-hard for every fixed $m \geq 2$.*

5 Scheduling on Contiguous Processors and Strip Packing

When we require each task to run on contiguous processors, an allotment $(\alpha_j)_{j \in J}$ can be interpreted as associating a rectangle of width α_j and height $p_j(\alpha_j)$ with each task $j \in J$. Optimally scheduling the tasks in J under this allotment amounts to packing these rectangles into a strip of width m of minimum length (height). Hence, malleable scheduling on contiguous processors corresponds to strip packing under discrete malleability. In the case of parallel tasks, contiguous scheduling is equivalent to strip packing with strip width m and rectangle widths in $\{1,\dots,m\}$. Clearly, any strip packing instance with rational data can be stated this way.

Note that even the problem of deciding whether a given feasible schedule is contiguous, i.e., whether it possesses a feasible contiguous mapping of tasks to processors, is NP-hard. This was shown in [5] in the context of assigning check-in counters at airports, and independently in [9]. We will argue, however, that all of our algorithms and reductions can be adapted to yield contiguous processors by construction. Thus, our results also hold for the corresponding strip packing problems. We omit the detailed proofs due to space constraints.

First, the dynamic program from Sec. 2.1 can easily be adapted to yield contiguous schedules: We merely need to keep track of the distinct processors used by every task in each state. Hence, the deduced FPTAS remains valid with a running time increased by a factor $m^{2\omega}$.

Corollary 4. *There exists an FPTAS for finding an optimal contiguous schedule of malleable tasks under precedence constraints of bounded width.*

Note that our algorithm yields a polynomial running time for classical strip packing instances with integral rectangle widths, only when the strip width m is polynomial in the input size. This assumption is quite common, see [10].

Next, observe that for $m \leq 2$, any schedule is contiguous. Consequently, Thm. 3 can be formulated for the contiguous case. Also, Cor. 1 remains valid, since for $\omega = 2$, at most two parallel tasks can be processed concurrently.

Corollary 5. *Optimal contiguous schedules on m processors under precedence constraints of width bounded by a constant, ω, can be found in polynomial time for malleable tasks with $\omega = m = 2$, and for parallel tasks with $\omega = 2$ and arbitrary m.*

Furthermore, the scheduling instances constructed in the reductions for Thms. 4 and 6 only use $m = 2$ processors. Also, the scheduling instances arising from the reduction for Thm. 5 clearly permit a contiguous schedule with the required makespan if and only if they permit any such schedule. Consequently, all of our hardness results carry over to the contiguous case.

Corollary 6. *Even when assuming monotonous penalties, contiguous scheduling of malleable and parallel tasks is NP-hard under precedence constraint which form a caterpillar on $m \geq 2$ processors, and under precedence constraints of width bounded by a constant, ω, with $\omega \geq 3$. For malleable tasks, it remains NP-hard for $\omega = 2$ when $m \geq 3$.*

Acknowledgments. We thank Rolf H. Möhring and Martin Skutella for fruitful discussions.

References

1. Augustine, J., Banerjee, S., Irani, S.: Strip packing with precedence constraints and strip packing with release times. In: Proceedings of SPAA, pp. 180–189 (2006)
2. Błażewicz, J., Liu, Z.: Scheduling multiprocessor tasks with chain constraints. European Journal of Operational Research 94(2), 231–241 (1996)

3. Dilworth, R.P.: A decomposition theorem for partially ordered sets. Annals of Mathematics 51(1), 161–166 (1950)
4. Drozdowski, M.: Scheduling multiprocessor tasks: an overview. European Journal of Operational Research 94(2), 215–230 (1996)
5. Duin, C.W., Sluis, E.V.: On the complexity of adjacent resource scheduling. Journal of Scheduling 9(1), 49–62 (2006)
6. Fulkerson, D.R.: Note on Dilworth's decomposition theorem for partially ordered sets. Proceedings of the American Mathematical Society 7(4), 701–702 (1956)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
8. Grigoriev, A., Woeginger, G.J.: Project scheduling with irregular costs: complexity, approximability, and algorithms. Acta Informatica 41(2–3), 83–97 (2004)
9. Günther, E.: Bin Scheduling: Partitionieren verformbarer Jobs mit Nebenbedingungen. Master's thesis, Technische Universität Berlin (2008) (in German)
10. Jansen, K., Thöle, R.: Approximation algorithms for scheduling parallel jobs: Breaking the approximation ratio of 2. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 234–245. Springer, Heidelberg (2008)
11. Jansen, K., Zhang, H.: An approximation algorithm for scheduling malleable tasks under general precedence constraints. ACM Transactions on Algorithms 2(3), 416–434 (2006)
12. Lepère, R., Mounié, G., Trystram, D.: An approximation algorithm for scheduling trees of malleable tasks. European Journal of Operational Research 142(2), 242–249 (2002)
13. Lepère, R., Trystram, D., Woeginger, G.J.: Approximation algorithms for scheduling malleable tasks under precedence constraints. International Journal of Foundations of Computer Science 13(4), 613–627 (2002)
14. Leung, J.-T.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC (2004)
15. Möhring, R.H.: Computationally tractable classes of ordered sets. In: Rival, I. (ed.) Algorithms and Order, pp. 105–194. Kluwer Academic Publishers, Dordrecht (1989)
16. Steiner, G.: On the complexity of dynamic programming for sequencing problems with precedence constraints. Annals of Operations Research 26, 103–123 (1990)
17. Turek, J., Wolf, J., Yu, P.: Approximate algorithms for scheduling parallelizable tasks. In: Proceedings of SPAA, pp. 323–332 (1992)
18. Verriet, J.: The complexity of scheduling graphs of bounded width subject to non-zero communication delays. Technical Report UU-CS-1997-01, Utrecht University (1997)
19. Woeginger, G.J.: When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? INFORMS Journal on Computing 12(1), 57–74 (2000)

Online Minimization Knapsack Problem

Xin Han and Kazuhisa Makino

Department of Mathematical Informatics, Graduate School of Information and Technology, University of Tokyo, Tokyo, 113-8656, Japan
hanxin.mail@gmail.com, makino@mist.i.u-tokyo.ac.jp

Abstract. In this paper, we address the online minimization knapsack problem, i. e., the items are given one by one over time and the goal is to minimize the total cost of items that covers a knapsack. We study the *removable* model, where it is allowed to remove old items from the knapsack in order to accept a new item. We obtain the following results.

(i) We propose an 8-competitive deterministic and memoryless algorithm for the problem, which contrasts to the result for the online maximization knapsack problem that no online algorithm has a bounded competitive ratio [8].
(ii) We propose a 2e-competitive randomized algorithm for the problem.
(iii) We derive a lower bound 2 for deterministic algorithms for the problem.
(iv) We propose a 1.618-competitive deterministic algorithm for the case in which each item has its size equal to its cost, and show that this is best possible.

1 Introduction

Knapsack problem is one of the most classical and studied problems in combinatorial optimization and has a lot of applications in the real world [9]. The (classical) knapsack problem is given a set of items with profits and sizes, and the capacity value of a knapsack, to maximize the total profit of selected items in the knapsack satisfying the capacity constraint. This problem is also called the *maximization* knapsack problem (Max-Knapsack). Many kinds of variants and generalizations of the knapsack problem have been investigated so far [9]. Among them, the *minimization* knapsack problem (Min-Knapsack) is one of the most natural ones (see [1,2,3,4] and [9, pp. 412-413]), that is given a set of items associated with costs and sizes, and the size of a knapsack, to minimize the total cost of selected items that cover the knapsack. Note that Min-Knapsack can be transformed into Max-Knapsack in polynomial time (and vice versa), i.e., they are polynomially equivalent. However, Min-Knapsack and Max-Knapsack exhibit relevant differences in approximation factors for the algorithms. For example, a polynomial time approximation scheme (PTAS) for Max-Knapsack does not directly lead to a PTAS for Min-Knapsack.

In this paper, we focus on the online version of problem Min-Knapsack. To our best knowledge, this is the first paper on online minimization knapsack problem. Here, "online" means that items are given over time, i.e., after a decision of rejection or acceptance is made on the current item, the next item is given, and once an item is rejected or removed, it cannot be considered again. The goal of the online minimization knapsack problem is the same as the offline version, i.e., to minimize the total cost.

Related work: It is well-known that offline Max-Knapsack and Min-Knapsack both admit a fully polynomial time approximation scheme (FPTAS) [1,4,9]. As for the online maximization knapsack problem, it was first studied on average case analysis by Marchetti-Spaccamela and Vercellis [12]. They proposed a linear time approximation algorithm such that the expected difference between the optimal and the approximation solution value is $O(\log^{3/2} n)$ under the condition that the capacity of the knapsack grows proportionally to n, the number of items. Lueker [11] further improved the expected difference to $O(\log n)$ under a fairly general condition on the distribution. Recently, Iwama and Taketomi [7] studied the problem on worst case analysis. They obtained a 1.618-competitive algorithm for the online Max-Knapsack under the *removable* condition, if each item has its size equal to its profit. Here the *removable* condition means that it is allowed to remove some items in the knapsack in order to accept a new item. They also showed that this is best possible by providing a lower bound 1.618 for this case. For the general case, Iwama and Zhang [8] showed that no algorithm for online Max-Knapsack has a bounded competitive ratio, even if the removal condition is allowed. Some generalizations of the online Max-Knapsack such as resource augmentations and Multi Knapsacks were also investigated [8,14,5].

Our results: In this paper, we study the online minimization knapsack problem. We first show that no algorithm has a bounded competitive ratio, if the *removable* condition is not allowed. Under the removable condition, we propose two *deterministic* algorithms for the online Min-Knapsack. The first one is simple and has competitive ratio $\Theta(\log \Delta)$, where Δ is the ratio of the maximum size to the minimum size in the items, and the second one has competitive ratio 8. This constant-competitive result for the online Min-Knapsack contrasts with the result for the online Max-Knapsack that no online algorithm has a bounded competitive ratio [8], which is surprising, since problems Max-Knapsack and Min-Knapsack are expected to have the same behavior from a complexity viewpoint (see Table 1).

The first algorithm is motivated by the observation: if all the items have the same size, then a simple greedy algorithm (called *Lowest Cost First* strategy) of picking items with the lowest cost first provides an optimal solution. The algorithm partitions the item set into $\lceil \log \Delta \rceil + 1$ subsets F_j by their *size*. When a new item d_t is given, the algorithm guesses the optimal value within $O(1)$ approximation factor, by using only the items in the knapsack together with the new item d_t, and for each class F_j, chooses items by Lowest Cost First strategy. Since each class F_j has cost at most $O(1)$ times the optimal value, we have an $O(\log \Delta)$-competitive algorithm, where we also provide a lower bound of the algorithm to show that it is $\Theta(\log \Delta)$-competitive.

Note that the first algorithm keeps too many extra items in the knapsack to guess the optimal value of the Min-Knapsack. In order to improve the algorithm, it has to keep items with the low total cost. However, this makes it difficult to guess the optimal value, since the item removed cannot be reused, even for guessing the optimal value. We devise the following strategy to overcome this difficulty. At each time, i) we guess the optimal value within $O(1)$ factor by repeatedly solving fractional Max-Knapsack problems to maximize the total size subject to bounded costs with respect to the items in the knapsack, together with the coming item, and ii) in order to find items to be kept, for each $j \geq 0$ we construct a subset F_j of items by solving the fractional Max-Knapsack problem subject to 2^{2-j} times the optimal cost, we keep items in $\bigcup_{j \geq 0} F_j$. We guarantee that each class F_j has cost at most 2^{2-j} times, which implies that the total cost in the knapsack is at most 8 times the optimal cost. Since the knapsack always contains a feasible solution of the Min- Knapsack problem, the procedure above leads to an 8-competitive algorithm.

We also show that no deterministic online algorithm achieves competitive ratio less than 2, and provides a *randomized* online algorithm with competitive ratio 2e ≈ 5.44. We finally consider the case in which each item has its cost equal to its size. Similarly to the online Max-Knapsack problem [7], we show that the online Min-Knapsack problem admits 1.618-competitive deterministic algorithm which matches the lower bound.

Table 1 summarizes the current status of the complexity of problems Max-Knapsack and Min-Knapsack, where the bold letters represent the results obtained in this paper.

Table 1. The current status of the complexity of problems Max-Knapsack and Min-Knapsack

			Max-Knapsack		Min-Knapsack	
			lower bound	upper bound	lower bound	upper bound
offline			FPTAS [6]		FPTAS [1]	
online	non-removable	general	unbounded [7]		**unbounded**	
		size = cost	unbounded [7]		**unbounded**	
	removable	general	unbounded [8]		**2**	**8** **2e** (randomized)
		size = cost	1.618 [7]	1.618 [7]	**1.618**	**1.618**

The rest of the paper is organized as follows. Section 2 gives definitions of the online Min-Knapsack problem, and show that the "removable" condition is necessary for the online Min-Knapsack problem. Section 3 presents algorithms for the online Min-Knapsack problem, and Section 4 gives a lower bound 2 for the online Min-Knapsack problem. Finally, in Section 5, we consider the case where each item has its cost equal to its size.

Due to space constraints, some proofs are omitted, which can be found in the full version.

2 Preliminaries

In this section, we give the definition of the online Min-Knapsack problem and show that why the removable condition is necessary for the problem.

Let us first define the offline minimization knapsack problem.

Problem Min-Knapsack

Input: A set of items $D = \{d_1, \ldots, d_n\}$ associated with cost $c : D \to \mathbb{R}_+$ and size $s : D \to \mathbb{R}_+$.

Output: A set of items $F \subseteq D$ that minimizes $\sum_{f \in F} c(f)$ subject to $\sum_{f \in F} s(f) \geq 1$.

Here we assume w.l.o.g. that the size of the knapsack is 1. For a set $U \subseteq D$, let $c(U) = \sum_{u \in U} c(u)$ and $s(U) = \sum_{u \in U} s(u)$.

In the online model, the objective is the same with the offline version. But the input is given over time. Namely, the knapsack of size 1 is known beforehand, and after a decision is made on the current item d_t associated with $c(d_t)$ and $s(d_t)$, the next one d_{t+1} is given. Once items are discarded, they cannot be used again, even for estimating an optimal value of the problem, i.e, we focus on the memoryless online algorithm. Note that this assumption is strict in the sense that most online algorithms can use the items discarded for the calculations. However we adopt this setting to tackle huge input data. Given an input sequence L and an online algorithm A, the *competitive ratio* of algorithm A is defined as follows:

$$R_A = \sup_L \frac{A(L)}{OPT(L)},$$

where $OPT(L)$ and $A(L)$ denotes the costs obtained by an optimal algorithm and the algorithm A, respectively. If $A(L)$ has no feasible solution, then we define $A(L) = +\infty$. If A is a randomized algorithm, then we have $R_A = \sup_L \frac{E[A(L)]}{OPT(L)}$.

In this paper, we consider *removable condition* for the online Min-Knapsack, i.e., it is allowed to remove or discard old items in the knapsack. It follows from the following lemma that removable condition is necessary to have a bounded competitive ratio. We leave the proof in the full version.

Lemma 1. *If at least one of the following conditions is not satisfied, then no algorithm has a bounded competitive ratio for the online Min-Knapsack problem.*

(i) *While the total size of the items given so far is smaller than* 1 *(the size of the knapsack), no item is rejected.*

(ii) *It is allowed to remove old items in the knapsack when a new item is given.*

From Lemma 1, in the subsequent sections, we consider the online Min-Knapsack problem under the removable condition.

3 Algorithms for the General Case

In this section, we present algorithms for the online Min-Knapsack problem under the removable condition. Note that in our model, once an item is removed or rejected, it cannot be used again, even for estimating the optimal value. Therefore, we have to keep items in the knapsack so that they adjust any forthcoming input sequence.

To construct an online algorithm with small competitive ratio, there are two points that we have to keep in mind: (I) keep feasible any time (i.e., the total size in the knapsack is at least 1), after the total size of the items given so far is at least 1, and (II) the total cost in the knapsack is not too far from the optimal cost, where (I) follows from (i) in Lemma 1.

3.1 A Simple Deterministic Algorithm

In this subsection, we give a simple online algorithm with a competitive ratio $\Theta(\log \Delta)$, where Δ is the ratio of the maximum size to the minimum size. The online algorithm is motivated by the observation: if all the items have the same size, then the greedy algorithm (*Lowest Cost First* selection strategy) of picking items with the lowest cost first provides an optimal solution.

For a non-negative integer t, let $D(t)$ denote the set of the first t items, i.e., $D(t) = \{d_1, \dots d_t\}$, and let $F(t)$ denote the set of items that our algorithm keeps in the knapsack after the t-round. Let t_0 be the first time when there is a feasible solution for $D(t)$, i.e., $t_0 = \min\{t \mid \sum_{i=1}^{t} s(d_i) \geq 1\}$. By Lemma 1 (i), for $t < t_0$, our algorithm keeps all the items, i.e., $F(t) = D(t)$.

Let us then consider when $t \geq t_0$. For an integer $-\infty < j < +\infty$, define

$$S_j = \{d \in D \mid 2^j < s(d) \leq 2^{j+1}\},$$

$D_j(t) = D(t) \cap S_j$ and $F_j(t) = F(t) \cap S_j$.

Our algorithm keeps $F(t)$ as the union of $\lceil \log_2 \Delta + 1 \rceil$ classes $F_j(t)$. When a new item d_t is given, the algorithm computes a guessed value $\beta(t)$ for the optimal cost $OPT(D(t))$ for the input $D(t)$ such that $\beta(t) = O(1)OPT(D(t))$, by using only the items in the knapsack $F(t-1)$ together with the new item d_t, and then for each j, we construct $F_j(t)$ from $F_j(t-1)$ by keeping the items with the total cost at most $3\beta(t)$ by the Lowest Cost First strategy.

Formally, the algorithm when $t \geq t_0$ is described as follows.

```
Algorithm A
1. $E(t) := F(t-1) \cup \{d_t\}$.
2. Guess: Compute a value $\alpha(t)$ by an approximation algorithm (e.g., [1,2])
   with $E(t)$ as the input. Set $\beta(t) := \min\{\beta(t-1), \alpha(t)\}$.
3. For each $j$, $F_j(t) := E(t) \cap S_j$ and if $c(F_j(t)) > 3\beta(t)$ then repeatedly
   remove an item with the highest cost until $c(F_j(t)) \leq 3\beta(t)$.
4. $F(t) := \bigcup_j F_j(t)$
```

Note that, for any time t, the total number of classes $F_j(t)$ needed in the algorithm is bounded by $\lceil \log_2 \Delta + 1 \rceil$. Therefore, the algorithm is $O(\log \Delta)$-competitive, if we have $s(F(t)) \geq 1$ (i.e., $F(t)$ is feasible) and $\beta(t) = O(1)OPT$ $(D(t))$ for all $t \geq t_0$. We shall show them by a series of lemmas, where the proofs for Lemmas 2, 3 and 4 are given in the full version.

Lemma 2. *Let j be an integer. At time $t \geq t_0$, we have $c(p) \geq c(q)$ for all $p \in D_j(t) - F_j(t)$ and $q \in F_j(t)$.*

Let $F^*(t)$ denote an optimal solution for an input $D(t)$ and $F^*_j(t) = F^*(t) \cap S_j$.

Lemma 3. *For a time $t \geq t_0$, assume that there is a feasible solution in $F(t)$, i.e., $s(F(t)) \geq 1$. Then, for all j, we have $c(F_j(t)) \geq 2\beta(t)$ if $F^*_j(t) \not\subseteq F_j(t)$.*

Lemma 4. *At any time $t \geq t_0$, $F(t)$ contains a feasible solution for $D(t)$ with cost at most $2OPT(D(t))$.*

Lemma 5. *Algorithm A is $O(\log \Delta)$-competitive.*

Proof. By Lemma 4, $F(t)$ contains a feasible solution for $D(t)$ with cost at most $2OPT(D(t))$.

Since (offline) Min-Knapsack problem admits a FPTAS [1],

$$\beta(t) \leq (1+\epsilon)OPT(F(t)) \leq 2(1+\epsilon)OPT(D(t))$$

for some $\epsilon > 0$ and the cost by algorithm A satisfies

$$A(D(t)) \leq 3(\lceil log_2 \Delta \rceil + 1)\beta(t),$$

and hence we have $A(D(t)) \leq O(\log \Delta)OPT(D(t))$. □

The next lemma shows that the analysis of the competitive ratio for algorithm A is tight.

Lemma 6. *Algorithm A is $\Omega(\log \Delta)$-competitive.*

Proof. To prove this lemma, we present an instance D such that $A(D) \geq \log \Delta \cdot OPT(D)$.

For $0 \leq i \leq k$, let b_i be an item with $s(b_i) = c(b_i) = 2^{-i}$, and we construct an input sequence D by $D = D^{(0)}, D^{(1)}, \ldots, D^{(k)}$, where $D^{(i)}$ is a sequence consisting of 2^i b_i's. Note that this instance has an optimal solution $F^* = \{b_0\}$ whose cost is $OPT(D) = 1$. On the other hand, algorithm A keeps all the items, and hence $A(D) = k + 1 > \log_2 \Delta \cdot OPT(D)$, where $\Delta = 2^k$ is the ratio of the largest size to the smallest size. □

By Lemmas 5 and 6, we have the following theorem.

Theorem 1. *Algorithm A is $\Theta(\log \Delta)$-competitive.*

3.2 An Improved Deterministic Algorithm

Note that the first algorithm keeps too many extra items in the knapsack to keep a feasible solution and to guess the optimal value of the Min-Knapsack. In order to obtain an $O(1)$-competitive online algorithm, for any time $t\,(\geq t_0)$, we represent the knapsack $F(t)$ as the union of subsets $F_j(t)$ $(j \geq 0)$ which satisfy the following three conditions. Note that here the definition of $F_j(t)$ is *different* from the one in the subsection 3.1.

1 A guessed value $\beta(t)$ satisfies $\beta(t) \leq r \cdot OPT(D(t))$ for some constant $r > 1$.
2 For each $j \geq 0$, $c(F_j(t)) \leq 2\beta(t)/r^j$.
3 $F(t) := \bigcup_j F_j(t)$ satisfies the feasibility, i.e., $s(F(t)) \geq 1$.

It is not difficult to see that the algorithm has constant competitive ratio if it satisfies all the conditions above. We now show how to construct such F_j's.

Let $F(t-1)$ denote a set of items in the knapsack at time $t-1$, and let $E(t) := F(t-1) \cup \{d_t\}$. For a guessed value $\beta(t)$, let $E_j(t) = \{d_i \in E(t) \mid c(d_i) \leq \beta(t)/r^j\}$ and construct $F_j(t)$ from $E_j(t)$ by repeatedly removing an item e with the highest unit cost $\frac{c(e)}{s(e)}$, until the total cost becomes at most $2\beta(t)/r^j$. Clearly this construction assures the second condition above.

To assure the first and third conditions, we first initialize $\beta(t)$ by $\beta(t) := c(E(t))$ if $t = t_0$; otherwise $\beta(t) := \beta(t-1)$. We check if $s(F_j(t)) \geq 1$ for each j. Let ℓ be the maximum number j such that $s(F_j(t)) \geq 1$. Then we have $OPT(D(t)) \leq c(F_l(t)) \leq 2\beta(t)/r^\ell$. If $OPT(D(t)) > 2\beta(t)/r^{\ell+1}$ holds in addition, then $2\beta(t)/r^\ell$ is a good guessed value for $OPT(D(t))$. However, in general this is not true, since some item in $D(t)$ has been already discarded before round t, and hence $2\beta(t)/r^\ell$ may not be a good guessed value for $OPT(D(t))$. In order to overcome this difficulty, we solve the following (offline) fractional maximization knapsack problem for each class $F_j(t)$.

$$\begin{aligned} \max \quad & \sum_{f \in F_j(t)} s(f) \cdot x(f) \\ \text{s.t.} \quad & \sum_{f \in F_j(t)} c(f) \cdot x(f) \leq \beta(t)/r^j; \\ & 0 \leq x(f) \leq 1, \quad f \in F_j(t). \end{aligned} \tag{1}$$

Let $FKP(F_j(t), \beta(t)/r^j)$ denote the optimal value of (1), where the second argument $\beta(t)/r^j$ denotes the capacity of the knapsack. It is well-known [9] that the fractional knapsack problem can be solved by a greedy approach for $s(f)/c(f)$. Let $\ell = \max\{j \mid FKP(F_j(t), \beta(t)/r^j) \geq 1\}$. Then we can see below that $\beta(t)/r^\ell$ is a good guessed value and $F_\ell(t)$ is feasible for our problem.

Formally, the algorithm is described as follows.

```
Algorithm B for $t \geq t_0$

1. $E(t) := F(t-1) \cup \{d_t\}$. If $t = t_0$, then $\alpha(t) := c(E(t))$; otherwise
   $\alpha(t) := \beta(t-1)$.
2. For each integer $j \geq 0$, construct a class $F_j(t)$ as follows.
   2.1 Let $E_j(t) := \{d_i \in E(t) \mid c(d_i) \leq \alpha(t)/r^j\}$ where $E_j(t)$ is not
       constructed if $E_j(t) = \emptyset$.
   2.2 Construct $F_j(t)$ from $E_j(t)$ by repeatedly removing an item $e$ with
       the highest unit cost $\frac{c(e)}{s(e)}$, until the total cost becomes at most
       $2\alpha(t)/r^j$.
3. Let $\ell = \max\{j \mid FKP(F_j(t), \alpha(t)/r^j) \geq 1\}$, let $F(t) := \bigcup_{j \geq \ell} F_j(t)$ and
   $\beta(t) := \alpha(t)/r^\ell$.
```

Observe that in Step 2.1 of the algorithm $E_j(t)$ is empty for all j with $\alpha(t)/r^j < \min\{c(d) \mid d \in D(t)\}$, and we have $\alpha(t) \leq t \max\{c(d) \mid d \in D(t)\}$. Hence, the number of nonempty $E_j(t)$ is bounded by $O\left(\log \frac{t \max\{c(d) \mid d \in D(t)\}}{\min\{c(d) \mid d \in D(t)\}}\right)$.

Lemma 7. *At any time $t \geq t_0$, the index ℓ in Step 3 must exist.*

Proof. We prove this lemma by induction on t. When $t = t_0$, we have $\alpha(t) = c(E(t))\,(= c(D(t)))$ and $E_0(t) = D(t)$. After Step 2.2, we have $F_0(t) = E_0(t)$, since $c(E_0(t)) = \alpha(t) < 2\alpha(t)$. We also have $FKP(F_0(t), \alpha(t)) = s(D(t)) \geq 1$, where the last inequality follows from the definition of t_0. Therefore, the lemma holds for $t = t_0$.

Assume that the lemma holds for time $t = t_1 (\geq t_0)$, i.e., $FKP(F_0(t_1), \alpha(t_1)) \geq 1$, and consider time $t = t_1 + 1$.

At time t, if the new item d_t is not selected in $F_0(t)$ at Step 2.2, then we have $F_0(t) = F_0(t_1)$. Then by the inductive hypothesis, we have $FKP(F_0(t), \alpha(t)) \geq 1$, where we note that $\alpha(t) = \beta(t_1)$. On the other hand if d_t is selected in $F_0(t)$ at Step 2.2, we have

$$FKP(F_0(t), \alpha(t)) = FKP(F_0(t), \beta(t_1)) \geq FKP(F_0(t_1), \beta(t_1)) \geq 1,$$

where the first inequality $FKP(F_0(t), \beta(t_1)) \geq FKP(F_0(t_1), \beta(t_1))$ follows from the greedy construction of $F_0(t)$ from $E_0(t)$. Hence the lemma holds for $t = t_1 + 1$. □

For a $U \subseteq D$ and a positive integer p, let $KP(U, p)$ denote the optimal value for the knapsack problem that maximize $\sum_{u \in U} s(u) \cdot x(u)$ subject to $\sum_{u \in U} c(u) \cdot x(u) \leq p$ and $x(u) \in \{0, 1\}$ for all $u \in U$. By definition, we have $KP(U, p) \leq FKP(U, p)$.

Lemma 8. *At any time $t (\geq t_0)$, we have $FKP(F_j(t), \alpha(t)/r^j) \geq KP(D(t), \alpha(t)/r^j)$ for all $j \geq 0$.*

Proof. At time $t(\geq t_0)$, let $D_j(t)$ denote the set of items with cost at most $\alpha(t)/r^j$ in $D(t)$. We shall prove that

$$FKP\Big(F_j(t), \frac{\alpha(t)}{r^j}\Big) = FKP\Big(D_j(t), \frac{\alpha(t)}{r^j}\Big).$$

Observe that: i) $\beta(t)$ and $\alpha(t)$ are non-increasing functions, ii) if $D_j(t) \geq \alpha(t)/r^j$ then the total cost of $F_j(t)$ is at least $\alpha(t)/r^j$ $(= 2\alpha(t)/r^r - \alpha(t)/r^j)$, since every item in $F_j(t)$ has cost at most $\alpha(t)/r^j$, and iii) $s(p)/c(p) \leq s(q)/c(q)$ holds for any items $p \in (D_j(t) - F_j(t))$ and $q \in F_j(t)$ such that q is contained in an optimal solution of $FKP\Big(F_j(t), \frac{\alpha(t)}{r^j}\Big)$, by the greedy construction of $F_j(t)$ in Step 2. Therefore, we have

$$FKP\Big(F_j(t), \frac{\alpha(t)}{r^j}\Big) = FKP\Big(D_j(t), \frac{\alpha(t)}{r^j}\Big).$$

This implies

$$FKP\Big(F_j(t), \frac{\alpha(t)}{r^j}\Big) = FKP\Big(D_j(t), \frac{\alpha(t)}{r^j}\Big) \geq KP\Big(D_j(t), \frac{\alpha(t)}{r^j}\Big) = KP\Big(D(t), \frac{\alpha(t)}{r^j}\Big).$$

□

Lemma 9. *At any time $t\,(\geq t_0)$ $\beta(t)$ satisfies $\beta(t) < r \cdot OPT(D(t))$.*

Proof. Assume this lemma does not hold, i.e., $\beta(t) \geq r \cdot OPT(D(t))$. Then we have

$$FKP(F_1(t), \beta(t)/r) \geq KP(D(t), \beta(t)/r) \geq KP(D(t), OPT(D(t))) \geq 1,$$

where the first inequality follows from Lemma 8, the second one follows from assumption $\beta(t) \geq r \cdot OPT(D(t))$, and the last one holds for $t \geq t_0$. This contradicts the maximality of ℓ at Step 3. □

Theorem 2. *When $r = 2$, algorithm B is 8-competitive, i.e., $B(D(t)) \leq 8OPT$ $(D(t))$ for any $t \geq t_0$.*

Proof. By Lemma 7 and the definition of $F_0(t)$, we have

$$s(F_0(t)) \geq FKP(F_0(t), \beta(t)) \geq 1,$$

i.e., $F_0(t)$ is a feasible solution for the Min-Knapsack with the input $D(t)$, and hence $F(t) = \bigcup_{j\geq 0} F_j(t)$ is also feasible. The total cost in the knapsack $F(t)$ satisfies

$$c(F(t)) \leq 2\beta(t)\sum_{j=0} r^{-j} < \frac{2r\beta(t)}{(r-1)} \leq \frac{2r^2}{r-1}OPT(D(t)),$$

where the last inequality follows from Lemma 9. Since $\frac{2r^2}{r-1} = 8$ if $r = 2$, this completes the proof. □

3.3 A Randomized Algorithm

Observe that the worst case of algorithm B is when the optimal cost $OPT(D(t))$ is sufficiently close to $\beta(t)/r$. We find that a randomized technique in [10] can foil the worst case. Namely, let ξ be a random variable uniformly distributed in $[0, 1)$. Then the competitive ratio can be improved if algorithm B uses $\alpha(t_0) = r^{\xi}c(E(t_0))$ instead of $\alpha(t_0) = c(E(t_0))$ in Step 1, i.e., if we shift $\alpha(t_0)$ by multiplying a factor r^{ξ}. Let us call this randomized algorithm RB.

We shall below show that $\mathrm{E}[RB(D)]/OPT(D) \leq 2e$ for all D against an oblivious adversary[13].

Theorem 3. *For $r = \mathrm{e}$, algorithm RB is $2\mathrm{e}$-competitive.*

4 A Lower Bound on the Online Min-Knapsack Problem

In this section, we give a lower bound 2 for the competitive ratio for the online Min-Knapsack problem. The main idea of our proof is given as follows. Assume that an online algorithm has competitive ratio smaller than 2. After $t \geq t_0$, if a small item with a small cost is given, the algorithm has to accept it, since otherwise the adversary can kill the algorithm by giving an item with large size and zero cost, i.e., the adversary will cause the online algorithm to have competitive ratio at least 2. However, after accepting small items, the total cost in the knapsack would be arbitrarily close to twice the total cost before accepting small items, This implies that the competitive ratio is at least 2. We leave the details of the proof in the full version .

Theorem 4. *Any deterministic algorithm for the online Min-Knapsack problem has competitive ratio at least* 2*.*

5 A Special Case Where the Cost Equals the Size

In this section, we focus on the case where every item has its cost equal to its size. We first give a lower bound 1.618 and then propose an online algorithm which matches the lower bound. The proof of Lemma 10 will be given in the full version.

Lemma 10. *If any item has its cost equal to its size, then no deterministic algorithm for the online Min-Knapsack problem has competitive ratio $r < 1+\mathrm{q}$ (≈ 1.618), where q is the golden ratio, i.e., q is the positive root for $\mathrm{q}^2 + \mathrm{q} = 1$.*

Let us then construct an online algorithm. Note that any optimal cost is at least 1, since any item has its cost equal to its size.

An item d is called *x-large*, *large*, *medium*, and *small* if $s(d) > 1 + \mathrm{q}$, $1 \leq s(d) \leq 1 + \mathrm{q}$, $\mathrm{q} < s(d) < 1$, and $0 < s(d) \leq \mathrm{q}$, respectively. Let us denote by *XL, L, M, S* the set of x-large, large, medium and small items,s respectively. In other words,

$$XL = \{d \in D \mid s(d) > 1 + \mathrm{q}\},\ L = \{d \in D \mid 1 \le s(d) \le 1 + \mathrm{q}\},$$
$$M = \{d \in D \mid \mathrm{q} < s(d) < 1\},\ S = \{d \in D \mid s(d) \le \mathrm{q}\}.$$

Similarly to the previous sections, let $D(t) = \{d_1, \dots, d_t\}$ and let $F(t)$ denote the set of items in the knapsack after the t-th round. Let t_0 be the first time when $D(t)$ has a feasible solution.

By Lemma 1, our algorithm accepts all the items before t_0, i.e., $F(t) = D(t)$. At time $t\,(\ge t_0)$, our algorithm keeps at most two medium items and at most one x-large item, i.e., $|F(t) \cap M| \le 2$ and $|F(t) \cap XL| \le 1$. If two medium items are contained in the knapsack, no x-large item is kept in the knapsack, i.e., if $|F(t) \cap M| = 2$ then $F(t) \cap XL = \emptyset$. Moreover, once we find a feasible solution U with the cost within $[1, 1+\mathrm{q}]$, then our algorithm only keeps this feasible solution in the knapsack and rejects all the forthcoming items, i.e., $F(t') = U$ for $t' \ge t$. For example, if d_t is large and $c(F(t-1)) \notin [1, 1+\mathrm{q}]$, then $F(t') = \{d_t\}$ for $t' \ge t$. Our algorithm always accepts the small items before finding a feasible solution with the cost within $[1, 1+\mathrm{q}]$. Table 2 shows three possible patterns for the number of x-large, large, medium and small items in the knapsack.

Let us now describe our algorithm.

```
Algorithm C for $t \ge t_0$

1. If $1 \le c(F(t-1)) \le 1+\mathrm{q}$, then $F(t) := F(t-1)$ and halt.
2. If $d_t \in XL$,    /* we have three cases */
   (a) If $s(F(t-1)) < 1$, then $F(t) := F(t-1) \cup \{d_t\}$.
   (b) If $|F(t-1) \cap M| = 2$, then $F(t) := F(t-1)$.
   (c) If $F(t-1) \cap XL = \{e\}$, then construct $F(t)$ from $F(t-1) \cup \{d_t\}$ by
       removing the largest x-large item $f$ (i.e., $f = d_t$ if $s(d_t) \ge s(e)$; otherwise,
       $f = e$)
3. If $d_t \in L$, then $F(t) := \{d_t\}$ and halt.    /* we have only one case */
4. If $d_t \in M$,    /* we have four cases */
   (a) if $s(d_t) + s(F(t-1) \cap S) \ge 1$ then let $F(t)$ be a feasible solution with
       cost at most $1+\mathrm{q}$.
   (b) if $|F(t-1) \cap M| = 2$, then construct $F(t)$ from $F(t-1) \cup \{d_t\}$ by removing
       the smallest medium item $f$.
   (c) if $|F(t-1) \cap M| = 1$, then $F(t) := (F(t-1) \cup \{d_t\}) \setminus XL$.
   (d) if $F(t-1) \cap M = \emptyset$, then $F(t) := F(t-1) \cup \{d_t\}$.
5. If $d_t \in S$,    /* we have three cases */
   (a) If $F(t-1) \cap M = \emptyset$ and $s(F(t-1) \cap S) + s(d_t) \ge 1$, then let $F(t)$ be a
       feasible solution with cost at most $1+\mathrm{q}$.
   (b) If $F(t-1) \cap M \ne \emptyset$ and $s(e) + s(F(t-1) \cap S) + s(d_t) \ge 1$ for some
       medium $e \in F(t-1)$, then let $F(t)$ be a feasible solution with cost at
       most $1+\mathrm{q}$.
   (c) Otherwise, $F(t) := F(t-1) \cup \{d_t\}$.
```

Table 2. Three possible patterns for the number of x-large, large, medium and small items in the knapsack

pattern	small	medium	large	x-large
1	0	0	1	0
2	≥ 0	2	0	0
3	≥ 0	≤ 1	0	≤ 1

Lemma 11. *For $U \subseteq D$, if $s(U \cap S) \geq 1$ or $s(U \cap S) + s(e) \geq 1$ for some $u \in U$, then U contains a feasible solution with the cost at most $1 + \mathrm{q}$.*

The above lemma ensures that Steps 4a, 5a, and 5b are always possible. By the same reason, Step 2 has only three cases.

Theorem 5. *Algorithm C has competitive ratio 1.618, which matches the lower bound.*

References

1. Babat, L.G.: Linear functions on the N-dimensional unit cube. Dokl. AKad. Nauk SSSR 222, 761–762 (1975) (Russian)
2. Csirik, J., Frenk, J.B.G., Labbé, M., Zhang, S.: Heuristics for the 0-1 Min-Knapsack problem. Acta Cybernetica 10(1-2), 15–20 (1991)
3. Güntzer, M.M., Jungnickel, D.: Approximate minimization algorithms for the 0/1 knapsack and subset-sum problem. Operations Research Letters 26, 55–66 (2000)
4. Gene, G., Levner, E.: Complexity of approximation algorithms for combinatorial problems: a survey. ACM SIGACT News 12(3), 52–65 (1980)
5. Horiyama, T., Iwama, K., Kawahara, J.: Finite-State Online Algorithms and Their Automated Competitive Analysis. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 71–80. Springer, Heidelberg (2006)
6. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM 22, 463–468 (1975)
7. Iwama, K., Taketomi, S.: Removable online knapsack problems. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 293–305. Springer, Heidelberg (2002)
8. Iwama, K., Zhang, G.: Optimal resource augmentations for online knapsack. In: APPROX-RANDOM 2007, pp. 180–188 (2007)
9. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Heidelberg (2004)
10. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. Information and Computation 131(1), 63–80 (1996); Preliminary version appeared in SODA (1993)
11. Lueker, G.S.: Average-case analysis of off-line and on-line knapsack problems. In: Proc. Sixth Annual ACM-SIAM SODA, pp. 179–188 (1995)
12. Marchetti-Spaccamela, A., Vercellis, C.: Stochastic on-line knapsack problems. Math. Programming 68(1, Ser. A), 73–104 (1995)
13. Motwani, R., Raghavan, P.: Randomized algorithms, ch.13 (online algorithms), Cambridge (2005)
14. Noga, J., Sarbua, V.: An online partially fractional knapsack problem. In: ISPAN 2005, pp. 108–112 (2005)

Optimization Problems in Multiple Subtree Graphs[⋆]

Danny Hermelin[1,⋆⋆] and Dror Rawitz[2]

[1] Department of Computer Science,
University of Haifa, Haifa 31905, Israel
danny@cri.haifa.ac.il

[2] Department of Electrical Engineering,
Tel-Aviv University, Tel-Aviv 69978, Israel
rawitz@eng.tau.ac.il

Abstract. We study various optimization problems in *t-subtree graphs*, the intersection graphs of t-subtrees, where a t-subtree is the union of t disjoint subtrees of some tree. This graph class generalizes both the class of chordal graphs and the class of t-interval graphs, a generalization of interval graphs that has recently been studied from a combinatorial optimization point of view. We present approximation algorithms for the Maximum Independent Set, Minimum Coloring, Minimum Vertex Cover, Minimum Dominating Set, and Maximum Clique problems in t-subtree graphs.

1 Introduction

Geometric intersection graphs are a very popular topic in algorithmic graph theory. This is mostly because of the many natural applications they model, and due to the rich combinatorial structure that comes along with most of them, allowing for numerous algorithmic techniques and frameworks. Two of the oldest and most well-studied graph classes in this area are the class of interval graphs, intersection graphs of intervals of a line, and the class of chordal graphs, intersection graphs of subtrees of a tree [1]. Many classical NP-complete problems become polynomial-time solvable in both these classes of graphs (for more details see [2] and references therein).

In [3,4], various optimization problems were considered in the class of t-interval graphs, a natural generalization of interval graphs. These are defined as intersection graphs of t-intervals, which are 1-dimensional objects formed by taking the union of t disjoint intervals. In this paper, we study the class of *t-subtree graphs* which generalizes t-interval graphs by replacing intervals with subtrees. Thus, t-subtree graphs form a hybrid between t-interval graphs and chordal graphs, and provide a natural generalization for these two graph classes.

[⋆] This work was done at the Caesarea Edmond Benjamin de Rothschild Foundation Institute (CRI), University of Haifa.

[⋆⋆] Supported by the Adams Fellowship of the Israel Academy of Sciences and Humanities.

We consider various classical optimization problems in t-subtree graphs. Given a t-subtree graph G along with its t-subtree representation $\mathcal{S}$ (a formal definition is given in Section 2), we present approximation algorithms for the following problems:

- Minimum Dominating Set: Find a minimum weight subset $\mathcal{S}' \subseteq \mathcal{S}$ such that for each t-subtree $S \in \mathcal{S}$ there is a t-subtree $S' \in \mathcal{S}'$ which intersects S.
- Maximum Independent Set: Find a maximum weight pairwise non-intersecting subset $\mathcal{S}' \subseteq \mathcal{S}$.
- Minimum Coloring: Partition $\mathcal{S}$ into the smallest number of subsets such that each subset is pairwise non-intersecting.
- Minimum Vertex Cover: Find a minimum weight subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $\mathcal{S} \setminus \mathcal{S}'$ is pairwise non-intersecting.
- Maximum Clique: Find a maximum weight pairwise intersecting subset $\mathcal{S}' \subseteq \mathcal{S}$.

Related work. Gavril [5] showed that the above optimization problems, with the exception of Minimum Dominating Set, can be solved in polynomial-time in chordal graphs. On the other hand, Minimum Dominating Set in chordal graphs was shown to be NP-hard in [6]. In t-interval graphs all problems are computationally hard. Griggs and West [7] showed that the class of graphs with maximum degree Δ are $\lceil(\Delta + 1)/2\rceil$-interval graphs. It follows that Minimum Dominating Set and Minimum Vertex Cover are APX-hard, for $t \geq 2$ [8], and that Minimum Coloring is NP-hard, for $t \geq 3$ [9]. Maximum Clique was shown to be NP-hard for $t \geq 3$ in [4]. Bar-Yehuda et al. [3] showed that, for $t \geq 2$, Maximum Independent Set in t-interval graphs is APX-hard, and that it cannot be approximated within a factor of $O(t/\log t)$ unless P=NP. Finally, it is NP-hard to determine whether a given graph is t-interval for $t \geq 2$ [10].

Our results. In this paper we present approximation algorithms for optimization problems in t-subtree graphs. For Maximum Independent Set, Minimum Coloring, Minimum Vertex Cover, and Maximum Clique we obtain approximation ratios of $2t$, $2t$, $2 - 1/t$, and $(t^2 - t + 1)/2$, respectively, which match the approximation ratios of the corresponding algorithms for t-interval graphs in [3,4] by extending these in a natural manner. We show that Minimum Dominating Set is different in that it is already as hard to approximate as Minimum Set Cover even in chordal graphs (*i.e.* 1-subtree graphs). We therefore consider the special case where each t-subtree has ℓ leaves, and provide an ℓ^2-approximation algorithm for this case.

2 Preliminaries

All graphs in this paper are simple and undirected. As usual, we denote the vertex-set and edge-set of a given graph G by $V(G)$ and $E(G)$, respectively. For a graph G and a subset of vertices $V \subseteq V(G)$, we let $G - V$ denote the graph obtained by deleting all vertices of V from G, and by $G[V]$ the subgraph

$G - (V(G) \setminus V)$. For a vertex $v \in V(G)$, we let $N(v)$ denote the set of neighbors of v, *i.e.* $N(v) = \{u \in V : (v,u) \in E\}$, and we let $N[v] = N(v) \cup \{v\}$.

Let $\mathcal{T}$ be an infinite rooted tree with vertices denoted by Greek letters. A set of vertices $T \subseteq V(\mathcal{T})$ is a *subtree* of $\mathcal{T}$ if $\mathcal{T}[T]$ is a tree. Two subtrees T and T' *intersect*, denoted $T \cap T'$, if they share a common vertex, and otherwise they are *disjoint*. A set S of at most t subtrees of $\mathcal{T}$ is called a t-subtree over $\mathcal{T}$. Two t-subtrees S and S' *intersect*, denoted $S \cap S'$, if there is a subtree $T \in S$ which intersects a subtree $T' \in S'$, and they are *disjoint* whenever they are non-intersecting. If $\alpha \in T$ for a subtree T in a t-subtree S, we slightly abuse notation by writing $\alpha \in S$.

A family of t-subtrees $\mathcal{S}$ is a *representation* of some graph G if there exists a bijective correspondence $v \to S_v$ from the vertices of G to the t-subtrees in $\mathcal{S}$ such that $\{u,v\} \in E(G)$ if and only if $S_u \cap S_v$. In this case, G is the *intersection graph* of $\mathcal{S}$, and thus a t-subtree graph, and we write this as $G = G_{\mathcal{S}}$. We will be considering weighted *t-subtree graphs*, *i.e.* t-subtree graphs G with weight functions $w : V(G) \to \mathbb{Q}^+$.

3 Minimum Dominating Set

We begin with Minimum Dominating Set. We show that this problem is as hard as Minimum Set Cover even when $t = 1$, *i.e.* in chordal graphs. This implies that it is NP-hard to approximate Minimum Dominating Set in 1-subtree graphs within $c \ln n$, for some constant c, due to [11]. We also show that Minimum Dominating Set is NP-hard to approximate within $\ell(\mathcal{S}) - 1 - \varepsilon$, for any $\varepsilon > 0$, where $\ell(\mathcal{S})$ is the maximum total number of leaves in any t-subtree belonging to $\mathcal{S}$. On the positive side, we present an $\ell(\mathcal{S})^2$-approximation algorithm that extends the t^2-approximation algorithm for Minimum Dominating Set in t-intervals graphs. This algorithm is based on a reduction to the Minimum Path Hitting problem, and on the approximation algorithm of Parekh and Segev [12] given for this problem.

3.1 Approximation Lower Bounds

We start by showing lower bounds for the approximation factor guarantee of any polynomial-time algorithm for Minimum Dominating Set in t-subtree graphs. These are obtained by simple approximation preserving reductions for Minimum Set Cover.

Lemma 1. *There is an approximation preserving reduction from* Minimum Set Cover *to* Minimum Dominating Set *in 1-subtree (chordal) graphs.*

Proof. Let $(X, \mathcal{C})$ be a Minimum Set Cover instance, where $X = \{x_1, \ldots, x_n\}$ is a universe of n elements, and $\mathcal{C} = \{C_1, \ldots, C_m\}$ is the family of m subsets of X. We assume without loss of generality that $\bigcup_j C_j = X$. We construct a tree $\mathcal{T}$ and a family $\mathcal{S}$ of (1-)subtrees as follows. First, $\mathcal{T}$ is a star with a center denoted by α_0 and n leaves denoted $\alpha_1, \ldots, \alpha_n$. The family $\mathcal{S}$ is a family of $n+m$ subtrees, $\mathcal{S} = \{T_1, \ldots, T_{n+m}\}$, one per each element in X and set in $\mathcal{C}$. The subtree T_i, for $i \in \{1, \ldots, n\}$, contains only the ith leaf α_i of $\mathcal{T}$. The subtree T_{n+j}, for $j \in \{1, \ldots, m\}$, is the subtree induced by $\{\alpha_0\} \cup \{\alpha_i \ : \ i \in C_j\}$. Finally, let $G = G_{\mathcal{S}}$ be the intersection graph of the subtrees in $\mathcal{S}$. Clearly the construction of $\mathcal{S}$ can be carried out in polynomial-time. To complete the proof, we argue that any set cover of X corresponds to a dominating set in G of equal size, and vice-versa.

Let $\mathcal{C}' \subseteq \mathcal{C}$ be a set cover of the universe X. Then, the subset of vertices $D = \{v_{n+j} \ : \ C_j \in \mathcal{C}'\}$, with v_{n+j} the vertex corresponding to the subtree T_{n+j}, dominates all vertices in G. Indeed, all vertices v_{n+j}, $1 \leq j \leq m$, dominate each other, and if a vertex v_i that corresponds to a leaf α_i is not dominated by D, then x_i is not covered by $\mathcal{C}'$. Conversely, suppose that D is a dominating set in G. We may assume that D does not contain vertices that correspond to leaves, since each vertex that correspond to a leaf can be replaced by a vertex that correspond to a subtree containing this leaf. Furthermore, since all leaves are dominated, the set $\mathcal{C}' = \{C_j \ : \ S_{n+j} \in D\}$ is a set cover of U. □

Next, we present a reduction from the special case of Minimum Set Cover in which each element appears in at most t sets, *i.e.* the t-Minimum Set Cover problem, to Minimum Dominating Set in t-subtree graphs.

Lemma 2. *For every positive integer $t \in \mathbb{N}$, there is an approximation preserving reduction from t-Minimum Set Cover to Minimum Dominating Set in t-subtree graphs.*

Proof. Given a t-Minimum Set Cover instance $(X, \mathcal{C})$, with $X = \{x_1, \ldots, x_n\}$ and $\mathcal{C} = \{C_1, \ldots, C_m\}$, we construct a tree $\mathcal{T}$ and a family $\mathcal{F}$ of subtrees as follows. The tree $\mathcal{T}$ consists of a root α_0 adjacent to m vertex-disjoint paths $\alpha_{j_1}, \ldots, \alpha_{j_n}$, for $j \in \{1, \ldots, m\}$. For each element $C_j \in \mathcal{C}$, we designate the subtree $T_j = \{\alpha_0, \alpha_{j_1}, \ldots, \alpha_{j_n}\}$. Also, we let $T_{j_i} = \{\alpha_{j_i}\}$. Now the set of t-subtrees $\mathcal{S}$ consists of $n+m$ t-subtrees, where $S_i = \{T_{j_i} \ : \ x_i \in C_j\}$, for $i \in \{1, \ldots, n\}$, and $S_{n+j} = \{T_j\}$, for $j \in \{1, \ldots, m\}$. Observe that $\mathcal{S}$ is a t-subtree family, and that it can be constructed in polynomial-time. We argue that dominating set in $G = G_{\mathcal{S}}$, the intersection graph of $\mathcal{S}$, corresponds to a set cover of X of equal size, and vice-versa.

Let $\mathcal{C}' \subseteq \mathcal{C}$ be a set cover of X. We claim that $D = \{v_{n+j} \ : \ C_j \in \mathcal{C}'\}$ is a dominating set in G. First, observe that all vertices v_{n+j}, where $1 \leq j \leq m$, dominate each other since each S_{n+j} includes the root α_0. Also, if a vertex v_i is not dominated by D, then x_i is not covered by any set in $\mathcal{C}'$. Conversely, let D be a dominating set in G. Without loss of generality, we may assume that $v_i \notin D$ for all $1 \leq i \leq n$, since if $x_i \in C_j$ then we can replace v_i with v_{n+j} in D. Thus, since all t-subtrees that correspond to elements are dominated, the set $\mathcal{C}' = \{C_j \ : \ T_{n+j} \in D\}$ is a set cover of X. □

It is known that it is NP-hard to approximate Minimum Set Cover within a factor of $c \ln n$ for some constant c [11], and that it is NP-hard to approximate t-Minimum Set Cover within $t-1-\varepsilon$ for any $\varepsilon > 0$ [13]. Thus, the two lemmas above show:

Corollary 1. Minimum Dominating Set *in t-subtree graphs is NP-hard to approximate within*

- $c \ln n$ *for some constant c,*
- $\ell(\mathcal{S}) - 1 - \varepsilon$ *for any $\varepsilon > 0$.*

3.2 Bounded Number of Leaves

We next present an $\ell(\mathcal{S})^2$-approximation algorithm for Minimum Dominating Set in t-chordal graphs. As a first step, we show that we can consider edge intersections in the subtrees, rather than vertex intersections.

Lemma 3. *Let G be a t-chordal graph with a t-subtree representation $\mathcal{S}$ over a tree $\mathcal{T}$. Then G has an t-subtree representation $\mathcal{S}^*$ over another tree $\mathcal{T}^*$, with $(u, v) \in E(G)$ if and only if the t-subtrees corresponding to u and v in $\mathcal{S}^*$ share an edge. Furthermore, $\mathcal{S}^*$ can be computed in polynomial-time, and $\ell(\mathcal{S}^*) = \ell(\mathcal{S})$.*

Proof. We construct a t-subtree representation $\mathcal{S}^*$ over $\mathcal{T}^*$ as follows. First, $\mathcal{T}^*$ is obtain by splitting every vertex α of $\mathcal{T}$ into two adjacent vertices:

$$V(\mathcal{T}^*) = \{\alpha' \ : \ \alpha \in V(\mathcal{T})\} \cup \{\alpha'' \ : \ \alpha \in V(\mathcal{T})\}$$
$$E(\mathcal{T}^*) = \{\{\alpha'', \beta'\} \ : \ \{\alpha, \beta\} \in E(\mathcal{T})\} \cup \{\{\alpha', \alpha''\} \ : \ \alpha \in V(\mathcal{T})\}$$

Second, for every subtree T belonging to some t-subtree in $\mathcal{S}$, we define the associated subtree T^* with

$$V(T^*) = \{\alpha' \ : \ \alpha \in V(T)\} \cup \{\alpha'' \ : \ \alpha \in V(T)\} \ .$$

Each t-subtree $S = \{T_1, \ldots, T_t\}$ in $\mathcal{S}$ is then associated with the t-subtree $S^* = \{T_1^*, \ldots, T_t^*\}$. It is not hard to verify that two subtrees S_1 and S_2 contain a node α if and only if the subtrees induced by S_1^* and S_2^* contain the edge (α', α''). Clearly, $\mathcal{S}^*$ can be computed in polynomial-time, and in addition $\ell(\mathcal{S}^*) = \ell(\mathcal{S})$ by construction. □

Next, we define a more general variant of Minimum Dominating Set in t-subtree graphs. In this variant, we are given two (not necessarily disjoint) families of t-subtrees, a red family $\mathcal{R} = \{R_1, \ldots, R_n\}$ and a blue family $\mathcal{B} = \{B_1, \ldots, B_m\}$, and our goal is to find a minimum weight subset $\mathcal{R}' \subseteq \mathcal{R}$ which dominates $\mathcal{B}$, *i.e.* every t-subtree $B \in \mathcal{B}$ is intersected by some $R \in \mathcal{R}'$. (We assume that $\mathcal{R}$ dominates $\mathcal{B}$.) Notice that when $\mathcal{B} = \mathcal{R}$, we return to Minimum Dominating Set in t-chordal graphs.

The extended variant of Minimum Dominating Set in t-subtree graphs can be formulated using the following linear integer program:

$$\begin{aligned} \min \ & \sum_{R \in \mathcal{R}} w(R) x(R) \\ \text{s.t.} \ & \sum_{R \ : \ B \sqcap R} x(R) \geq 1 \quad \forall B \in \mathcal{B} \\ & x(R) \in \{0, 1\} \qquad \forall R \in \mathcal{R} \end{aligned} \tag{DS}$$

where $x(R)$ is a variable corresponding $R \in \mathcal{R}$ and is interpreted as to $x(R) = 1$ if and only if R is taken to the solution. We use $\sqcap$ instead of $\cap$ to remind the reader that we consider edge intersections, rather than vertex intersections. The linear programming relaxation of DS is obtained by replacing the integrality constraints by: $x(R) \geq 0$, for every $R \in \mathcal{R}$.

We will need the notion of descending paths and ℓ-paths in $\mathcal{T}$. Recall that $\mathcal{T}$ is rooted. A *descending path* in $\mathcal{T}$ is a path $\alpha_1, \alpha_2, \ldots, \alpha_p$, where α_i is an ancestor of α_j for $i < j$. An ℓ-path P in $\mathcal{T}$ is a collection of at most ℓ *descending paths* which are pairwise edge-disjoint. An *ℓ-path representation* of a t-subtree graph G, is a representation which consists only of ℓ-paths.

The proof of the next lemma is immediate:

Lemma 4. *If G is a t-subtree graph with a t-subtree representation $\mathcal{S}$, then there exists an ℓ-path representation of G with $\ell = \ell(\mathcal{S})$.*

For the rest of this section we design an approximation algorithm for (DS) in graphs with ℓ-path representations (where intersections are by edges). We first address the case where both $\mathcal{R}$ and $\mathcal{B}$ contain 1-paths, each consisting of a single descending path. In [12], Parekh and Segev devised a 4-approximation algorithm for the MINIMUM PATH HITTING problem using a reduction to this special case of (DS). Formally, they obtained the following result:

Lemma 5 ([12]). *If $\mathcal{R}$ and $\mathcal{B}$ consist of 1-subtrees, each of which contains a single descending path, then the LP-relaxation of (DS) has an integral optimal solution, and this solution can be computed in polynomial time.*

We next show how to use this result to obtain an approximation algorithm in the case of $\ell > 1$. Let $\mathcal{R}$ and $\mathcal{B}$ be two families of ℓ-paths. We may assume without loss of generality that all ℓ-paths in $\mathcal{R}$ and $\mathcal{B}$ contain exactly ℓ paths. Now, let x^* denote the corresponding optimal solution of the LP-relaxation of DS.

For a descending path P in a blue ℓ-path $B \in \mathcal{B}$, let $\mathcal{R}(P) \subseteq \mathcal{R}$ denote that subset of red ℓ-paths that have descending paths intersecting P. For each $B \in \mathcal{B}$, we select a unique *representative* descending path $P_B \in B$ that maximizes the expression $\sum_{R \in \mathcal{R}(P)} x^*(R)$. In other words, the representative P_B is the maximum dominated descending path of B. For a red ℓ-path $R \in \mathcal{R}$, we let P_R^i denote the ith path in R, for $1 \leq i \leq \ell$. We construct a new 1-path instance $(\mathcal{R}', \mathcal{B}', w')$ by taking the representatives of the blue ℓ-subtrees to be $\mathcal{B}'$, *i.e.* $\mathcal{B}' = \{P_B : B \in \mathcal{B}\}$, and defining $\mathcal{R}' = \left\{P_R^i : P_i \in R, R \in \mathcal{R}\right\}$ with $w'(P_R^i) = w(R)/\ell$ for $P_R^i \in \mathcal{R}'$. Next we use Lemma 5 to obtain an integral optimal solution x' of the LP-relaxation of (DS) with respect to the new instance $(\mathcal{R}', \mathcal{B}', w')$. We output the solution $\mathcal{D} \subseteq R$ defined by $\mathcal{D} = \left\{R : x'(P_R^i) = 1 \text{ for some } i\right\}$.

Lemma 6. *$\mathcal{D}$ is an ℓ^2-approximate dominating set of $\mathcal{B}$.*

Proof. First observe that $\mathcal{D}$ dominates $\mathcal{B}$, since all representatives are dominated by $\mathcal{D}$. Also, note that

$$\sum_{R \in \mathcal{D}} w(R)x(R) \leq \ell \cdot \sum_{R' \in \mathcal{R}'} w'(R')x'(R'),$$

and that this inequality is tight in case no two descending paths of the same ℓ-path appear in $\mathcal{D}$. To complete the proof of the lemma, we show that

$$\sum_{R \in \mathcal{R}} w(R)x(R) \le \ell^2 \cdot \sum_{R \in \mathcal{R}} w(r)x^*(R).$$

For this, we define a fractional solution $\bar{x}$ for the new instance $(\mathcal{R}', \mathcal{B}', w')$ by setting $\bar{x}(P_R^i) = \ell \cdot x^*(R)$ for each $P_R^i \in \mathcal{R}'$. To see that $\bar{x}$ is feasible consider any blue ℓ-interval $B \in \mathcal{B}$ and its representative P_B. By our selection of P_B, we have $\sum_{R \in \mathcal{R}(P_B)} x^*(R) \ge 1/\ell$, since otherwise x^* would not be feasible. It follows that $\sum_{R' \in \mathcal{R}', R' \sqcap B'} \bar{x}(R') \ge 1$ for each $B' \in \mathcal{B}'$. By Lemma 5, we know that the integral solution x' is an optimal fractional solution for $(\mathcal{R}', \mathcal{B}', w')$. Hence, the weight of $\bar{x}$ is at least as high as the weight of x'. It follows that

$$\sum_{R \in \mathcal{R}} w(R)x(R) \le \ell \cdot \sum_{R' \in \mathcal{R}'} w'(R')x'(R') \le \ell \cdot \sum_{R' \in \mathcal{R}'} w'(R')\bar{x}(R')$$

Furthermore,

$$\sum_{R' \in \mathcal{R}'} w'(R')\bar{x}(R') = \sum_{R \in \mathcal{R}} \sum_{P \in R} \frac{w(R)}{\ell} \cdot \ell \cdot x^*(R) = \ell \cdot \sum_{R \in \mathcal{R}} w(R) \cdot x^*(R)$$

and we are done. □

Corollary 2. Minimum Dominating Set *in t-subtree graphs can be approximated in polynomial-time within a factor of $\ell(\mathcal{S})^2$, where $\mathcal{S}$ is the representation of the input graph.*

We remark that our algorithm works for the case where $\mathcal{R}$ contains r-paths and $\mathcal{B}$ contains b-paths, and in this case the approximation ratio is $r \cdot b$. In fact, the instance that is generated by our second reduction (see Section 3.1) can be described by $r = 1$ and $b = t$, and in this case the approximation ratio of our algorithm comes close to the $\ell(\mathcal{S})$ lower bound given in Corollary 1.

4 Maximum Independent Set

We next consider Maximum Independent Set. We present a $2t$-approximation algorithm for Maximum Independent Set that is based on the fractional local-ratio $2t$-approximation algorithm for t-interval graphs from [3].

Let $G = (V, E)$ be a t-subtree graph, and let $(\mathcal{T}, \mathcal{S})$ be its t-subtree representation, with ρ the root of $\mathcal{T}$. We define the *root* of a subtree T of $\mathcal{T}$, denoted $\rho(T)$, to be the node in T that is closest to ρ in $\mathcal{T}$. (Note that $\rho(T)$ can be ρ itself.) We let $\mathrm{root}(\mathcal{S})$ denote the set of roots of subtrees that belong to t-subtrees of $\mathcal{S}$. That is, $\mathrm{root}(\mathcal{S}) = \{\rho(T) \,:\, T \in S, S \in \mathcal{S}\}$.

Lemma 7. *Let S and S' be two intersecting t-subtrees in $\mathcal{S}$. Then, there exists some vertex $\alpha \in root(\mathcal{S})$ such that $\alpha \in S$ and $\alpha \in S'$.*

Proof. If S and S' intersect, they have a pair of intersecting subtrees $T \in S$ and $T' \in S'$. For these two trees we have either $\rho(T) \in T'$ or $\rho(T') \in T$. □

Recall that, for a vertex v of G, S_v denotes the t-subtree in $\mathcal{S}$ corresponding to v, and $w(v)$ denotes the weight of v. Lemma 7 implies that it is sufficient to look for intersections at root($\mathcal{S}$). This allows us to formulate Maximum Independent Set in t-subtree graphs as the following integer program:

$$\begin{aligned} &\max \sum_v w(v) \cdot x(v) \\ &\text{s.t.} \quad \sum_{v \,:\, \alpha \in S_v} x(v) \leq 1 \quad \alpha \in \text{root}(\mathcal{S}) \\ &\qquad x(v) \in \{0,1\} \qquad \forall v \in V(G) \end{aligned} \tag{IS}$$

As usual, $x(v)$ above denotes a variable corresponding to a vertex v of G which is to be interpreted as $x(v) = 1$ if and only if the vertex v is chosen to the independent set. The linear programming relaxation of (IS) is obtained by replacing the constraints $x(v) \in \{0,1\}$ with $0 \leq x(v) \leq 1$, for every vertex v of G. Note that the integer (and linear) programming formulation for Maximum Independent Set in t-interval graphs given in [14] is identical to the one above when $\mathcal{T}$ is a path.

Given a feasible solution x to the LP-relaxation of (IS), let us call the sum $\sum_{u \in N[v]} x(u)$ the *fractional neighborhood* of a vertex v with respect to x (recall that $N[v]$ denotes the set of neighbors of v in G including v itself). The fractional local-ratio based algorithm for Maximum Independent Set in t-interval graphs given in [3] essentially works by repeatedly selecting a vertex with a minimal fractional neighborhood, and deciding whether this vertex is in the solution independent set according to the recursive solution for $G - N[v]$. They proved that if for any feasible solution x, there is always some vertex v with fractional neighborhood at most r, then their algorithm computes an r-approximate independent set. In order to extend their algorithm to t-subtree graphs, we prove the following lemma which generalizes the corresponding lemma for t-interval graphs from [3]:

Lemma 8. *Given any feasible solution x of the LP-relation of of (IS), there exists a vertex v of G with fractional neighborhood at most $2t$ with respect to x.*

Proof. In order to prove this lemma, it is enough to show that

$$\sum_v \sum_{u \in N[v]} x(v) \cdot x(u) = \sum_v x(v) \sum_{u \in N[v]} x(u) \leq 2t \cdot \sum_v x(v) .$$

If $\{u,v\} \in E$ then $u \in N[v]$ and $v \in N[u]$. Therefore, the term $x(v) \cdot x(u)$ is counted twice in the sum on the left hand side for every pair of neighboring vertices v and u. Furthermore, if $\{u,v\} \in E$ then either there exists $T \in S_v$ such that $\rho(T) \in S_u$ or there exists $T \in S_u$ such that $\rho(T) \in S_v$. Thus,

$$\sum_v \sum_{u \in N[v]} x(v) \cdot x(u) \ \leq \ 2 \cdot \sum_v \sum_{T \in S_v} \sum_{\rho(T) \in S_u} x(v) \cdot x(u) .$$

Since x is a feasible solution of P, for every v and $T \in S_v$, we get that

$$\sum_{\rho(T) \in S_u} x(v) \cdot x(u) \;=\; x(v) \cdot \sum_{\rho(T) \in S_u} x(u) \;\leq\; x(v) \;.$$

Therefore,

$$\sum_v \sum_{u \in N[v]} x(v) \cdot x(u) \;\leq\; 2 \cdot \sum_v \sum_{T \in S_v} x(v) \;=\; 2t \cdot \sum_v x(v) \;,$$

and we are done. □

Corollary 3. Maximum Independent Set *in t-subtree graphs can be approximated in polynomial-time within a factor of $2t$.*

5 Minimum Coloring and Vertex Cover

In this section we present approximation algorithms for Minimum Coloring and Minimum Vertex Cover. The former achieves an approximation factor of $2t$, and the latter achieves a factor of $2 - 1/t$. Both algorithms rely on the following structural lemma for t-subtree graphs, which extends the corresponding lemma for t-interval graphs from [3]:

Lemma 9. *Any t-subtree graph G with maximum clique size k can be colored in polynomial-time using at most $2t(k-1)$ colors.*

Proof. Let G be a t-subtree, and let $\mathcal{S}$ denote its t-subtree representation. Denote by G^* the intersection graph of $\bigcup \mathcal{S}$, the set of subtrees that appear in $\mathcal{S}$. Clearly, $|V(G^*)| \leq t \cdot |V(G)|$, since each vertex in G corresponds to at most t vertices in G, and $|E(G)| \leq |E(G^*)|$, since each edge in G corresponds to at least one edge in G^*. Notice that G^* is chordal and has maximum clique size at most k. Thus, $|E(G^*)| < (k-1)|V(G^*)|$, as can be seen by counting all forwards edges in a simplicial ordering of G. Therefore,

$$|E(G)| \;\leq\; |E(G^*)| \;<\; (k-1)|V(G^*)| \;\leq\; t(k-1)|V(G)| \;.$$

It follows that the average degree of G is less than $2t(k-1)$, and so the standard greedy algorithm can be used to color G with at most $2t(k-1)$ colors. □

Since the maximum clique size of a graph G is a lower bound on the number of colors used in any coloring of G, and in particular in an optimal one, we obtain:

Corollary 4. Minimum Coloring *in t-subtree graphs can be approximated in polynomial-time within a factor of $2t$.*

Let us next consider Minimum Vertex Cover. Here we propose an algorithm which consists of two stages. The first stage involves removing triangles from our input graph G by applying a technique originally introduced in [15] in order to remove short odd cycles. Using this technique, we obtain in polynomial-time a triangle-free subgraph G' of G such that any r-approximate vertex cover of G' can be easily transformed into a $\max\{r, 1.5\}$-approximate vertex cover of G. The same triangle-cleaning phase is also performed in [4] to approximate Minimum Vertex Cover in t-interval graphs.

The second stage consists of using the algorithm of Hochbaum [16] which gives a factor of $(2 - 2/c)$ for Minimum Vertex Cover in graphs that can be colored in polynomial-time with c colors. Combining this with Lemma 9, we get an algorithm that computes in polynomial-time a $(2 - 1/t)$-approximate vertex cover for the triangle-free subgraph G' produced in the first stage. As mentioned above, this vertex cover can be transformed into a vertex cover of G which is $(2 - 1/t)$-approximate for $t \geq 2$.

Corollary 5. Minimum Vertex Cover *in t-subtree graphs can be approximated in polynomial-time within a factor of* $(2 - 1/t)$.

6 Maximum Clique

In this section we present a $(t^2 - t + 1)/2$-approximation algorithm for Maximum Clique in t-subtree graphs.

We begin with the notion of a transversal. A *transversal* of a subset $\mathcal{S}' \subseteq \mathcal{S}$ is a set of vertices $\{\alpha_1, \ldots, \alpha_\tau\} \subset V(\mathcal{T})$ such that for every $S \in \mathcal{S}'$ there is at least one $\alpha_i \in \{\alpha_1, \ldots, \alpha_\tau\}$ with $\alpha_i \in S$. Note that due to Lemma 7, we can assume that any transversal is a subset of $\mathrm{root}(\mathcal{S}')$. The *transversal number* of $\mathcal{S}'$ is the minimum size of any transversal of $\mathcal{S}$.

A pairwise intersecting subset of $\mathcal{S}$ is called a τ-clique if it has transversal number equal to τ. (Note that a τ-clique is not a clique with τ vertices.) Berger [17] proved upper bounds on the transversal number of any pairwise intersecting family of t-subtrees.

Lemma 10 ([17]). *Any pairwise intersecting subset $\mathcal{S}' \subseteq \mathcal{S}$ is a $(t^2 - t + 1)$-clique.*

As in t-interval families, the maximum weight 2-clique in $\mathcal{S}$ can be computed in polynomial-time due to a couple of simple observations.

Lemma 11. *A maximum weight 2-clique in $\mathcal{S}$ can be computed in polynomial time.*

Proof. Consider a pair of vertices $\alpha, \beta \in \mathrm{root}(\mathcal{S})$, and let

$$\mathcal{S}' = \{S \in \mathcal{S} \ : \ \alpha \in S \text{ or } \beta \in S\} \ .$$

Then the intersection graph $G_{\mathcal{S}'}$ is the complement of a bipartite graph, since both $\{S \in \mathcal{S} \ : \ \alpha \in S\}$ and $\{S \in \mathcal{S} \ : \ \beta \in S\}$ are pairwise intersecting. Since Maximum Independent Set is polynomial-time solvable in bipartite graphs, we can compute the maximum weight clique in $G_{\mathcal{S}'}$ in polynomial-time. Thus, by iterating over all $O(n^2)$ pairs of vertices in $\mathrm{root}(\mathcal{S})$, we can compute the maximum weight 2-clique in $\mathcal{S}$ in polynomial time. □

Combining both lemmas above, we obtain:

Corollary 6. Maximum Clique *in t-subtree graphs can be approximated in polynomial-time within a factor of* $(t^2 - t + 1)/2$.

References

1. Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. Journal of Combinatorial Theory, Series B 16(1), 47–56 (1974)
2. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
3. Bar-Yehuda, R., Halldórsson, M.M., Naor, J., Shachnai, H., Shapira, I.: Scheduling split intervals. SIAM Journal on Computing 36(1), 1–15 (2006)
4. Butman, A., Hermelin, D., Lewenstein, M., Rawitz, D.: Optimization problems in multiple-interval graphs. In: 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 268–277 (2007)
5. Gavril, F.: Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. SIAM Journal on Computing 1(2), 180–187 (1972)
6. Booth, K.S., Johnson, J.H.: Dominating set in chordal graphs. SIAM Journal on Computing 11(1), 191–199 (1982)
7. Griggs, J.R., West, D.B.: Extremal values of the interval number of a graph. SIAM Journal on Algebraic and Discrete Methods 1(1), 1–14 (1980)
8. Papadimitriou, C., Yannakakis, M.: Optimization, approximation, and complexity classes. Journal of Computer and Systems Sciences 43, 425–440 (1991)
9. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified np-complete graph problems. Theoretical Computer Science 1, 237–267 (1976)
10. West, D.B., Shmoys, D.B.: Recognizing graphs with fixed interval number is NP-complete. Discrete Applied Mathematics 8, 295–305 (1984)
11. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: 29th ACM Symposium on the Theory of Computing, pp. 475–484 (1997)
12. Parekh, O., Segev, D.: Path hitting in acyclic graphs. Algorithmica 52(4), 466–486 (2008)
13. Dinur, I., Guruswami, V., Khot, S., Regev, O.: A new multilayered PCP and the hardness of hypergraph vertex cover. SIAM Journal on Computing 34(5), 1129–1146 (2005)
14. Bar-Yehuda, R., Halldórsson, M.M., Naor, J., Shachnai, H., Shapira, I.: Scheduling split intervals. In: 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 732–741 (2002)
15. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. Annals of Discrete Mathematics 25, 27–46 (1985)
16. Hochbaum, D.S.: Efficient bounds for the stable set, vertex cover and set packing problems. Discrete Applied Mathematics 6, 243–254 (1983)
17. Berger, E.: KKM – A topological approach for trees. Combinatorica 25(1), 1–18 (2004)

Multi-Criteria TSP: Min and Max Combined

Bodo Manthey

University of Twente, Department of Applied Mathematics
P.O. Box 217, 7500 AE Enschede, The Netherlands
`b.manthey@utwente.nl`

Abstract. We present randomized approximation algorithms for multi-criteria traveling salesman problems (TSP), where some objective functions should be minimized while others should be maximized. For the symmetric multi-criteria TSP (STSP), we present an algorithm that computes $(2/3-\varepsilon, 4+\varepsilon)$ approximate Pareto curves. Here, the first parameter is the approximation ratio for the objectives that should be maximized, and the second parameter is the ratio for the objectives that should be minimized. For the asymmetric multi-criteria TSP (ATSP), we present an algorithm that computes $(1/2 - \varepsilon, \log_2 n + \varepsilon)$ approximate Pareto curves. In order to obtain these results, we simplify the existing approximation algorithms for multi-criteria Max-STSP and Max-ATSP. Finally, we give algorithms with improved ratios for some special cases.

1 Multi-Criteria TSP

1.1 Traveling Salesman Problems

The traveling salesman problem (TSP) is a basic problem in combinatorial optimization. An instance of *Max-TSP* is a complete graph $G = (V, E)$ with edge weights $w : E \to \mathbb{Q}_+$. The goal is to find a *Hamiltonian cycle* (also called a *tour*) of maximum weight, where the weight of a Hamiltonian cycle is the sum of its edge weights. (The weight of an arbitrary set of edges is analogously defined.) If G is undirected, then we speak of *Max-STSP* (symmetric TSP). If G is directed, we have *Max-ATSP* (asymmetric TSP). *Min-TSP* is similarly defined, but now the edge weights $d : E \to \mathbb{Q}_+$ are required to fulfil the triangle inequality: $d(u, v) \leq d(u, x) + d(x, v)$ for all $u, v, x \in V$ (without the triangle inequality, approximating the problem is impossible). The aim is to find a Hamiltonian cycle of minimum weight. *Min-STSP* is the symmetric variant, where G is undirected, while *Min-ATSP* is the asymmetric variant.

All four variants of TSP are NP-hard and APX-hard. Thus, we are in need of approximation algorithms. Christofides' algorithm [14] achieves a ratio of 3/2 for Min-STSP. Min-ATSP can be approximated with a factor of $\frac{2}{3} \cdot \log_2 n$, where n is the number of vertices of the instance [8]. The currently best approximation algorithm for Max-STSP achieves an approximation ratio of 7/9 [12], and the currently best algorithm for Max-ATSP achieves a ratio of 2/3 [9].

Cycle covers are one of the main tools for designing approximation algorithms for the TSP [3, 9, 8, 12]. A *cycle cover* of a graph is a set of vertex-disjoint cycles such that every vertex is part of exactly one cycle. Hamiltonian cycles are special cases of cycle covers that consist just of a single cycle. The general idea is to compute an initial cycle cover, and then we join the cycles to obtain a Hamiltonian cycle.

1.2 Multi-Criteria Optimization

In many optimization problems, there is more than one objective function. This is also the case for the TSP: We might want to minimize travel time, expenses, number of flight changes, etc., while a taxi driver might want to maximize his profit, or we want to maximize, for instance, our profit along the way. This gives rise to multi-criteria TSP, where Hamiltonian cycles are sought that optimize several objectives simultaneously. However, as far as we are aware, multi-criteria TSP has only been considered in a restricted setting, where either all objectives should be minimized or all objectives should be maximized. In this paper, we consider the general setting with both types of objectives at the same time.

If k objectives are to be maximized and ℓ objectives are to be minimized, then we have *k-Max-ℓ-Min-ATSP* and *k-Max-ℓ-Min-STSP*. If the number of criteria does not matter, we will also speak of *MC-ATSP* and *MC-STSP*. If $\ell = 0$ or $k = 0$, then we obtain the special cases *k-Max-ATSP* and *k-Max-STSP* as well as *ℓ-Min-ATSP* and *ℓ-Min-STSP*. Analogously, if the number of criteria is unimportant, we have *MC-Max-ATSP* and so on.

With respect to a single objective function, the notion of an optimal solution is well-defined. But if we have more than one objective function, there is no natural notion of a best choice. Instead, we have to content ourselves with trade-off solutions. The goal of multi-criteria optimization is to deal with this dilemma. In order to transfer the notion of an optimal solutions to multi-criteria optimization problems, *Pareto curves* (also known as *Pareto sets* or *efficient sets*) were introduced introduced (cf. Ehrgott [6]). A Pareto curve is a set of solutions that are potential optimal choices.

An instance of k-Max-ℓ-Min-ATSP is a directed complete graph $G = (V, E)$ with edge weights $w_1, \ldots, w_k : E \to \mathbb{Q}_+$ and $d_1, \ldots, d_\ell : E \to \mathbb{Q}_+$. The functions $w_1, \ldots, w_k$ should be maximized while $d_1, \ldots, d_\ell$ should be minimized. We call $w_1, \ldots, w_k$ the *max objectives* and $d_1, \ldots, d_\ell$ the *min objectives*. For convenience, let $w = (w_1, \ldots, w_k)$ and $d = (d_1, \ldots, d_\ell)$. Inequalities of vectors are meant component-wise.

A Hamiltonian cycle H *dominates* another Hamiltonian cycle H' if $w(H) \geq w(H')$ and $d(H) \leq d(H')$ and at least one of these inequalities is strict. This means that H is strictly preferable to H'. A *Pareto curve* of solutions contains all solutions that are not dominated by another solution. For other optimization problems, multi-criteria variants are defined analogously.

Unfortunately, Pareto curves cannot be computed efficiently in many cases: First, they are often of exponential size. Second, they are NP-hard to compute even for otherwise easy optimization problems. Third, TSP is NP-hard already with a single objective function, and optimization problems do not become easier with more objectives involved. Therefore, we have to be satisfied with approximate Pareto curves.

A set $\mathcal{P}$ of Hamiltonian cycles is called an (α, β) *approximate Pareto curve* for the instance (G, w, d) if the following holds: For every Hamiltonian cycle H', there exists an $H \in \mathcal{P}$ with $w(H) \geq \alpha w(H')$ and $d(H) \leq \beta d(H')$. We have $\alpha \leq 1$, $\beta \geq 1$, and a $(1, 1)$ approximate Pareto curve is a Pareto curve.

An algorithm is called an (α, β) *approximation algorithm* if, given G and w, it computes an (α, β) approximate Pareto curve. It is called a *randomized* (α, β) *approximation* if its success probability is at least $1/2$. This success probability can be amplified to $1 - 2^{-m}$ by executing the algorithm m times and taking the union of all sets of solutions. A *fully polynomial time approximation scheme* (FPTAS) for a multi-criteria optimization problem computes $(1 - \varepsilon, 1 + \varepsilon)$ approximate Pareto curves in time polynomial in the size of the instance and $1/\varepsilon$ for all $\varepsilon > 0$. Multi-criteria matching admits a *randomized FPTAS* [13], i. e., the algorithm succeeds in computing a $(1 - \varepsilon, 1 + \varepsilon)$ approximate Pareto curve with a probability of at least $1/2$. This randomized FPTAS yields also a randomized FPTAS for the multi-criteria cycle cover problem [11].

1.3 Previous Work

Most research on multi-criteria TSP is about heuristics for finding approximate solutions without any worst-case guarantee. We refer to Ehrgott and Gandibleux [6,7] for a comprehensive survey.

The first result concerning computing approximate Pareto curves for the TSP is due to Angel et al. [1,2], who considered Min-STSP restricted to edge weights 1 and 2. Ehrgott [5] considered a variant of MC-Min-STSP, where all objectives are encoded into a single objective by using some norm. MC-Min-STSP allows for a $(2+\varepsilon)$ approximation [11]. Bläser et al. [4] devised the first randomized approximations for MC-Max-STSP and MC-Max-ATSP. Their algorithms achieve ratios of $\frac{1}{k} - \varepsilon$ for k-Max-STSP and $\frac{1}{k+1} - \varepsilon$ for k-Max-ATSP. This has been improved to $2/3 - \varepsilon$ and $1/2 - \varepsilon$, respectively [10]. MC-Min-ATSP can be approximated with a factor of $\log_2 n + \varepsilon$ [10].

All approximation algorithms mentioned above deal with the special cases where we have either only min objectives or only max objectives. As far as we are aware, nothing is known so far about the approximability of multi-criteria TSP with both min and max objectives.

1.4 New Results

We present randomized approximation algorithms for k-Max-ℓ-Min-TSP for any $k, \ell \in \mathbb{N}$. As far as we are aware, this is the first paper that deals with approximation algorithms for multi-criteria TSP with both min and max objectives simultaneously. Our approximation algorithm for k-Max-ℓ-Min-ATSP computes $(1/2 - \varepsilon, \log_2 n + \varepsilon)$ approximate Pareto curves for any ℓ and k (Section 3). Our approximation algorithm for k-Max-ℓ-Min-STSP computes $(2/3 - \varepsilon, 4 + \varepsilon)$ approximate Pareto curves for any ℓ and k (Section 4). The running-times of our algorithms are polynomial in the input size for any fixed k, ℓ, and ε. But, since even the sizes of approximate Pareto curves are exponential in the number of objectives, it is unavoidable that the running-time is exponential in k and ℓ.

The main difficulty is that min and max objectives are different in nature: For max objectives, we have to collect as much weight as possible. If we have a substructure, i. e., a collection of paths that can be extended to a Hamiltonian cycle, then we can add any edges to actually get a Hamiltonian cycle. For min objectives, we have to be careful since adding any single heavy edge can deteriorate the approximation ratio.

The idea to deal with this difficulty is to first detect a collection of paths that have sufficient weight with respect to the max objectives. (In fact, we compute a set of collections of paths since a single collection does not suffice.) We will take care that these collections of paths are not too heavy with respect to the min objectives. After that, we connect our collections of paths to get Hamiltonian cycles. In this second step, we only pay attention to the min objectives; we already have enough weight with respect to the max objectives, and adding further edges does not decrease the weight.

In the next section, we introduce *decompositions*, which have already been used to approximate MC-Max-TSP [4,10]. Then we present our algorithms and their analyses in the subsequent sections. As a byproduct, our algorithms are simplified $1/2 - \varepsilon$ and $2/3 - \varepsilon$ approximation algorithms for MC-Max-ATSP and MC-Max-STSP, respectively. In particular, they avoid the recursion from k to $k - 1$ objectives that was used in the earlier approximation algorithms for MC-Max-TSP [10,4]. Finally, we consider some variants of the problem like combining asymmetric and symmetric objective functions (Section 5).

Due to space constraints, most proofs are omitted due to space constraints.

2 Decompositions

From now on, let $\eta_{k,\varepsilon} = \frac{\varepsilon^2}{2 \ln k} < 1$ for $\varepsilon > 0$. We assume $\varepsilon < \frac{1}{2 \ln k}$ throughout the paper. This is no restriction since the number k of max objectives is considered to be fixed. For $n \in \mathbb{N}$, let $[n] = \{1, 2, \ldots, n\}$.

We call a Hamiltonian cycle H a *ξ-heavy-weight Hamiltonian cycle* if there exists an $i \in [k]$ and an edge $e \in H$ such that $w_i(e) > \xi w(H)$. In this case, e is called a *ξ-heavy-weight edge* of H. If ξ is clear from the context, we also speak simply of a heavy-weight Hamiltonian cycle and a heavy-weight edge. Vice versa, H is a *ξ-light-weight Hamiltonian cycle* if it is not ξ-heavy-weight. Light-weight and heavy-weight cycle covers as well as heavy-weight edges of cycle covers are defined analogously.

A *decomposition* of a cycle cover C is a set $P \subseteq C$ of edges that consists solely of paths. The collection P of paths is obtained by removing a single edge of every cycle of C. The set P is called a γ decomposition if $w(P) \geq \gamma w(C)$. Decompositions play a crucial role in approximating MC-Max-TSP: We can add edges to a collection P of paths to get a Hamiltonian cycle. Thus, if C allows for an γ decomposition P, then we can find a Hamiltonian cycle $H \supseteq P$ with $w(H) \geq w(P) \geq \gamma w(C)$.

For our algorithms, we exploit that $(1/2-\varepsilon)$ decompositions of directed $\eta_{k,\varepsilon}$-light-weight cycle covers and $(2/3-\varepsilon)$ decompositions of undirected $\eta_{k,\varepsilon}$-light-weight cycle covers exist and can be found in polynomial time [10].

We call the procedure that finds decompositions Decompose with parameters C, w, and ε: C is a cycle cover (directed or undirected), $w = (w_1, \ldots, w_k)$ are k edge weights, and $\varepsilon > 0$. Then Decompose(C, w, ε) returns a $(1/2-\varepsilon)$- or $(2/3-\varepsilon)$-decomposition $P \subseteq C$, depending on whether C is directed or undirected, provided that C is an $\eta_{k,\varepsilon}$-light-weight cycle cover.

In addition to Decompose, we use the following existing algorithms for our algorithms: CyCo-Approx denotes the randomized FPTAS for cycle covers: on input $(G, w, d, \varepsilon, p)$, CyCo-Approx returns a $(1-\varepsilon, 1+\varepsilon)$ approximate Pareto curves of cycle covers with respect to (G, w, d) with a success probability of at least $1-p$. We use CyCo-Approx for computing both undirected and directed cycle covers. If either d or w is missing, CyCo-Approx computes a $(1-\varepsilon)$ or $(1+\varepsilon)$ approximate Pareto curve with respect to w or d, respectively.

MST-Approx denotes the deterministic FPTAS for multi-criteria spanning trees of minimum weight [13]: MST-Approx(G, d, ε) computes a $(1+\varepsilon)$ approximate Pareto curve of spanning trees of the instance (G, d).

By MinATSP-Approx, we denote the $(\log_2 n + \varepsilon)$ approximation algorithm for MC-Min-ATSP [10]: On input (G, d, ε, p), MinATSP-Approx computes a $(\log_2 n + \varepsilon)$ approximate Pareto curve for MC-Min-ATSP for the instance (G, d) with a success probability of at least $1-p$.

3 Asymmetric Multi-Criteria TSP

3.1 Preparation for MC-ATSP

In this section, we focus our attention on the max objectives w. For a graph $G = (V, E)$ and a subset $K \subseteq E$ of G's edges, we obtain G_{-K} by contracting all edges of K. Contracting a single edge (u, v) means removing all outgoing edges of u, removing all incoming edges of v, and identifying u and v. (The set K will always be such that no conflicts arise during contraction. In particular, the order in which the edges are contracted does not matter.) Analogously, for a Hamiltonian cycle H and edges K, we obtain a Hamiltonian cycle H_{-K} of G_{-K} by contracting the edges in K. We will usually have $K \subseteq H$ in this case.

If (G, w, d) is an instance for a multi-criteria TSP problem, then (G_{-K}, w, d) denotes the instance with w and d modified according to the edge contractions.

For any Hamiltonian cycle H, let $\zeta_i = \max\{w_i(e) \mid e \in H\}$ be the weight of the heaviest edge with respect to the i-th objective. Let $\zeta = \zeta(H) = (\zeta_1, \ldots, \zeta_k)$. We will distinguish between H being a light-weight cycle cover, i. e., all components of $\zeta = \zeta(H)$ are small, and H being a heavy-weight cycle cover, i. e., there is some i such that ζ_i is large. ¿From the edge weights w and ζ, we obtain new edge weights w^ζ by setting the weight of all edges that are heavier than any edge in H to 0:

$$w^\zeta(e) = \begin{cases} w(e) & \text{if } w(e) \leq \zeta \text{ and} \\ 0 & \text{if } w_i(e) > \zeta_i \text{ for some } i. \end{cases}$$

This does not affect the weight of H since all edges $e \in H$ fulfil $w(e) \leq \zeta$. The reason for this definition is the following: Assume that H is a light-weight cycle cover, and assume that we have a $(1-\varepsilon)$ approximate Pareto curve $\mathcal{C}^{\zeta(H)}$ of cycle covers with respect to $w^{\zeta(H)}$. Then $\mathcal{C}^{\zeta(H)}$ contains a light-weight cycle cover whose weight is close to H's weight. This is stated more precisely in the following lemma.

Lemma 1 (Manthey [10]). *Let $\varepsilon > 0$ be sufficiently small. Let H be an $(\eta_{k,\varepsilon/2} - (\frac{\varepsilon}{2})^3)$-light-weight Hamiltonian cycle. Let $\zeta = \zeta(H)$, and let $\mathcal{C}^\zeta$ be a $(1-\frac{\varepsilon}{2})$ approximate Pareto curve of cycle covers with respect to w^ζ.*

Then $\mathcal{C}^\zeta$ contains a cycle cover C with $w^\zeta(C) \geq (1-\frac{\varepsilon}{2}) \cdot w(H)$ and $w^\zeta(e) \leq \eta_{k,\varepsilon/2} \cdot w^\zeta(C)$ for all $e \in C$. This cycle cover C yields a decomposition $P \subseteq C$ with $w(P) \geq (1/2 - \varepsilon) \cdot w(H)$.

This is all we need so far for dealing with light-weight Hamiltonian cycles. Next, we deal with heavy-weight Hamiltonian cycles $\tilde{H}$. Of course it can happen that we somehow get a light-weight cycle cover C that approximates $\tilde{H}$, i. e., $w(C) \geq (1-\varepsilon)w(\tilde{H})$. In this case, we can apply decomposition and are done.

However, we cannot guarantee that we find such a cycle cover, not even that such a cycle cover exists. Thus, heavy-weight Hamiltonian cycles need special treatment. The idea how to deal with them is to collect a few number of heavy-weight edges. This should be done such that the following properties are met: The collection should contain a $1/2-\varepsilon$ fraction of the weight of $\tilde{H}$ with respect to some objective functions. And the rest of $\tilde{H}$, after all edges of the collection have been contracted, should be a light-weight Hamiltonian cycle. This would allow us to use decomposition for the rest of $\tilde{H}$. Let $f(k,\varepsilon) = k \cdot \left\lceil \frac{\log(\frac{1}{2}+\varepsilon)}{\log(1-\eta_{k,\varepsilon/2}+(\frac{\varepsilon}{2})^3)} \right\rceil$. Our goal is now to prove that for every Hamiltonian cycle H, there exists a set $K \subseteq H$ of cardinality at most $f(k,\varepsilon)$ such that H_{-K} is a $\left(\eta_{k,\varepsilon/2} - (\frac{\varepsilon}{2})^3\right)$-light-weight Hamiltonian cycle.

Lemma 2. *For every H and every $\varepsilon > 0$, there exists a subset $K \subseteq H$ such that $|K| \leq f(k,\varepsilon)$ and, for every $i \in [k]$, we have*

1. *$w_i(K) \geq (1/2 - \varepsilon)w_i(H)$ or*
2. *$w_i(e) \leq \left(\eta_{k,\varepsilon/2} - (\frac{\varepsilon}{2})^3\right)w_i(H_{-K})$ for all $e \in H_{-K}$.*

3.2 Approximation Algorithm for MC-ATSP

From the results of the previous section, we know that ζ and K exist such that, for every $\tilde{H}$, we will find an appropriate light-weight cycle cover that eventually yields a tour H whose weight approximates $\tilde{H}$'s weight. To actually obtain an algorithm, we have to find K and ζ. But there is only a polynomial number of possibilities for ζ and K: For all ζ and for all $i \in [k]$, we can assume that there is an edge with $w_i(e) = \zeta_i$. Thus, there are at most $O(n^2)$ choices for ζ_i, hence at most $O(n^{2k})$ in total. The cardinality of K is bounded in terms of $f(k,\varepsilon)$ as we have shown in the lemma above. For fixed k and ε, there is only

```
    $\mathcal{P}_{\text{TSP}} \leftarrow$ ATSP-APPROX$(G, w, d, \varepsilon)$
input: directed complete graph $G = (V, E)$, $w : E \to \mathbb{Q}_+^k$, $d : E \to \mathbb{Q}_+^\ell$, $\varepsilon > 0$
output: $(1/2 - \varepsilon, \log_2 n + \varepsilon)$ approximate Pareto curve $\mathcal{P}_{\text{TSP}}$ for $k$-Max-$\ell$-Min-ATSP
    with a success probability of at least $1/2$
 1: for all path covers $K \subseteq E$ with $|K| \leq f(k, \varepsilon)$ and bounds $\zeta$ do
 2:     $\mathcal{C}_{K,\zeta} \leftarrow$ CYCO-APPROX$\big(G_{-K}, w^\zeta, d, \frac{\varepsilon}{2}, \frac{1}{4n^{2k+f(k,\varepsilon)}}\big)$
 3:     for all $I \subseteq [k]$ and $C \in \mathcal{C}_{K,\zeta}$ do
 4:         if $w_I^\zeta(e) \leq \eta_{k,\varepsilon/2} \cdot w_I^\zeta(C)$ for all $e \in C$ then
 5:             $P \leftarrow$ DECOMPOSE$\big(C, w_I^\zeta, \frac{\varepsilon}{2}\big)$
 6:             let $V'$ be the start-points of paths of $P$
 7:             $\mathcal{P}'_{\text{TSP}} \leftarrow$ MINATSP-APPROX$\big(V', d, \frac{\varepsilon}{2}, \frac{1}{2^k 4n^{2k+f(k,\varepsilon)}|\mathcal{C}_{K,\zeta}|}\big)$
 8:             for all $H' \in \mathcal{P}'_{\text{TSP}}$ do
 9:                 $A \leftarrow H' \cup C$
10:                 obtain a tour $H''$ from the Eulerian set $A$ of edges with $H'' \supseteq P$
11:                 combine $H''$ and $K$ to a tour $H$; add $H$ to $\mathcal{P}_{\text{TSP}}$
```

Algorithm 1. ATSP-APPROX: Approximation algorithm for MC-ATSP

a polynomial number of subsets of cardinality at most $f(k, \varepsilon)$. We can restrict K to be a path cover, which is an acyclic set of edges such that both indegree and outdegree of each vertex is at most one. Of course, the running-time of our algorithm is exponential in the number k of max objectives. But this is unavoidable since the sizes of approximate Pareto curves can be exponential in the number of objectives. In the following, w_I for a set $I \subseteq [k]$ denotes the vector of edge weights restricted to the components in I. Instead of taking edges one-by-one as in the proof of Lemma 2, we take all edges at once. This means that we take a subset of the edges of cardinality at most $f(k, \varepsilon)$. Furthermore, we do not distinguish between light-weight and heavy-weight Hamiltonian cycles: light-weight Hamiltonian cycles are simply those for which $K = \emptyset$ works.

The min objectives remain to be taken into account. The main idea behind the algorithm is first to collect enough weight with respect to the max objectives. This gives us a collection of paths that fulfil the weight requirements for the max objectives. We have to be careful not to get too much weight with respect to the min objectives. After that we connect the paths using MINATSP-APPROX, which is the approximation algorithm for MC-Min-ATSP. Overall, we get ATSP-APPROX (Algorithm 1) and the following theorem.

Theorem 1. *For every $\varepsilon > 0$,* ATSP-APPROX *(Algorithm 1) is a randomized $(1/2 - \varepsilon, \log_2 n + \varepsilon)$ approximation algorithm for k-Max-ℓ-Min-ATSP for any $k, \ell \in \mathbb{N}$. For fixed ε, k, and ℓ, its running-time is polynomial in the input size.*

Proof. For want of space, we only analyze the approximation ratio. To do this, we assume that all randomized computations are successful. We have to show that for every Hamiltonian cycle $\tilde{H}$, there exists a Hamiltonian cycle $H \in \mathcal{P}_{\text{TSP}}$ with $w(H) \geq (1/2 - \varepsilon) w(\tilde{H})$ and $d(H) \leq (\log_2 n + \varepsilon) d(\tilde{H})$. Thus, let $\tilde{H}$ be an arbitrary Hamiltonian cycle. By Lemma 2, there exists a set $K \subseteq \tilde{H}$ of cardinality at most $f(k, \varepsilon)$ and a set $I \subseteq [k]$ with the following properties:

- For every $i \in [k] \setminus I$, we have $w_i(K) \geq (1/2 - \varepsilon) w_i(\tilde{H})$.
- For every $i \in I$ and for every edge $e \in H_{-K}$, we have $w_i(e) \leq \bigl(\eta_{k,\varepsilon/2} - (\frac{\varepsilon}{2})^3\bigr) w_i(H_{-K})$.

Let $\zeta = \zeta(H_{-K})$. According to Lemma 1, the set $\mathcal{C}_{K,\zeta}$ contains a cycle cover C with the following properties:

- C is a $\eta_{k,\varepsilon/2}$-light-weight cycle cover with respect to w_I.
- $w_i(C) \geq (1 - \frac{\varepsilon}{2}) w_i(\tilde{H})$ for every $i \in I$.
- $d(C) \leq (1 + \frac{\varepsilon}{2}) d(\tilde{H}_{-K})$.

This means that there exists a decomposition $P \subseteq C$ such that $w_i(P) \geq (1/2 - \varepsilon) w_i(\tilde{H}_{-K})$ and P consists of at most $n/2$ paths. The former follows from Lemma 1. The latter holds since H_{-K} has at most n vertices and every cycle of C consists of at least two vertices, which implies that every connected component of P consists of at least two vertices.

Thus, V' has at most $n/2$ vertices. This implies that $\mathcal{P}'_{\mathrm{TSP}}$ contains a Hamiltonian cycle H' with $d(H') \leq \bigl(\log_2(\frac{n}{2}) + \frac{\varepsilon}{2}\bigr) d(H_{-K})$.

We obtain the Hamiltonian cycle H'' of $V \setminus K$ as follows: Assume that P contains a path from u to v and H' contains an edge from u to x. Then we add the path from u to v plus the edge (v, x) to H''. We do this for all paths of P. The triangle inequality guarantees $d(v, x) \leq d(v, u) + d(u, x)$. This yields

$$\begin{aligned} d(H'') &\leq d(C) + d(H') \leq d(C) + \left(\log_2\left(\tfrac{n}{2}\right) + \tfrac{\varepsilon}{2}\right) \cdot d(H_{-K}) \\ &\leq \left(1 + \tfrac{\varepsilon}{2} + \log_2\left(\tfrac{n}{2}\right) + \tfrac{\varepsilon}{2}\right) \cdot d(H_{-K}) = (\log_2 n + \varepsilon) \cdot d(H_{-K}). \end{aligned}$$

We observe that $d(\tilde{H}) = d(K) + d(\tilde{H}_{-K})$ since $K \subseteq \tilde{H}$. Furthermore, $d(H) = d(H'') + d(K)$. Thus, $d(H) \leq (\log_2 n + \varepsilon) \cdot d(H_{-K}) + d(K) \leq (\log_2 n + \varepsilon) \cdot d(\tilde{H})$. In addition, we have $w_i(H) \geq w_i(K) \geq \left(\frac{1}{2} - \varepsilon\right) \cdot w(\tilde{H})$ for every $i \in I$ and $w_i(H) \geq w_i(P) + w_i(K) \geq \left(\frac{1}{2} - \varepsilon\right) w_i(\tilde{H}_{-K}) + w_i(K) \geq \left(\frac{1}{2} - \varepsilon\right) w_i(\tilde{H})$ for every $i \in [k] \setminus I$. This proves the approximation ratio. □

4 Symmetric Multi-Criteria TSP

Of course, ATSP-Approx works also for MC-STSP. However, this ignores d and w being symmetric. In this section, we present a $(2/3 - \varepsilon, 4 + \varepsilon)$ approximation algorithm for MC-STSP.

4.1 Preparation for MC-STSP

As we did for ATSP, we first focus our attention on the max objectives w.

For our approximation algorithm, we need counterparts of Lemmas 1 and 2. The following function g plays a similar role as f in the directed case: $g(k, \varepsilon) = k \cdot \left\lceil \frac{\log(\frac{1}{3} + \frac{\varepsilon}{3})}{\log(1 - \eta_{k,\varepsilon/3} + (\frac{\varepsilon}{3})^3)} \right\rceil$. For bounds $\zeta = (\zeta_1, \ldots, \zeta_k) \in \mathbb{Q}_+^k$, we define w^ζ in the same way as for directed graphs. We have the following counterpart of Lemma 1.

Lemma 3 (Manthey [10]). *Let $\varepsilon > 0$ be arbitrary. Let $\tilde{H}$ be an undirected Hamiltonian cycle that is $\bigl(\eta_{k,\varepsilon/3} - (\frac{\varepsilon}{3})^3\bigr)$-light. Let $\zeta = \zeta(\tilde{H})$, and let $\mathcal{C}^\zeta$ be a $\bigl(1 - \frac{\varepsilon}{3}\bigr)$ approximate Pareto curve of cycle covers with respect to w^ζ.*

Then $\mathcal{C}^\zeta$ contains a cycle cover C with $w^\zeta(C) \geq \bigl(1 - \frac{\varepsilon}{3}\bigr)w(\tilde{H})$ and $w^\zeta(e) \leq \eta_{k,\varepsilon/3} w^\zeta(C)$ for all $e \in C$. This cycle cover C yields a decomposition $P \subseteq C$ with $w(P) \geq \bigl(\frac{2}{3} - \frac{2\varepsilon}{3}\bigr)w(\tilde{H})$.

However, the main difficulty when dealing with STSP is that contractions are no longer possible. If we contract an edge in the straight-forward way, we obtain a directed instance. Since we aim at better approximation ratios for STSP than we have for ATSP, something more sophisticated has to be done. Instead of taking only single edges, we take longer paths. We do not contract these paths, but set the weights of all edges incident to vertices on the path to 0. In this way, we can later remove the edges of a cycle that traverse these vertices, and then we can add the edges of the path. The problem is that we might lose the two edges at the ends of the paths; we cannot force them to be in a cycle cover in the same way. However, as the following lemma shows, we can choose the two edges at the end such that they contribute only little to the weight of the Hamiltonian cycle. Thus, we do not lose too much weight and are still able to achieve a good approximation ratio. The following lemma shows that any sufficiently long path contains an edge that is light with respect to all objectives $w_1, \ldots, w_k$.

Lemma 4 (Manthey [10]). *Let $\tilde{H}$ be a Hamiltonian cycle on n vertices, and let $e_1, \ldots, e_m$ be any m distinct edges of $\tilde{H}$. Then there exists a $z \in [m]$ such that $w(e_z) \leq \frac{k}{m} \cdot w(\tilde{H})$.*

Now let $\tilde{H}$ be a Hamiltonian cycle, and let $K \subseteq \tilde{H}$. Let $L = L(K) = \{v \in V \mid \exists e \in K : v \in e\}$ be the set of vertices incident to edges in K. Let w^{-L} be defined by $w^{-L}(e) = w(e)$ if $e \cap L = \emptyset$ and $w^{-L}(e) = 0$ if $e \cap L \neq \emptyset$. This means that the weight of edges incident to L is set to 0, which includes the edges in K. But there are more edges whose weight is affected by w^{-L}: Let

$$T = T(K) = \{e \in \tilde{H} \mid e \notin K, e \cap L(K) \neq \emptyset\}$$

be the set of edges that have exactly one endpoint in L. The weights of these edges are set to 0 in w^{-L}, but we cannot force them to be in any cycle cover as mentioned above. (They are the edges at the ends of the paths in K.) The following lemma is the undirected counterpart of Lemma 2. In particular, it takes care of the set T. This set T is only needed for the analysis and not for the algorithm.

Lemma 5. *For every Hamiltonian cycle H and every $\varepsilon > 0$, there exists a subset $K \subseteq H$ of at most $g(k, \varepsilon)$ paths, each of length at most $\frac{6k}{\varepsilon}g(k, \varepsilon)$ with the following properties: Let $L = L(K)$ and $T = T(K)$. For every $i \in [k]$, we have*

1. *$w_i(K) \geq (2/3 - \varepsilon)w_i(H)$ or*
2. *$w_i^{-L}(e) \leq \bigl(\eta_{k,\varepsilon/3} - (\frac{\varepsilon}{3})^3\bigr)w_i^{-L}(H)$ for all $e \in H$.*

Furthermore, we have $w(T) \leq \frac{2\varepsilon}{3}w(H)$.

```
$\mathcal{P}_{\text{TSP}} \leftarrow$ STSP-Approx$(G, w, d, \varepsilon)$
input: undirected complete graph $G = (V, E)$, $w : E \to \mathbb{Q}_+^k$, $d : E \to \mathbb{Q}_+^\ell$, $\varepsilon > 0$
output: $(2/3 - \varepsilon, 4 + \varepsilon)$ approximate Pareto curve $\mathcal{P}_{\text{TSP}}$ for $k$-Max-$\ell$-Min-ATSP with
    a success probability of at least $1/2$
 1: for all $K \subseteq E$ consisting of $\leq g(k, \varepsilon)$ paths of length $\leq 6kg(k, \varepsilon)$ and all $\zeta$ do
 2:     $L \leftarrow L(K)$
 3:     $\mathcal{C}_{L,\zeta} \leftarrow$ CyCo-Approx$\big(G, w^{-L,\zeta}, \frac{\varepsilon}{4}, \frac{1}{n^{2k+6kg^2(k,\varepsilon)}}\big)$
 4:     for all $I \subseteq [k]$ and $C \in \mathcal{C}_{L,\zeta}$ do
 5:         if $w_I^{-L,\zeta}(e) \leq \eta_{k,\varepsilon/3} \cdot w_I^{-L,\zeta}(C)$ for all $e \in C$ then
 6:             $P \leftarrow$ Decompose$\big(C, w_I^{-L,\zeta}, \frac{\varepsilon}{4}\big)$
 7:             remove edges of weight 0 from $P$
 8:             choose one end-point of each path of $P$ and $K$ to obtain $V'$
 9:             let $G'$ be the corresponding graph
10:             $\mathcal{T} \leftarrow$ MST-Approx$\big(G', d, \frac{\varepsilon}{4}\big)$
11:             for all $T \in \mathcal{T}$ do
12:                 combine $T$, $P$, and $K$ to a spanning tree $T'$ of $G$
13:                 duplicate each edge of $T'$ to get an Eulerian graph $T''$
14:                 traverse $T''$, take shortcuts to get a tour $H \supseteq P \cup K$; add $H$ to $\mathcal{P}_{\text{TSP}}$
```

Algorithm 2. STSP-Approx: Approximation algorithm for MC-STSP

4.2 Approximation Algorithm for MC-STSP

The main difficulty in getting approximation ratios for MC-STSP is threefold: First, we have to be more careful than for MC-ATSP since contractions are impossible. When inserting the edges of the set K, we have to take into account two points: First, we need all edges of K since we need the weight for the max objectives. Second, we cannot afford to add arbitrary edges to build a Hamiltonian cycle since this might add too much weight with respect to the min objectives. Third, concerning the approximation ratio, we will construct Eulerian graphs from which we obtain the Hamiltonian cycles by taking shortcuts. However, on the one hand, we have to make sure that none of the edges of K is removed by taking shortcuts. On the other hand, this gives us another factor of 2 in the approximation ratio. (The problem is that Christofides' algorithm for MC-Min-STSP gives us only a ratio of $2+\varepsilon$ instead of $3/2$ as it does for Min-STSP with a single objective.) We deal with these issues in the proof of the main theorem of this section. Overall, we obtain Algorithm 2 (STSP-Approx) and the following result.

Theorem 2. *For every $\varepsilon > 0$,* STSP-Approx *(Algorithm 2) is a randomized $(2/3-\varepsilon, 4+\varepsilon)$ approximation algorithm for k-Max-ℓ-Min-STSP for any $k, \ell \in \mathbb{N}$. For fixed ε, k, and ℓ, its running-time is polynomial in the input size.*

5 Variants

Since k and ℓ are usually quite small, a natural question is if the approximation ratios can be improved for particular values of k and ℓ.

Our first observation is that k-Max-1-Min-STSP allows for a $(2/3 - \varepsilon, 3.5 + \varepsilon)$ approximation: Instead of using the spanning tree heuristic in lines 10 to 13, we use Christofides' algorithm [14]. More general and along the same lines: If ℓ-Min-STSP can be approximated with a ratio of s_ℓ, then this yields a $(2/3 - \varepsilon, s_\ell + 2 + \varepsilon)$ approximation algorithm for k-Max-ℓ-Min-STSP.

Our second observation concerns ATSP: If ℓ-Min-ATSP can be approximated with a ratio of $s_\ell(n)$ on graphs with n vertices, then this yields a $(1/2 - \varepsilon, 1 + s_\ell(n/2) + \varepsilon)$ approximation algorithm for k-Max-ℓ-Min-ATSP. This follows immediately from the analysis in Section 3. In particular, for $\ell = 1$, we obtain a $(1/2 - \varepsilon, \frac{2}{3} \log_2 n + \frac{1}{3} + \varepsilon)$ approximation using the algorithm of Feige and Singh for Min-ATSP [8].

Finally, an obvious variant of multi-criteria TSP that has not been analyzed yet is a combination of ATSP and STSP: Some objectives are asymmetric, while others are symmetric. The difficulty with this variant is that, for asymmetric objectives, only cycle covers with a minimum cycle length of two can be computed efficiently. Thus, if also the symmetric objectives require cycle cover computations, which is the case for symmetric max objectives, it seems hard to get approximation ratios better than the trivial ratios that we obtain by using the ATSP algorithms for both symmetric and asymmetric objectives.

One setting, however, allows for better ratios: If the max objectives are asymmetric and the min objectives are symmetric, then a straightforward combination of ATSP-Approx (Algorithm 1) and STSP-Approx (Algorithm 2) gives a ratio of $(1/2 - \varepsilon, 4 + \varepsilon)$: We run ATSP-Approx until we have enough weight with respect to the max objectives w. Then we switch to STSP-Approx to connect the components. We do not lose any weight with respect to w by connecting the components, although the max objectives w are asymmetric.

6 Concluding Remarks

We have presented approximation algorithms for multi-criteria traveling salesman problems that have min and max objectives simultaneously. Our algorithms work for any fixed number of minimization and maximization objectives. They are randomized and have polynomial running-time. The approximation ratios obtained, $(1/2 - \varepsilon, \log_2 n + \varepsilon)$ for MC-ATSP and $(2/3 - \varepsilon, 4 + \varepsilon)$ for MC-STSP, match the approximation ratios for multi-criteria TSP with only maximization or only minimization problems, except for the Min-STSP part of MC-STSP. For this, the ratio is only $4 + \varepsilon$, compared to $2 + \varepsilon$ for MC-Min-STSP. This raises the questions whether this $4 + \varepsilon$ can be improved. More precisely: If there exists an r_ℓ approximation algorithm for ℓ-Min-STSP, does this yield a $(2/3 - \varepsilon, r_\ell)$ approximation algorithm for k-Max-ℓ-Min-STSP? So far, we only get a performance ratio of $(2/3 - \varepsilon, r_\ell + 2 + \varepsilon)$ according to Section 5.

To simplify the analysis of the approximability of multi-criteria TSP, it would be nice if any improvement for k-Max-STSP also yields an improvement for k-Max-ℓ-Min-STSP: Assume that k-Max-STSP can be approximated with a ratio of s_k and ℓ-Min-STSP can be approximated with a ratio of r_ℓ. Does this yield a (r_k, s_ℓ) approximation for k-Max-ℓ-Min-STSP? Or at least a (f_k, g_ℓ) approximation for some non-trivial functions f_k and g_ℓ that depend on r_k and s_ℓ? The same question arises for k-Max-ATSP, ℓ-Min-ATSP, and k-Max-ℓ-Min-ATSP.

Finally, we ask whether there are also faster and deterministic algorithms for multi-criteria TSP. The algorithms presented here use randomness only because no deterministic FPTAS for multi-criteria cycle covers is known. Maybe either the randomized FPTAS can be derandomized or cycle covers as an intermediate step can be avoided at all.

References

1. Angel, E., Bampis, E., Gourvés, L.: Approximating the Pareto curve with local search for the bicriteria TSP(1,2) problem. Theoret. Comput. Sci. 310(1–3), 135–146 (2004)
2. Angel, E., Bampis, E., Gourvès, L., Monnot, J.: (Non)-approximability for the multi-criteria *TSP*(1,2). In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 329–340. Springer, Heidelberg (2005)
3. Bläser, M., Manthey, B.: Approximating maximum weight cycle covers in directed graphs with weights zero and one. Algorithmica 42(2), 121–139 (2005)
4. Bläser, M., Manthey, B., Putz, O.: Approximating multi-criteria max-TSP. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 185–197. Springer, Heidelberg (2008)
5. Ehrgott, M.: Approximation algorithms for combinatorial multicriteria optimization problems. Int. Trans. Oper. Res. 7(1), 5–31 (2000)
6. Ehrgott, M.: Multicriteria Optimization. Springer, Heidelberg (2005)
7. Ehrgott, M., Gandibleux, X.: A survey and annotated bibliography of multiobjective combinatorial optimization. OR Spectrum 22(4), 425–460 (2000)
8. Feige, U., Singh, M.: Improved approximation ratios for traveling salesperson tours and paths in directed graphs. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 104–118. Springer, Heidelberg (2007)
9. Kaplan, H., Lewenstein, M., Shafrir, N., Sviridenko, M.I.: Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. J. ACM 52(4), 602–626 (2005)
10. Manthey, B.: On approximating multi-criteria TSP. In: Proc. 26th Int. Symp. on Theoretical Aspects of Computer Science (STACS), pp. 637–648 (2009)
11. Manthey, B., Ram, L.S.: Approximation algorithms for multi-criteria traveling salesman problems. Algorithmica 53(1), 69–88 (2009)
12. Paluch, K., Mucha, M., Mądry, A.: A 7/9 - approximation algorithm for the maximum traveling salesman problem. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009 and RANDOM 2009. LNCS, vol. 5687, pp. 298–311. Springer, Heidelberg (2009)
13. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: Proc. 41st Ann. IEEE Symp. on Foundations of Computer Science (FOCS), pp. 86–92. IEEE, Los Alamitos (2000)
14. Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2001)

Packet Routing: Complexity and Algorithms[⋆]

Britta Peis, Martin Skutella, and Andreas Wiese

Technische Universität Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany
{peis,skutella,wiese}@math.tu-berlin.de

Abstract. Store-and-forward packet routing belongs to the most fundamental tasks in network optimization. Limited bandwidth requires that some packets cannot move to their destination directly but need to wait at intermediate nodes on their path or take detours. In particular, for time critical applications, it is desirable to find schedules that ensure fast delivery of the packets. It is thus a natural objective to minimize the makespan, i.e., the time at which the last packet arrives at its destination. In this paper we present several new ideas and techniques that lead to novel algorithms and hardness results.

1 Introduction

In this paper we study the packet routing problem. Given a set of packets in a network originating at possibly different start vertices, we want to transfer them to their respective destination vertices. The goal is to minimize the overall makespan, that is the time when the last packet arrives at its destination. We consider the offline version of the problem in which all information about the network and the packets, in particular the start- and destination vertices, are given in advance. In our routing model, we assume store-and-forward routing. This means that every node can store arbitrarily many packets but each link (a directed or undirected edge) can be used by only one packet at a time. We study the case where the paths of the packets are fixed in advance as well as the case where their computation is part of the problem. Moreover, we distinguish between different types of underlying graphs, e.g., directed graphs, undirected graphs, planar graphs or trees.

This problem has important applications in all settings where packets need to be transferred through a network. The priority of the packets in the schedule is not immediately clear and inappropriate routing rules can lead to inefficient schedules. Delay due to packet latency is not desirable. In particular, in time-critical applications, packets need to be delivered within a certain time frame in order to work accurately. Therefore, we are interested in schedules that guarantee the packets to arrive at their respective destinations as early as possible. Finding such efficient schedules in distributed systems is a challenging task.

[⋆] This work was partially supported by Berlin Mathematical School, by DFG research center MATHEON, and by the DFG Focus Program 1307 within the project "Algorithm Engineering for Real-time Scheduling and Routing".

1.1 Packet Routing Problem

The packet routing problem is defined as follows: Let $G = (V, E)$ be a directed or undirected graph. A packet $M_i = (s_i, t_i)$ is a tuple consisting of a start vertex $s_i \in V$ and a destination vertex $t_i \in V$. Let $\mathcal{M} = \left\{M_1, M_2, M_3, ..., M_{|\mathcal{M}|}\right\}$ be a set of packets.

Then $(G, \mathcal{M})$ is an instance of the *packet routing problem with variable paths.* The problem has two parts: First, for each packet M_i we need to find a path $P_i = (s_i = v_0, v_1, \ldots, v_{\ell-1}, v_\ell = t_i)$ from s_i to t_i such that $\{v_i, v_{i+1}\} \in E$ if G is undirected and $(v_i, v_{i+1}) \in E$ if G is directed for all i with $0 \leq i \leq \ell - 1$. Assuming that it takes one timestep to send a packet along an edge we need to find a routing schedule for the packets such that

– each message M_i follows its path P_i from s_i to t_i and
– each edge is used by at most one packet at a time

We assume that time is discrete and that all packets take their steps simultaneously. The objective is to minimize the makespan, i.e., the time when the last packet has reached its destination vertex. For each packet M_i we define D_i to be the length of the shortest path from s_i to t_i, assuming that all edges have unit length. Moreover, the *dilation* D is defined by $D := \max_i D_i$. It holds that D is a lower bound on the length of an optimal schedule.

Since there are algorithms known to determine paths for routing the packets (see [28,18] or simply take shortest paths) we will also consider the packet *routing problem with fixed paths.* An instance of this problem is a tripel $(G, \mathcal{M}, \mathcal{P})$ such that G is a (directed or undirected) graph, $\mathcal{M}$ is a set of packets and $\mathcal{P}$ is a set of predefined paths. Because the paths of the packets are given in advance they do not need to be computed here. The aim is to find a schedule with the properties described above such that the makespan is minimized. For each packet M_i we define D_i to be the length of the path P_i, again assuming that all edges have unit length. Like above we define the *dilation* D by $D := \max_i D_i$. For each edge e we define C_e to be the number of packets that are routed along edge e. The *congestion* C is then defined by $C := \max_e C_e$. It holds that C and D are lower bounds on the length of an optimal schedule.

Throughout the paper we will use the notation $|S|$ for the length of a schedule S. We call a schedule *direct* if each packet is delayed only in its start vertex. For a packet routing instance I with fixed or variable paths let $OPT\,(I)$ denote a schedule with minimum makespan. For an algorithm $\mathcal{A}$ for the packet routing problem denote by $\mathcal{A}(I)$ the schedule computed by $\mathcal{A}$ for the instance I. The algorithm $\mathcal{A}$ is an α-approximation algorithm if it runs in polynomial time and for all instances I it holds that $|\mathcal{A}(I)| \leq \alpha \cdot |OPT(I)|$. We call α the *approximation ratio* or *performance ratio* of $\mathcal{A}$.

1.2 Related Work

Packet routing and related problems have been widely studied in the literature. Di Ianni [7] shows that the so-called delay routing problem is NP-hard. The proof implies that the packet routing problem is NP-hard as well. Leung et al. [21, chapter 37] study packet routing on different graph classes, including in- and out-trees. In particular, they show that for those trees the farthest-destination-first (FDF)-algorithm works optimally. Busch et al. [5] study the direct routing problem, i.e., the problem of finding the shortest direct schedule. They give complexity results and algorithms for finding direct schedules.

Mansour and Patt-Shamir [22] study greedy scheduling algorithms (algorithms that always forward a packet if they can) in the setting where the paths of all packets are shortest paths. They prove that in this setting every packet M_i reaches its destination after at most $D_i + |\mathcal{M}| - 1$ steps where D_i is the length of the path of M_i and $|\mathcal{M}|$ is the number of packets in the network. Thus, giving priority to the packets according to the lengths of their paths yields an optimal algorithm if we assume that the path-lengths are pairwise distinct.

Leighton et al. [19] show that there is always a routing schedule that finishes within $O(C + D)$ steps. In [20] Leighton et al. present an algorithm that finds such a schedule in polynomial time. However, this algorithm is not suitable for practical applications since the hidden constants are very large. There are also some local algorithms for this problem (algorithms in which each node must take the scheduling decisions for its packets without knowing the packets in the rest of the network) needing $O(C) + (\log^* |\mathcal{M}|)^{O(\log^* |\mathcal{M}|)} D + \mathit{poly}(\log |\mathcal{M}|)$ [26] and $O\left(C + D + \log^{1+\epsilon} |\mathcal{M}|\right)$ [24] steps with high probability. For the case that all paths are shortest paths, Meyer auf der Heide et al. [2] present a randomized online routing protocol which needs only $O(C + D + \log |\mathcal{M}|)$ steps with high probability. Busch, Magdon-Ismail, and Mavronicolas [4] present a bufferless routing algorithm whose length is bounded by $O\left((C + D) \cdot \log^3 (n + |\mathcal{M}|)\right)$ where n denotes the size of the network. Using the algorithm by Leighton et. al as a subroutine, Srinivasan and Teo [28] present an algorithm that solves the packet routing problem with variable paths with a constant approximation factor. This algorithm was recently improved by Koch et al. [18] for the more general message routing problem (where each message consists of several packets).

The packet routing problem is closely related to the multi-commodity flow over time problem [10,15,16]. In particular, Hall et al. [15] show that this problem is NP-hard, even in the very restricted case of series-parallel networks. We obtain the packet routing problem with variable paths if we additionally require unit edge capacities, unit transit times, and integral flow values. If there is only one start and one destination vertex then the packet routing problem is equivalent to the quickest flow problem. It can be solved optimally in polynomial time, e.g., using the Ford-Fulkerson algorithm for the maximum flow over time problem [11,12] together with a binary search framework. Using Megiddo's method of parametric search [23], Burkard, Dlaska, and Klinz [3] present a faster algorithm which solves the quickest flow problem in strongly polynomial time. Adler et al. [1] study the problem of scheduling as many packets as possible through a given network in a certain time frame. They give approximation algorithms and NP-hardness results.

For our algorithm on directed trees we need to find a path coloring for the paths of the packets. The path coloring problem is widely studied in the literature. For instance, Raghavan et al. [27] present a (3/2)-approximation algorithm for the path coloring problem on undirected trees. Erlebach et al. [9] give NP-hardness results and algorithms for the problem. In particular, they improve the algorithm by Raghavan et al. mentioned above and present a (4/3)-approximation. For coloring directed paths on bidirected trees (i.e., trees in which each edge represents two links, one in each direction) there are algorithms known which need at most $\frac{5}{3}L$ colors where L denotes the maximum load on a directed link [8]. This is tight since there are instances which actually need $\frac{5}{3}L$ colors [17]. Gargano et al. [14] investigate the problem of coloring all directed paths in a bidirected tree.

1.3 Our Contributions

We present three individual algorithms and several complexity results for the packet routing problem. If the underlying graph is a directed tree, we first show how to solve the path coloring problem optimally. This is based on ideas presented in [9,14]. Having computed such a coloring, we present a new technique which constructs a direct schedule whose length is bounded by $C + D - 1$. In comparison, the best known algorithm for computing direct schedules for packet routing on general trees guarantees a schedule of length $2C + D - 2$ [5]. (Note that the authors of [5] assume that the edges can be used in opposite directions at the same time.)

The new idea and method we employ is likely to be useful as a subroutine for packet routing on other topologies as well. Note that makespan $C + D - 1$ is a 2-approximation since C and D are both lower bounds on the optimum, but it guarantees a much better ratio if $C \ll D$ or $C \gg D$. Moreover, we show that $C + D - 1$ is the best ratio we can possibly guarantee in terms of C and D since there are instances which actually need this many steps. We show that even for the very simple case of directed trees the natural farthest-destination-first-algorithm (FDF) can yield arbitrarily large approximation factors. However, we show how to use it as a subroutine to obtain a 2-approximation algorithm for undirected trees. This guarantees a better performance ratio than the algorithm by Busch et al. [5] since $2C + D - 2$ can asymptotically as bad as a 3-approximation.

Then we present a very general condition which guarantees a direct schedule of a given time horizon T in directed graphs. Also, we present an algorithm which computes this schedule. This result is particularly substantial in the case where $T = D$ since then we can guarantee an optimal schedule. As an application, we show that if the paths of all packets are shortest paths and their lengths are pairwise different, we can compute an optimal direct schedule of length D. This improves [22] where it was shown that under this condition there exists a (not necessarily directed) schedule of this length. We would like to emphasize that in our understanding of directed graphs, each edge can be used only in the direction of the edge. For all our algorithms we show that the analysis of the respective approximation ratios is tight. The algorithms are presented in Section 2.

Then, in Section 3, we study the complexity of the packet routing problem: We show that it is NP-hard to approximate within a factor of $(6/5 - \epsilon)$, for any $\epsilon > 0$. This implies, in particular, that there is no polynomial time approximation scheme (PTAS), unless $P = NP$. We show that this still holds if we restrict the graph topology to directed trees or chain graphs with fixed paths. Then we investigate whether there can be an algorithm with an absolute error. We answer the question in the negative. We show that it is NP-hard to approximate the packet routing problem with fixed paths with an absolute error of k for any fixed $k \geq 0$. This holds even on planar graphs.

Due to space restrictions, some proofs are shortened or moved to the appendix. For full details we refer to our technical report [25].

2 Algorithms

In this section we study approximation algorithms for packet routing. For directed trees we show that the path coloring problem can be solved optimally in $O(n \log C)$ time, where n denotes the number of vertices in the graph and C the congestion of the packet routing instance. Based on this, we present an algorithm which computes a schedule whose length is bounded by $C + D - 1$ (D denotes the dilation). For undirected trees, we present a 2-approximation algorithm. It uses the farthest-destination-first-algorithm (FDF) as a subroutine. Even though the latter performs optimally on in- and out-trees [21], we show that on general directed trees it might compute arbitrarily bad schedules. Then we give a condition for the existence of a direct schedule with a given time horizon T and an algorithm which computes this schedule. In particular, if the lengths of the paths of the packets are pairwise different, all paths are shortest paths, and the graph is directed this yields an optimal direct schedule of length D. This improves [22] where for this setting the existence of a not necessarily direct schedule of that length was proven.

2.1 Approximation for Directed Trees

For directed trees we show how to construct a direct schedule of length at most $C + D - 1$ in polynomial time. The algorithm works as follows: First we find a coloring for the paths of the packets such that two paths that share an edge have different colors. We will show that the number of colors needed is exactly C. We assign to each packet the color of its path. Then we assign to each edge a time-dependent color. The idea behind this is that we transfer a packet P with color c_P along an edge $e = (u, v)$ only when e has the color c_P. We define the coloring such that for two consecutive edges $e = (u, v)$ and $e' = (v, w)$ it holds that at time t the edge e' has always the color that e had at time $t - 1$. This ensures that once a packet starts moving, it will never stop until it reaches its destination. In the sequel we describe the algorithm in detail.

Path Coloring. First we want to find a coloring for the paths such that two paths with the same color do not share an edge. Our algorithm works in two phases: in the first phase we consider each vertex v together with its adjacent vertices (this subgraph forms a star). For each of these subgraphs (together with the paths of the packets in this subgraph) we solve the path coloring problem optimally (here we use the fact that our tree is directed). Then we combine all these part-solutions and obtain a solution for the global path-coloring problem.

Phase one: Let v be a vertex and denote by T_v the subgraph induced by v and its adjacent vertices. We want to find a coloring for the paths which use edges in T_v. We reduce this problem to the edge-coloring problem on bipartite multigraphs (note that it is crucial that the edges in T_v are directed). This construction was already mentioned in [14].

Let U be the set of vertices which have outgoing edges to v, i.e., $U = \{u|\,(u,v) \in E\}$. Similarly, let W be the set of vertices having ingoing edges from v, i.e., $W = \{w|\,(v,w) \in E\}$. We construct an undirected graph B_v as follows: the set $U \cup W$ forms the set of vertices in B_v. For each path P that goes from a vertex $u \in U$ through v to a vertex $w \in W$ we introduce an edge $e_P := \{u, w\}$ in B_v. For all paths P that start in a vertex $u \in U$ and end in v we introduce a new vertex $w_P \in W$ and an edge $e_P := \{u, w_P\}$ in B_v. Similarly, for paths P that start in v and end in a vertex $w \in W$ we add a vertex v_P and introduce an edge $e_P := \{v_P, w\}$ in B_v. Thus, for the maximum degree $\Delta\,(B_v)$ of a node in B_v it holds that $\Delta\,(B_v) \leq C$. Also, it holds that two edges e_P and $e_{P'}$ share an end-vertex if and only if their corresponding paths P and P' share an edge in T_v. Thus, a valid edge-coloring for B_v implies a valid path coloring for T_v and vice versa. Moreover, from the construction it follows that B_v is bipartite. We compute a minimum edge coloring for B_v (e.g., see [6]). The number of colors needed equals the maximum node degree $\Delta\,(B_v)$ [6].

Phase two: Now we combine the found solutions for the graphs T_v one by one to obtain a global solution (a similar construction is described in [9, Lemma 2]). We start with an arbitrary vertex v and the path coloring of T_v. Now let v' be a vertex adjacent to v and consider the graph $T_{v'}$. We permute the colors of the paths in $T_{v'}$ such that the paths which use the edge (v, v') (or (v', v), respectively) have the same colors in T_v and $T_{v'}$. We iterate over the vertices by always adding a vertex that is adjacent to one of the vertices that have been considered already. Eventually, for each edge $e = (u, v)$ all paths that use e have the same color in T_u and T_v. Since T is a tree in each iteration we can find a valid permutation of the colors of the paths by using a simple greedy strategy. Since for each graph T_v we find path colorings with at most C colors, the resulting path coloring for T has C colors as well.

Time-Dependent Edge Coloring. Now we construct a time-dependent coloring $c : E \times \mathbb{N} \to \{1, 2, ..., C\}$ for the edges of T. It has the *consecutive property:* for two consecutive edges $e = (u, v)$ and $e' = (v, w)$ it holds that $c\,(e, i) = c\,(e', i + 1)$. Since our graph is a directed tree such a coloring can be found with a greedy method: Start with an arbitrary edge e and define its coloring $c\,(e, i) := i \bmod C$ for all $i \in \mathbb{N}$. Then inductively assign the colors to the remaining edges such that the consecutive property holds.

Routing Schedule. Finally, we describe the scheduling algorithm. First, we assign to each packet the color of its path. Now let M be a packet which is located on a vertex u at time t and which needs to use the edge $e = (u, v)$ next. Let c_P be the color of M. We move M along e in the first timestep t' with $t' \geq t$ and $c(e, t') = c_P$. Due to the consecutive property of the time-dependent edge coloring a packet is never delayed once it has left its start vertex. Denote by $DTREE(I)$ the resulting schedule for an instance I.

Theorem 1. *Let T be a directed tree and let $I = (T, \mathcal{M})$ be a packet routing instance. It holds that $|DTREE(I)| \leq C + D - 1$. A packet is never delayed once it has left its start vertex (direct routing). Moreover, $DTREE(I)$ can be computed in $O(n \cdot |\mathcal{M}| \cdot \log C)$.*

Proof. Since no two packets with the same color share an edge there can be at most one packet that uses an edge e at a time t. Each packet M waits in its origin vertex for at most $C - 1$ timesteps. Due to the consecutive property once it left its start vertex it moves to its destination without being delayed any further. Thus, the length of the overall makespan is bounded by $C - 1 + D$.

The edge coloring problem on bipartite multigraphs can be solved optimally in $O(m \log \Delta)$ where m denotes the number of edges in the graph and Δ the maximum node degree, see [6]. Thus, computing the optimal path coloring for one graph T_v can be done in $O(|\mathcal{M}| \cdot \log C)$ and for all graphs T_v in $O(n \cdot |\mathcal{M}| \cdot \log C)$.

Then we need to combine the colorings for the graphs T_v to a global path coloring. We say a path P *touches* a vertex v if P goes through v, starts in v or ends in v. We pick an arbitrary vertex v and color all paths which touch v in the colors that they have in T_v. After this initialization we iterate by taking vertices v' which are adjacent to already considered vertices. When we iterate we need to find a color permutation for $T_{v'}$ which is consistent with the coloring for T_v. This permutation is partly already defined by the color assignment for T_v. The remainder can be found in $O(C) \subseteq O(|\mathcal{M}|)$ steps. Since the order of the vertices can be obtained by a depth-first-search the second phase can be done in $O(n \cdot |\mathcal{M}|)$. This gives a total runtime of $O(n \cdot |\mathcal{M}| \cdot \log C)$. □

Note that the bound $C + D - 1$ is the best bound we can give in terms of C and D since there are packet routing instances which need this many steps. E.g., consider a path of length D with vertices $v_0, ..., v_D$ and C packets all with start vertex v_0 and destination vertex v_D.

2.2 Algorithm for Undirected Trees

It is not clear how the technique described in Section 2.1 could be applied to undirected trees. In particular, the path coloring problem on undirected trees is significantly harder than on directed trees. Also, it is not clear how the consecutive edge-coloring technique could be applied to undirected trees. However, we present a 2-approximation for the packet routing problem on undirected trees.

The algorithm works as follows: Let $T = (V, E)$ be a tree and let $I = (T, \mathcal{M})$ be a packet routing instance. Let v_r be an arbitrary vertex. We define v_r to be the root of the tree. We observe that the path of each packet can be split into two subparts: in the first part M_i moves towards v_r. In the second part M_i moves away from v_r. Let v_i be the vertex which divides these two parts. We split the routing problem into two subproblems: First we move each packet M_i from s_i to v_i. In the second part we move each packet M_i from v_i to t_i. We observe that the first part is a packet routing problem on an in-tree (a tree in which all vertices have an out-degree of at most one) and can therefore easily be solved optimally in polynomial time (FDF-algorithm, see [21]). Similarly, the second part is a packet routing instance on an out-tree (a tree in which all vertices have an in-degree of at most one) which can also be solved optimally in polynomial time (FDF-algorithm, see [21]). In the overall schedule for I, we run the optimal schedule for the first part. Then we run the optimal schedule for the second part. Denote by $TREE\,(I)$ the resulting schedule for the instance I.

Theorem 2. *For the schedule $TREE\,(I)$ it holds that $|TREE(I)| \leq 2 \cdot |OPT(I)|$.*

Proof. Since the length of an optimal schedule for each of the two subproblem forms a lower bound on the size of an optimal schedule for the original problem, we achieve an approximation ratio of two. $\square$

It can be shown that the time needed to compute $TREE\,(I)$ is bounded by $O\left(n^2\right)$ where $n = \max\{|\mathcal{M}|, |V|\}$. For details see [25].

When implementing the algorithm one would not let a packet M_i wait in v_i until all other packets have finished the first part of the schedule. We would rather always move a packet when the next edge on its path is free (and prioritize the packets such that at each timestep each packet is at least as far on its path as in the original schedule described above). But even then there are instances which show that our analysis is asymptotically tight [25].

2.3 Directed Graphs and Shortest Paths

We give a condition which allows the existence of a direct schedule within a given time horizon T. As a corollary we obtain that if the lengths of the paths are pairwise different there is an optimal direct schedule of length D.

Let $G = (V, A)$ be a directed graph, let $T \geq 0$ be a time horizon and let $I = (G, \mathcal{M}, \mathcal{P})$ be a packet routing instance. We demand that for the lengths of the paths of the packets the following conditions hold:

– All paths are shortest paths.
– For each packet M_i denote by $\mathcal{M}_i$ the set of packets M_j such that P_j shares an edge with P_i and $D_j \geq D_i$. We demand that $|\mathcal{M}_i| \leq T - D_i + 1$.

For two packets M_i and M_j such that P_i and P_j share an edge we define a value $d\,(M_i, M_j)$. Let v_{ij} be the first vertex on P_i and P_j which is used by both paths. Denote by $d\,(s_i, v_{ij})$ and $d\,(s_j, v_{ij})$ the number of edges on P_i and P_j between s_i and v_{ij} and between s_j and v_{ij}, respectively. Then we define $d\,(M_i, M_j) := d\,(s_i, v_{ij}) - d\,(s_j, v_{ij})$. Note that $d\,(M_i, M_j) = -d\,(M_j, M_i)$.

Assuming that M_i is delayed for k timesteps in the beginning, M_i collides with M_j if and only if P_i and P_j share an edge and M_j is delayed for $d(M_i, M_j) + k$ steps (here we use the fact that G is a directed graph and all paths are shortest paths). We order the packets in decreasing order of the lengths of their paths. W.l.o.g. we assume that $M_0, ..., M_k$ is such an order. Now we iterate over the packets. In the i-th iteration, we consider the packet M_i. Denote by w_j with $0 \leq j \leq i-1$ the waiting time which was computed in a previous iteration for the packet M_j. (This implies that in our schedule the packet M_j waits for w_j steps and then moves to its destination without any further delay.) We say a packet M_j *blocks* a certain waiting time m for M_i if P_i and P_j share an edge and $m = d(M_i, M_j) + w_j$. Note that each packet whose path shares an edge with P_i blocks exactly one waiting time for P_i.

Let m be the smallest unblocked waiting time for M_i. We define $w_i := m$. In our schedule the packet M_i then waits for w_i steps and then moves to its destination without any further delay. We denote by D_i the length of P_i. We will prove in Theorem 3 that $D_i + w_i \leq T$ and thus M_i needs at most T steps to reach its destination. We denote by $SPATHS(I)$ the resulting schedule.

Theorem 3. *Let G be a directed graph, let $I = (G, \mathcal{M}, \mathcal{P})$ be a packet routing instance, and let $T \geq 0$ be a time horizon with the above conditions. Then for the schedule $SPATHS(I)$ it holds that $|SPATHS(I)| \leq T$. Moreover, $SPATHS(I)$ is a direct schedule.*

Proof. It remains to prove that $D_i + w_i \leq T$ for each packet M_i. Since we considered the packets in decreasing order of their path lengths, only packets in $\mathcal{M}_i$ can possibly block a certain waiting time for M_i. Since $|\mathcal{M}_i| \leq T - D_i + 1$ and $M_i \in \mathcal{M}_i$ we conclude that at most $T - D_i$ waiting times for M_i are blocked by packets M_j with $j < i$. This proves that $w_i \leq T - D_i$ which implies that $D_i + w_i \leq D_i + (T - D_i) = T$. □

Note that the bound for the length of $SPATH(I)$ is tight. E.g., let C and D be arbitrary positive integers and consider a packet routing instance as follows: Let the graph be a directed path with vertices $v_0, v_1, ..., v_D$ and consider C packets all with start vertex v_0 and destination vertex v_D. We define $T := C + D - 1$. Then the above conditions are satisfied (since for all packets M_i we have that $|\mathcal{M}_i| = |\mathcal{M}| = C = T - D_i + 1$) and the length of the optimal schedule is exactly T. Moreover, if we weaken our condition and require only that $|\mathcal{M}_i| \leq T - D_i + 2$ we cannot guarantee the existence of a schedule of length T anymore. E.g., take the above example with $C + 1$ packets from v_0 to v_D and $T := C + D - 1$. Then each schedule needs at least $C + D > T$ steps.

We obtain the following two corollaries:

Corollary 1. *Let G be a directed graph, let $I = (G, \mathcal{M}, \mathcal{P})$ be a packet routing instance, and let $T \geq 0$ be a time horizon with the following conditions:*

- *All paths are shortest paths.*
- *Let $\mathcal{M}_i$ denote the set of packets whose path has at least $D - i$ edges. For each $i \geq 0$ we have that $|\mathcal{M}_i| \leq i + 1$.*

Then the schedule $SPATHS(I)$ is optimal with $|SPATHS(I)| = D$.

Corollary 2. *Let G be a directed graph and let $I = (G, \mathcal{M}, \mathcal{P})$ be a packet routing instance such that all paths are shortest paths and the lengths of all paths are pairwise different. Then the schedule $SPATHS(I)$ is optimal with $|SPATHS(I)| = D$.*

Compare that in [22] it was shown that under the condition of Corollary 2 there is a (not necessarily direct) schedule whose length is bounded by D. We proved that in this case there is even a direct schedule with this makespan.

2.4 Farthest-Destination-First-Algorithm on Directed Trees

The farthest-destination-first-algorithm prioritizes the packets according to the length of their remaining path. That is, packets whose remaining path is longer have a higher priority than packets whose remaining path is shorter. Ties are broken arbitrarily. It was shown by Leung [21] that on in-trees and on out-trees the FDF-algorithm works optimally. However, we show that on general directed trees the FDF-algorithm can perform arbitrarily bad in terms of the achieved performance ratio. For an instance I of the packet routing problem, denote by $FDF(I)$ a longest schedule that the FDF-algorithm could possibly compute.

Theorem 4. *For every $k \geq 1$ there is a a directed tree T_k and a packet routing instance $I_k = (T_k, \mathcal{M}_k)$ such that*

$$|FDF\,(I_k)| \geq k \cdot |OPT\,(I_k)|$$

Due to space constraints we refer to our technical report [25] for the construction.

3 Complexity Results

In this section we study the complexity of the packet routing problem. Due to space constraints we refer to our technical report [25] for detailed descriptions of the reductions.

Theorem 5. *For all $\epsilon > 0$, there is no $(6/5 - \epsilon)$-approximation algorithm for the packet routing problem with fixed paths, unless $P = NP$.*

Proof (sketch). In the reduction we employ a technique which was used in [29] for showing that the general acyclic job shop problem is NP-hard to approximate within an approximation factor of $5/4 - \epsilon$. We reduce from 3-BOUNDED-3-SAT [13]. In this variant of 3-SAT in the given formula each variable occurs at most three times (positive and negative). □

We can modify the reduction to show that the packet routing problem is also NP-hard to approximate on planar graphs and even on directed trees. Note here that in the latter case it does not make a difference whether the paths of the packets are given in advance or not. As a corollary we obtain the same result for directed chain graphs with given paths.

All these reductions rely on creating a gap of one time unit between yes- and no-instances of 3-BOUNDED-3-SAT. This raises the question whether it is NP-hard to approximate the packet routing problem with an *absolute* error in polynomial time.

Theorem 6. *For all $k > 0$, there is no approximation algorithm for the packet routing problem with fixed paths which guarantees an absolute error of at most k, unless $P = NP$.*

Acknowledgements

We would like to thank Joachim Reichel, Ronald Koch and Daniel Plümpe for helpful discussions.

References

1. Adler, M., Khanna, S., Rajaraman, R., Rosén, A.: Time-constrained scheduling of weighted packets on trees and meshes. Algorithmica 36, 123–152 (2003)
2. Meyer auf der Heide, F., Vöcking, B.: Shortest-path routing in arbitrary networks. Journal of Algorithms 31 (1999)
3. Burkard, R.E., Dlaska, K., Klinz, B.: The quickest flow problem. ZOR — Methods and Models of Operations Research 37, 31–58 (1993)
4. Busch, C., Magdon-Ismail, M., Mavronicolas, M.: Universal bufferless packet switching. SIAM Journal on Computing 37, 1139–1162 (2007)
5. Busch, C., Magdon-Ismail, M., Mavronicolas, M., Spirakis, P.: Direct routing: Algorithms and complexity. Algorithmica 45, 45–68 (2006)
6. Cole, R., Ost, K., Schirra, S.: Edge-coloring bipartite multigraphs in $O(E \log D)$ time. Combinatorica 21, 5–12 (2001)
7. di Ianni, M.: Efficient delay routing. Theoretical Computer Science 196, 131–151 (1998)
8. Erlebach, T., Jansen, K.: An optimal greedy algorithm for wavelength allocation in directed tree networks. In: Proceedings of the DIMACS Workshop on Network Design: Connectivity and Facilities Location, vol. 40, pp. 117–129. AMS (1997)
9. Erlebach, T., Jansen, K.: The complexity of path coloring and call scheduling. Theoretical Computer Science 255, 33–50 (2001)
10. Fleischer, L., Skutell, M.: Quickest flows over time. SIAM Journal on Computing 36, 1600–1630 (2007)
11. Ford, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. Operations Research 6, 419–433 (1958)
12. Ford, L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press, Princeton (1962)
13. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the theory of NP-completeness. Freeman, New York (1979)
14. Gargano, L., Hell, P., Perennes, S.: Colouring paths in directed symmetric trees with applications to WDM routing. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 505–515. Springer, Heidelberg (1997)

15. Hall, A., Hippler, S., Skutella, M.: Multicommodity flows over time: Efficient algorithms and complexity. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 397–409. Springer, Heidelberg (2003)
16. Hoppe, B., Tardos, E.: The quickest transshipment problem. Mathematics of Operations Research 25, 36–62 (2000)
17. Jansen, K.: Approximation results for wavelength routing in directed binary trees. In: Proceedings of the Workshop on Optics and Computer Science (1997)
18. Koch, R., Peis, B., Skutella, M., Wiese, A.: Real-time message routing and scheduling. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. LNCS, vol. 5687, pp. 217–230. Springer, Heidelberg (2009)
19. Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet routing and job-scheduling in $O(\textit{congestion} + \textit{dilation})$ steps. Combinatorica 14, 167–186 (1994)
20. Leighton, F.T., Maggs, B.M., Richa, A.W.: Fast algorithms for finding $O(\textit{congestion} + \textit{dilation})$ packet routing schedules. Combinatorica 19, 375–401 (1999)
21. Leung, J.Y.-T.: Handbook of Scheduling: Algorithms, Models and Performance Analysis (2004)
22. Mansour, Y., Patt-Shamir, B.: Greedy packet scheduling on shortest paths. Journal of Algorithms 14 (1993)
23. Megiddo, N.: Combinatorial optimization with rational objective functions. Mathematics of Operations Research 4, 414–424 (1979)
24. Ostrovsky, R., Rabani, Y.: Universal $O(\textit{congestion} + \textit{dilation} + \log^{1+\epsilon} N)$ local control packet switching algorithms. In: Proceedings of the 29th annual ACM Symposium on Theory of Computing, pp. 644–653 (1997)
25. Peis, B., Skutella, M., Wiese, A.: Packet routing: Complexity and algorithms. Technical Report 003-2009, Technische Universität Berlin (February 2009)
26. Rabani, Y., Tardos, É.: Distributed packet switching in arbitrary networks. In: Proceedings of the 28th annual ACM Symposium on Theory of Computing, pp. 366–375. ACM, New York (1996)
27. Raghavan, P., Upfal, E.: Efficient routing in all-optical networks. In: Proceedings of the 26th annual ACM Symposium on Theory of Computing, pp. 134–143. ACM, New York (1994)
28. Srinivasan, A., Teo, C.-P.: A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. SIAM Journal on Computing 30 (2001)
29. 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)

Minimal Cost Reconfiguration of Data Placement in Storage Area Network

Hadas Shachnai[1], Gal Tamir[1], and Tami Tamir[2]

[1] Computer Science Department, The Technion, Haifa 32000, Israel
{hadas,gal}@cs.technion.ac.il
[2] School of Computer science, The Interdisciplinary Center, Herzliya, Israel
tami@idc.ac.il

Abstract. Video-on-Demand (VoD) services require frequent updates in file configuration on the storage subsystem, so as to keep up with the frequent changes in movie popularity. This defines a natural *reconfiguration problem* in which the goal is to minimize the cost of moving from one file configuration to another. The cost is incurred by file replications performed throughout the transition. The problem shows up also in production planning, preemptive scheduling with set-up costs, and dynamic placement of Web applications. We show that the reconfiguration problem is NP-hard already on very restricted instances. We then develop algorithms which achieve the optimal cost by using servers whose load capacities are increased by $O(1)$, in particular, by factor $1 + \delta$ for any small $0 < \delta < 1$ when the number of servers is fixed, and by factor of $2 + \varepsilon$ for arbitrary number of servers, for some $\varepsilon \in [0, 1)$. To the best of our knowledge, this fundamental optimization problem is studied here for the first time.

1 Introduction

Video on Demand (VoD) services have become common in library information retrieval, entertainment and commercial applications. In a VoD system, clients are connected through a network to a set of servers which hold a large library of video programs. Each client can choose a program he wishes to view and the time he wishes to view it. The service should be provided within a small latency and guaranteeing an almost constant transfer rate of the data. The transmission of a movie to a client requires the allocation of unit load capacity (or, a *data stream*) on a server which holds a copy of the movie.

Since video files are typically large, it is impractical to store copies of all movies on each server. Moreover, as observed in large VoD systems (see, e.g., [6,21]), the distribution of accesses to movie files is highly skewed; indeed, only small fraction of the movies are requested frequently, while the vast majority (i.e., more than 80%) of the movies are rarely accessed. Hence, the number of copies held for each movie needs to reflect its popularity. The goal is to store the movie files on the servers in a way which enables to satisfy as many client requests as possible, subject to the storage and load capacity constraints.

Formally, suppose that the system consists of M video program files and N servers. Each movie file i, $1 \leq i \leq M$, is associated with a popularity parameter

$p_i^0 \in (0,1]$, where $\sum_{i=1}^{M} p_i^0 = 1$. Each server j, $1 \le j \le N$, is characterized by (i) its storage capacity, C_j, that is the number of files that can reside on it,[1] and (ii) its load capacity, L_j, which is the number of data streams that can be read simultaneously from that server. For a given popularity vector $\{p_1^0, \dots, p_M^0\}$, the *broadcast demand* of file i is $D_i^0 = p_i^0 \mathcal{L}$, where $\mathcal{L} = \sum_{j=1}^{N} L_j$ is the total load capacity of the system.[2] The *data placement problem* is to determine a placement of file copies on the servers and the amount of load capacity assigned to each file copy, so as to maximize the total amount of broadcast demand satisfied by the system. A solution to the placement problem can be represented as two $M \times N$ matrices: (i) The *placement matrix*, A, a $\{0,1\}$-matrix, $A_{i,j} = 1$ iff a copy of movie file i is stored on server j. (ii) The *broadcast matrix* B, $B_{i,j} \in \{0, 1, \dots, L_j\}$, $B_{i,j}$ is the number of broadcasts of movie i transmitted from server j. A legal placement has to satisfy the following conditions:

– $A_{i,j} = 0 \Rightarrow B_{i,j} = 0$. Clearly, server j can transmit broadcasts of movie i only if it holds a copy of this movie.
– For each server j, $\sum_i B_{i,j} \le L_j$, that is, the total number of broadcasts transmitted from server j does not exceed its load capacity.
– For each server j, $\sum_i A_{i,j} \le C_j$, that is, the number of files stored on server j does not exceed its storage capacity.

A placement is *perfect* if it satisfies the broadcast demands of all movie files. Formally, $\forall i,\ \sum_j B_{i,j} = D_i^0$. Under certain conditions, it is known that a perfect placement always exists (see Section 1.2).

The above *static* data placement problem captures well the goal of maximizing throughput in periods of time where broadcast requirements remain unchanged. However, in general, throughout the operation of a VoD system new movies are released and may become most popular, while the popularity of the previously *hot* movies drops. The system should be able to support any change in the distribution on file popularities. Thus, in order to maintain high throughput, the system needs to adjust the placement of file copies and the allocation of load capacity to these copies. This involves replications and deletions of files. File replications incur significant cost as they require bandwidth and other resources on the source, as well as the destination server. Minimizing this cost is crucial for optimizing system performance. This is the focus of our paper.

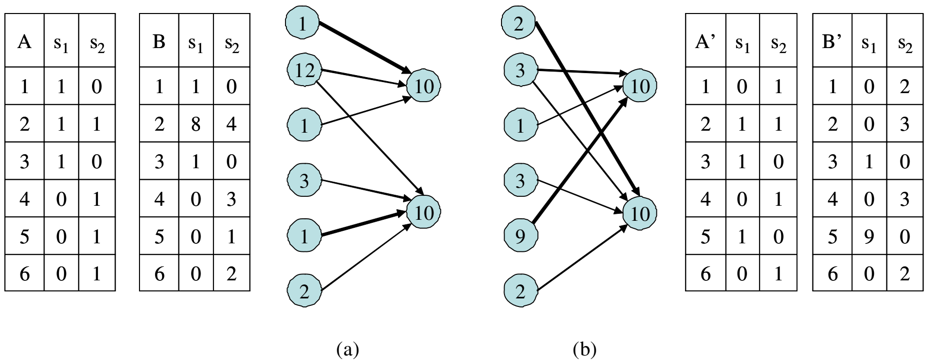
Our *dynamic* data placement problem can be formalized as follows. Given a perfect placement of file copies on the servers, with the popularity vector $\langle p_1^0, \dots, p_M^0 \rangle$, suppose that the popularity vector changes to $\langle p_1, \dots, p_M \rangle$, with the corresponding broadcast demands $\langle D_1, \dots, D_M \rangle$. The *reconfiguration problem* is to modify the initial data placement to a perfect placement for $\langle D_1, \dots, D_M \rangle$ at minimum total cost. In updating system configuration, the cost of storing a new copy of movie file i on server j is given by $s_{i,j}$, while the assignment of load capacity to existing copy of file i on server j is free. We denote by $c_{i,j}$ the cost

[1] Unless specified otherwise, we assume that all files have the same size.
[2] The broadcast demands are assumed to be integers. Rounded values can be obtained by standard solutions for the apportionment problem [22].

of having a copy of movie i on server j after the reconfiguration. Given the initial placement matrix A, we denote by A' the placement after reconfiguration. Then, by definition, $c_{i,j} = 0$ if $A_{i,j} = 1$, and $c_{i,j} = s_{i,j}$ if $A_{i,j} = 0$ and $A'_{i,j} = 1$. In other words, the cost of increasing the (i, j)-entry in the assignment matrix, A, is $s_{i,j}$ while changes in the broadcast matrix B are free. The total cost of switching from a placement A to a placement A' is given by $\sum_{i,j} c_{i,j}$. Note that file deletions incur no cost. Clearly, the new assignment must satisfy the three legal-placement conditions.

A VoD system is *homogeneous* if all servers have the same load capacities, i.e., $L_1 = \cdots = L_N = L$, and the same storage capacities, i.e., $C_1 = \cdots = C_N = C$ (see, e.g., [5,10]). In this paper we assume that the system is *semi-homogeneous*, i.e., all servers have the same load capacities and arbitrary storage capacities.

Example 1. Consider a system of two servers which holds 6 movies. The popularity vector is $\langle 0.05, 0.6, 0.05, 0.15, 0.05, 0.1\rangle$. Both servers have the same load capacity $L_1 = L_2 = 10$, while the storage capacities are $C_1 = 3, C_2 = 4$. Having $\mathcal{L} = 20$, the demand vector is $D^0 = \langle 1, 12, 1, 3, 1, 2\rangle$. Figure 1(a) presents a possible perfect placement for this instance. The assignment is described by the assignment and broadcast matrices A, B and by a bipartite graph, in which the left hand side nodes represent movie files and the right hand side nodes represent servers; an edge (i, j) implies that a copy of movie file i is stored on server j. The maximal degree of a server-node is its storage capacity. Assume that the popularity vector is changed to $\langle 0.1, 0.15, 0.05, 0.15, 0.45, 0.1\rangle$. Figure 1(b) presents a new placement, obtained from the previous one by adding (and deleting) copies of two files. The new placement is perfect for the new demand vector $D = \langle 2, 3, 1, 3, 9, 2\rangle$. The reconfiguration cost is $c_{1,2} + c_{5,1}$.



A	s_1	s_2
1	1	0
2	1	1
3	1	0
4	0	1
5	0	1
6	0	1

B	s_1	s_2
1	1	0
2	8	4
3	1	0
4	0	3
5	0	1
6	0	2

A’	s_1	s_2
1	0	1
2	1	1
3	1	0
4	0	1
5	1	0
6	0	1

B’	s_1	s_2
1	0	2
2	0	3
3	1	0
4	0	3
5	9	0
6	0	2

Fig. 1. A perfect placement before (a) and after (b) the popularity change. Bold edges represent changes in storage assignment.

Applications. As mentioned above, a main motivation for this work comes from the constant need for dynamic data placement in VoD systems. Our reconfiguration problem shows up also in production planning, as well as in machine scheduling (see a survey in [17]). Suppose that M tasks are processed by N machines. Each machine has limited amount of resources and a time interval in which it is active. The resource requirements of the tasks are changing over time. Tasks may need to be reassigned to the machines in order to fit their new requirement. Reassignment of tasks incurs some cost due to migration overhead and the set-up of the machines. The goal is to reassign the tasks to the machine so as to minimize the transition cost. Finally, our problem naturally arises in dynamic placement of clustered Web applications (see, e.g., [8]). Web applications are dynamically placed on server machines so as to adjust system configuration to the availability of resources. The goal is to maximize the amount of client demands that can be satisfied by the applications while minimizing the number of placement changes.

1.1 Our Results

We first show (in Section 2) that the reconfiguration problem is NP-hard, already when the system consists of two servers, with unit reconfiguration costs and very restricted changes in file popularity. In practical scenarios, it is often the case that the new popularity vector has a perfect placement. This occurs, e.g., where the new vector is a *permutation* of the initial vector, that is, the popularity distribution function remains unchanged. For such scenarios, we give in Sections 3 and 4 algorithms which solve the reconfiguration problem optimally, by using servers whose load capacities are increased by a small constant factor. Specifically, for a fixed number of servers, we give in Section 3 an algorithm which accepts as parameter a value $0 < \delta < 1$ and achieves the optimal reconfiguration cost by using servers whose load capacities are $L(1+\delta)$. The running time of the algorithm depends on the value of δ (see Section 3.1). For more general inputs, in which the number of servers may be arbitrarily large, we give in Section 4 an algorithm that achieves the optimal cost, by using servers whose load capacities are increased by factor of $2 + \varepsilon$, for some $\varepsilon \in [0, 1)$.

Our main approximation technique, applied (in Section 4) to general instances of the reconfiguration problem, relies on finding a linear programming relaxation whose optimal (fractional) solution is a lower bound on the optimal solution for our problem, and for which we can apply rounding without increasing the total cost. To find such a relaxation, we iteratively modify the initial linear relaxation for our problem until we obtain a linear program which reduces our problem to job scheduling on unrelated machines. It is worth noting that even though the optimal *integral* solutions for the programs in this sequence cannot be related to the optimum cost for our problem, it holds that the optimal (fractional) solution for each program is a lower bound for the optimum cost for our problem.

Due to space constraints, some of the proofs and implementation details are omitted. The detailed results appear in [15].

1.2 Related Work

The data placement problem has been extensively studied (see, e.g., [20,5,10,18,8] and a comprehensive survey in [9]). The paper [16] considers the problem of finding a *perfect placement* of movie files on the servers. The paper shows the hardness of the perfect placement problem and that such a placement always exists, e.g., when $\sum_{j=1}^{N} C_j \geq M + N - 1$. The paper [16] also presents an algorithm for the data placement problem, for inputs in which the ratio L_j/C_j is equal for all $1 \leq j \leq N$ (*uniform ratio servers*). The paper shows that the algorithm achieves a ratio of $1 - 1/(1 + C_{min})$ to the optimal, where $C_{min} = \min_j C_j$. Golubchik et al. gave in [5] a tighter analysis of this algorithm and showed that it achieves the ratio $1 - 1/(1 + \sqrt{C_{min}})^2$, and that this ratio is optimal for *any* algorithm for this problem. The paper [5] also presents a PTAS for the data placement problem with uniform ratio servers. Later papers considered a generalized version of the problem, where files may be of different sizes (see, e.g., [10,18]).

For the more realistic model, where file popularities may change over time, there has been some earlier work which refers to the resulting *data migration* problem: Compute an efficient plan for moving data stored on devices (e.g., a set of servers) in a network from one configuration to another. Since the servers are constrained in handling simultaneous transmissions of files, data migration is done in rounds, where each round handles the delivery of a subset of the files to their destinations. Common objective functions are minimizing the makespan of the migration schedule, or the sum of completion times of the servers (see, e.g., [12,13]). The paper [11] considers a somewhat 'dual' reconfiguration problem: the goal is to convert the existing layout to a good new layout (that is part of the solution), using a limited number of migration rounds. Surveys of known results for the data migration problem are given in [11,4]. The data migration problem differs from our reconfiguration problem in several ways. (*i*) The final configuration is given as part of the input for data migration, while it is part of the solution for our problem. (*ii*) In data migration the output is a migration schedule, while no assignment schedule is output when solving the reconfiguration problem, and finally, (*iii*) in data migration we measure the quality of the migration schedule, while in our problem we measure the cost of the final configuration.

There has been some other work on reconfiguration of data placement, in which heuristic solutions were investigated through experimental studies (e.g., [14,23,3,7]). The paper [8] studies a generalization of our reconfiguration problem, in which file deletions incur unit costs, and the files are of arbitrary sizes. The paper presents experimental results for greedy-based heuristics for the problem. We are not aware of earlier theoretical results for our reconfiguration problem.

2 Hardness Result

We show that the reconfiguration problem is NP-hard even if the system consists of only two servers, and even if the popularity changes are limited such that the new popularity vector is a permutation of the previous one. In other words, the popularity *distribution function* is preserved. We use a reduction from a variant of the subset-sum problem. For a set of integers X, let S_X denote the total size of elements in X.

Definition 1. *The* smallest subsets with a given difference *problem is defined as follows: Given are two sets of non-negative integers $X = \{x_1, x_2, \ldots x_{n_X}\}$ and $Y = \{y_1, y_2, \ldots, y_{n_Y}\}$, and an integer z. W.l.o.g, $n_X \leq n_Y$. It is known that there exists a subset $Y'' \subseteq Y$ of size n_X such that $S_X = S_{Y''} + z$. The goal is to find the smallest integer $k \geq 1$ such that there exist $X' \subseteq X$ and $Y' \subseteq Y$, where $|X'| = |Y'| = k$, and $S_{X'} = S_{Y'} + z$. Note that such an integer k must exist, since for $k = n_X$, the sets $X' = X, Y' = Y''$ form a solution.*

Based on the hardness of the the *smallest subsets with a given difference* problem (hardness proof omitted), we can prove the following:

Theorem 1. *The reconfiguration problem is NP-hard even if the system consists of only two servers, all replication costs are uniform, and even if the popularity changes are limited such that the new popularity vector is a permutation of the previous one.*

3 Minimal Cost Algorithm for Fixed Number of Servers

We present a poly-time algorithm which finds a minimal-cost reconfiguration for a semi-homogeneous system, assuming that the number of servers, N, is some fixed constant. The algorithm outputs a placement which achieves the optimal cost and uses servers with load capacities $(1 + 3\delta)L$, for a parameter $\delta \in (0, 1]$.

Given the parameter $0 < \delta \leq 1$, a movie file i is considered *big*, if $D_i \geq \delta L$, else, movie i is small. Our algorithm handles separately the two types of movies. It produces allocations with the following properties: (P_1) For every big movie i, and every server j, the broadcast allocation $B_{i,j}$ of server j to movie i is either 0 or at least δL and an integral multiple of $\delta^2 L$, i.e., $B_{i,j} = k\delta^2 L$, where k is an integer in $[1/\delta, 1/\delta^2]$. (P_2) Every small movie is stored on a single server, on which it is allocated all of its broadcast demand. Formally, for any small movie i, for a single server j, $B_{i,j} = D_i$, and for any $j' \neq j$, $B_{i,j'} = 0$.

We show below that if a slight augmentation of the load capacities is allowed, then a minimal cost reconfiguration fulfilling the above properties can be found in polynomial time. In addition, limiting the set of reconfigurations to ones fulfilling the properties does not affect the minimal cost. An overview of the algorithm is given in Figure 2.

```
Reconfiguration Algorithm
  For each storage and load allocation to the big movies, satisfying $P_1$, do:
    Let $L_j^b$ and $C_j^b$ be the total load and storage allocation to big movies on server $j$.
    For all $1 \leq j \leq N$ do $L_j := L - L_j^b + 2\delta$ and $C_j := C_j - C_j^b$
    Find a minimum cost placement of the small movies on the servers
      assuming server $j$ has load capacity $L_j$ and storage capacity $C_j$.
  Select a configuration for the big movies which yields minimum total cost.
```

Fig. 2. Algorithm for updating data placement on a set of servers

3.1 Analysis of the Algorithm

In analyzing the algorithm, we use the next technical lemmas.

Lemma 1. *Limiting the allocation to a one satisfying* (P_1) *and* (P_2) *may require an increase by at most a factor of* $(1+2\delta)$ *in the load capacity, without changing the configuration cost.*

Lemma 2. *The set of possible configurations of copies of the big movies, along with the corresponding allocations of load capacities, has a polynomial size.*

Lemma 3. *Given a configuration of the big movies on the servers, there exists a polynomial time algorithm which finds for the small movies a placement of minimum cost, fulfilling property* (P_2)*, where each server* j *has storage capacity* C_j *and load capacity* $L_j(1+\delta)$*.*

Proof. Let R denote the set of small movies, and $M_r = |R|$. Index the small movies $1, \ldots, M_r$. Denote by $x_{i,j} \in \{0, 1\}$ an indicator variable for the assignment of movie i to server j, $i \in R$ and $1 \le j \le N$. The costs $c_{i,j}$ are the given replication costs. Note that once the big movies have been assigned, the servers may have different load capacities; however, by scaling the broadcast requirements of the movies, we may assume, w.l.o.g., that the load capacities of the servers satisfy $L_1 = \cdots = L_N = \hat{L}$. Specifically, for a movie i and server j, define $D_{i,j} = D_i \cdot \hat{L}/L_j$. The assignment will be determined by rounding the solution to the following linear program, LP.

$$
\begin{array}{lll}
\textit{(LP)}: \quad \text{minimize} & \displaystyle\sum_{i=1}^{M_r}\sum_{j=1}^{N} x_{i,j} \cdot c_{i,j} & \\
\text{subject to:} & \displaystyle\sum_{i=1}^{M_r} x_{i,j} \cdot D_{i,j} \le \hat{L} & \text{for } 1 \le j \le N, \\
& \displaystyle\sum_{i=1}^{M_r} x_{i,j} \le C_j & \text{for } 1 \le j \le N, \\
& \displaystyle\sum_{j=1}^{N} x_{i,j} = 1 & \text{for } 1 \le i \le M_r, \\
& x_{i,j} \ge 0 & \text{for } 1 \le j \le N, \quad 1 \le i \le M_r.
\end{array}
$$

Claim 1. Given an optimal solution for LP, with the cost C, there exists a polynomial time algorithm which finds an assignment of the small movies to servers of load capacity $\hat{L}(1+\delta)$, whose total cost is at most C.

Proof. Given an optimal (fractional) solution for LP, we use a rounding technique of Shmoys and Tardos [19]. Specifically, we construct a bipartite graph in which server j is represented by at most C_j vertices, $1 \le j \le N$. A solution for the fractional matching problem on this graph induces an integral matching of the same cost with the same cardinality. We show below that the resulting integral solution may use servers whose load capacities are at most $\hat{L}(1+\delta)$.

Formally, sort the small movies in non-decreasing order by load requirements. Let $G_B = (V \cup U, E)$ be a bipartite graph, where $U = \{u_i | 1 \le i \le M_r\}$ represents the set of small movies, and V is the set of server vertices: $V = \{v_{j,k} | 1 \le j \le N, 1 \le k \le \sigma_j\}$ where $\sigma_j = \lceil \sum_{i=1}^{M_r} x_{i,j} \rceil$ is the total number of small movies stored on server j. Clearly, $\sigma_j \le C_j$. The vertices $v_{i,1}, \ldots, v_{i,\sigma_j}$ correspond to server j.

The set of edges E of G_B is defined as follows. Given the values of $x_{i,j}$ for $1 \le i \le M_r$, $1 \le j \le N$, for any server j:

(i) If $\sum_{i=1}^{M_r} x_{i,j} \le 1$ then there is a single vertex $v_{i,1} \in V$ corresponding to server j. In this case, for any $1 \le i \le M_r$ such that $x_{i,j} \ge 0$, we add in G_B an edge $(u_i, v_{j,1})$, and set its weight to be $w(u_i, v_{j,1}) = x_{i,j}$.

(ii) If $\sum_{i=1}^{M_r} x_{i,j} > 1$, find the minimum index i_1 such that $\sum_{i=1}^{i_1} x_{i,j} \ge 1$, then E contains all the edges $(u_i, v_{j,1})$, $1 \le i \le i_1 - 1$ for which $x_{i,j} > 0$. For each of these edges set $w(u_i, v_{j,1}) = x_{i,j}$. Now, add to E an edge $(u_{i_1}, v_{j,1})$, whose weight is $w(u_{i_1}, v_{j,1}) = 1 - \sum_{i=1}^{i_1-1} w(u_i, v_{j,1})$. Thus, the sum of weights of the edges incident to $v_{j,1}$ is exactly 1. If $\sum_{i=1}^{i_1} x_{i,j} > 1$ add an edge $(u_{i_1}, v_{j,2})$, whose weight is $w(u_{i_1}, v_{j,2}) = (\sum_{i=1}^{i_1} x_{i,j}) - 1$. Proceed next to movies with $i > i_1$ i.e., those with smaller broadcast requirements on server j. Similar to the above process for $v_{j,1}$, add edges incident to $v_{j,2}$, until a total of exactly one movie is assigned to $v_{j,2}$, and so on.

Let i' be the index of the last movie for which an edge is assigned this way, i.e, $i' = i_{\sigma_j - 1}$. Now, for any $i > i'$ for which $x_{i,j} > 0$ add an edge (u_i, v_{j,σ_j}) and set $w(u_i, v_{j,\sigma_j}) = x_{i,j}$.

For each server vertex $v_{j,k}$, let $D_{j,k}^{max}$ ($D_{j,k}^{min}$) denote the maximum (minimum) of the broadcast requirements $D_{i,j}$ corresponding to the edges $(u_i, v_{j,k})$ incident to $v_{j,k}$. We note that the weight function on the edges of G_B defines a fractional matching, in which any movie vertex U_i is exactly matched to a server vertex $v_{j,k}$, $1 \le k \le \sigma_j - 1$. In other words, for any $1 \le j \le N$ and $1 \le k \le \sigma_j - 1$, $\sum_{i=1}^{M_r} w(u_i, v_{j,k}) = 1$. In addition, for all $1 \le j \le N$ and $1 \le k \le \sigma_j - 1$,

$$D_{j,k}^{min} \ge D_{j,k+1}^{max}. \tag{1}$$

We now summarize the steps of the rounding procedure which assigns the small movies to the servers.

1. Given an optimal solution for LP, form the bipartite graph G_B.
2. Find a min-cost (integer) matching H that matches all movie vertices in G_B.
3. For each edge $(u_i, v_{j,k}) \in H$ place movie i on server j.

We show that the assignment obtained in Step 3 of the algorithm has cost C, and that the overall load capacity used on any server is at most $\hat{L}(1+\delta)$. Since we defined in G_B a fractional matching of cost C, there exists in G_B an integral matching, H, whose cost is C, such that all the movie vertices are exactly matched. Therefore, the matching found in Step 2 has cost C. Since the cost of the assignment is equal to the cost of the matching, the output solution has at most the optimal cost C.

Next, we show that the total broadcast requirement assigned on each server is at most $L(1+\delta)$. Consider the movies assigned to server j. For any $1 \le j \le N$, there are $\sigma_j \le C_j$ vertices representing server j in G_B. Each of these vertices $v_{j,k}$ adds at most one movie file to server j (the movie which corresponds to the edge selected for the matching H, among those incident to $v_{j,k}$). Therefore, at most C_j small movies are assigned to server j. It follows that the total broadcast requirement of the movies on server j is at most

$$\sum_{k=1}^{\sigma_j} D_{j,k}^{max} \quad \le \quad D_{j,1}^{max} + \sum_{k=2}^{\sigma_j} D_{j,k}^{max} \quad \le \quad \delta\hat{L} + \sum_{k=1}^{\sigma_j - 1} D_{j,k}^{min}$$

$$\le \delta L + \sum_{k=1}^{\sigma_j} \sum_{\{i|(u_i, v_{j,k}) \in E\}} D_{i,j} \cdot w(u_i, v_{j,k}) = \delta\hat{L} + \sum_{i=1}^{M_r} D_{i,j} x_{i,j} \le \hat{L}(1+\delta).$$

The second inequality follows from (1) and the fact that $D_{i,j} \le \delta\hat{L}$ for all $1 \le i \le M_r$. This completes the proof. ■

Combining the above lemmas, we summarize in the next result.

Theorem 2. *Given a system of N servers, each having load capacity L and arbitrary storage capacities $C_j \ge 1$, $1 \le j \le N$, the Reconfiguration algorithm finds in polynomial time a placement of the files whose cost is optimal, by using servers with load capacities $L(1+3\delta)$.*

4 Minimal Cost Algorithm for Any Number of Servers

In this section we consider a system with *arbitrary* number of servers. We first show that when M is *fixed*, the problem can be optimally solved.

Theorem 3. *The reconfiguration problem is solvable in polynomial time when M, the number of movies, is fixed.*

For the case where M may be arbitrarily large, we present below a polynomial time algorithm which finds a minimal-cost reconfiguration in a semi-homogeneous system. Given an instance I, our algorithm outputs a minimal-cost placement, by using servers of load capacities $(2+\varepsilon)L$ where $\varepsilon = max_{\{i|D_i > L\}}\{D_i/L - \lfloor D_i/L \rfloor\}$.

The Algorithm. The following is an overview of the algorithm. (i) Partition the movies to *big* and *small*; movie i is big if $D_i > L$, else movie i is small. (ii) Solve a linear programming relaxation to obtain a lower bound on the optimal solution for the reconfiguration problem. (iii) Round the (fractional) solution of the linear program to obtain an integral solution of optimal cost. (iv) Use the integral solution to assign movie copies to N servers, where server j has storage capacity C_j and load capacity $(2+\varepsilon)L$.

$$
\begin{array}{lll}
(LP_1): \text{ minimize} & \displaystyle\sum_{i\in Big}\sum_{j=1}^{N} x_{i,j}\cdot c_{i,j} + \sum_{i\in Small}\sum_{j=1}^{N} y_{i,j}\cdot c_{i,j} & \\
\text{subject to:} & \displaystyle\sum_{i\in Big} x_{i,j}\cdot L + \sum_{i\in Small} y_{i,j}\cdot D_i \le L \quad \text{for } 1\le j\le N & (2)\\
& \displaystyle\sum_{i\in Big} x_{i,j} + \sum_{i\in Small} y_{i,j} \le C_j \quad \text{for } 1\le j\le N & (3)\\
& \displaystyle\sum_{j=1}^{N} x_{i,j} = \frac{D_i}{L} \quad \text{for } i\in Big & (4)\\
& \displaystyle\sum_{j=1}^{N} y_{i,j} = 1 \quad \text{for } i\in Small & (5)\\
& 0\le x_{i,j}\le 1 \quad \text{for } 1\le j\le N,\ i\in Big & \\
& 0\le y_{i,j}\le 1 \quad \text{for } 1\le j\le N,\ i\in Small &
\end{array}
$$

Solving an LP Relaxation. We show how a natural LP relaxation for our problem can be modified to obtain another LP, from which we derive an optimal integral solution. Let $x_{ij}\in[0,1]$ denote the fraction of the load capacity L of server j allocated to big movie i. We denote by y_{ij} the fraction of D_i allocated to small movie i on server j. Also, $c_{i,j}$ is the given replication cost (which depends on the initial configuration). Consider the following LP relaxation, LP_1 for the reconfiguration problem.

Constraints (2) ensure that the total load capacity used by copies of the big movies and by the small movies on each server is at most L. Constraints (3) ensure that the total storage required on server j is at most C_j. Constraints (4) and (5), together with constraints (2), guarantee that each (big or small) movie is allocated D_i broadcasts.

Next, we modify LP_1 as follows. For any big movie i let $k_i=\lfloor D_i/L\rfloor$. Consider the linear program LP_2, in which constraints (2) and (4) are replaced by

$$\sum_{i\in Big} x_{i,j}\cdot\frac{D_i}{k_i} + \sum_{i\in Small} y_{i,j}\cdot D_i \le L \quad \text{for } 1\le j\le N \quad \text{and} \tag{6}$$

$$\sum_{j=1}^{N} x_{i,j} = k_i \quad \text{for } i\in Big \tag{7}$$

Note that constraints (6) allow to assign to big movie i some fraction of D_i/k_i on server j; also, constraints (7) guarantee that big movie i is allocated D_i broadcasts. Finally, partition each big movie i to k_i sub-movies. Thus, we replace the variables $x_{i,j}$, $1\le j\le N$, $i\in Big$ by the set of variables $x_{i,j,r}$, $1\le r\le k_i$. Intuitively, we partition the load requirement of movie i to k_i, so we can now consider k_i sub-movies, where each needs to be allocated $\hat{D}_i=\frac{D_i}{k_i}$ broadcasts. Note that $\hat{D}_i\le\max_{i\in Big}\{D_i/k_i\}$. We rewrite the LP relaxation as LP_3.

Rounding the Fractional Solution. We note that LP_3 can be viewed as the linear programming relaxation of an input for job scheduling on unrelated machines with cardinality constraints, in which we need to schedule a set of jobs on N unrelated machines. The set of jobs $\mathcal{J}$ corresponds to all the small movies and the collection of sub-movies for the big movies, i.e., $|\mathcal{J}| = |Small| + \sum_{i \in Big} k_i$. The processing time of a job corresponding to a small movie i on machine j is $p_{ij} = D_i$, and the processing time of any of the jobs corresponding to the k_i sub-movies of big movie i on machine j is $p_{ij} = \lceil D_i/k_i \rceil$. Note that if $k_i = 1$ then $p_{ij} = D_i < 2L$. If $k_i > 1$ then $D_i/k_i < 1.5L$ implying (for all $L > 1$) $\lceil D_i/k_i \rceil < 2L$. In the reduction to the scheduling problem, the cost of processing job i on machine j is c_{ij}, $\forall\ i$ and $1 \le j \le N$. The makespan of any machine j is at most L, and the maximal number of jobs that can be assigned to machine j is C_j. The goal is to schedule all jobs on the machines, subject to the makespan and cardinality constraints, so as to minimize the total cost.

Given an optimal solution for LP_3, we can apply the rounding technique of Shmoys and Tardos [19], as described in the proof of Claim 1. The resulting integral solution can be used to determine the storage allocation. The assignment matrix is given by the variables $x_{i,j,r}, y_{i,j}$ and the broadcast matrix is given by the allocated processing time. Formally, for a small movie i, assign a single copy of i on server j if $y_{i,j} = 1$, i.e., $A_{i,j} = 1$ and $B_{i,j} = D_i$. For any big movie i, note that each sub-movie is allocated $\lceil D_i/k_i \rceil > L$ processing units, therefore (given that the makespan of the rounded solution is at most $2L$) for all i, j, $\sum_{r=1}^{k_i} x_{i,j,r} \in \{0, 1\}$. If $\sum_{r=1}^{k_i} x_{i,j,r} = 1$ then assign a copy of big movie i on server j, i.e., $A_{i,j} = 1$ and $B_{i,j} = \lceil D_i/k_i \rceil$.

$$
\begin{array}{lll}
(LP_3): & \text{minimize} & \displaystyle\sum_{i \in Big} \sum_{r=1}^{k_i} \sum_{j=1}^{N} x_{i,j,r} \cdot c_{i,j} + \sum_{i \in Small} \sum_{j=1}^{N} y_{i,j} \cdot c_{i,j} \\
 & \text{subject to:} & \displaystyle\sum_{i \in Big} \sum_{r=1}^{k_i} x_{i,j,r} \cdot \lceil \frac{D_i}{k_i} \rceil + \sum_{i \in Small} y_{i,j} D_i \le L \quad 1 \le j \le N \\
 & & \displaystyle\sum_{i \in Big} \sum_{r=1}^{k_i} x_{i,j,r} + \sum_{i \in Small} y_{i,j} \le C_j \qquad 1 \le j \le N \\
 & & \displaystyle\sum_{j=1}^{N} x_{i,j,r} = 1 \qquad \text{for} \quad i \in Big, \quad 1 \le r \le k_i \\
 & & \displaystyle\sum_{j=1}^{N} y_{i,j} = 1 \qquad \text{for} \quad i \in Small \\
 & & 0 \le x_{i,j,r} \le 1 \qquad \text{for} \quad 1 \le j \le N, \quad i \in Big, \quad 1 \le r \le k_i \\
 & & 0 \le y_{i,j} \le 1 \qquad \text{for} \quad 1 \le j \le N, \quad i \in Small
\end{array}
$$

Analysis. The proof of the following theorem is based on the fact that the optimal solution for LP_3 is a lower bound for the cost of an optimal solution.

Theorem 4. *The above algorithm outputs in polynomial time a solution of cost at most $OPT(I)$. The movies can be stored on N servers with storage capacities $C_1, \ldots, C_N$ and load capacities $(2+\varepsilon)L$ where $\varepsilon = max_{\{i|D_i>L\}}\{D_i/L - \lfloor D_i/L \rfloor\}$.*

In the full version of the paper [15] we extend the above result to a system where the servers may have arbitrary load capacities.

References

1. Chou, C.F., Golubchik, L., Lui, J.C.S.: A performance study of dynamic replication techniques in continuous media servers. In: IEEE MASCOTS (2000)
2. Caprara, A., Kellerer, H., Pferschy, U., Pisinger, D.: Approximation algorithms for knapsack problems with cardinality constraints. Euro. Journal of OR 123 (2000)
3. Dukes, J., Jones, J.: Using Dynamic Replication to Manage Service Availability in a Multimedia Server Cluster. In: Roca, V., Rousseau, F. (eds.) MIPS 2004. LNCS, vol. 3311, pp. 194–205. Springer, Heidelberg (2004)
4. Gandhi, R., Halldórsson, M.M., Kortsarz, G., Shachnai, H.: Improved results for data migration and open shop scheduling. ACM Trans. on Algorithms 2(3) (2006)
5. Golubchik, L., Khanna, S., Khuller, S., Thurimella, R., Zhu, A.: Approximation algorithms for data placement on parallel disks. In: Proc. of SODA (2000)
6. Griwodz, C., Bar, M., Wolf, L.C.: Long-term movie popularity models in Video-on-demand Systems: Or the life of an On-demand movie. ACM Multimedia (1997)
7. Guo, X., Li, J., Yang, J., Wang, J.: The research on dynamic replication and placement of file using dual-threshold dynamic file migration algorithm. In: CSSE, vol. (3) (2008)
8. Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., Tantawi, A.: Dynamic application placement for clustered web applications. In: WWW (2006)
9. Kashyap, S.: Algorithms for data placement, reconfiguration and monitoring in storage networks. Ph.D. Dissertation. CS Department, Univ. of Maryland (2007)
10. Kashyap, S., Khuller, S.: Algorithms for non-uniform size data placement on parallel disks. In: FST & TCS (2003)
11. Kashyap, S., Khuller, S., Wan, Y.-C., Golubchik, L.: Fast reconfiguration of data placement in parallel disks. In: ALENEX (2006)
12. Khuller, S., Kim, Y., Wan, Y.C.: Algorithms for Data Migration with Cloning. In: ACM Symposium on Principles of Database Systems (2003)
13. Kim, Y.: Data Migration to Minimize the Average Completion Time. In: Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms, pp. 97–98 (2003)
14. Lie, P.W.K., Lui, J.C.S., Golubchik, L.: Threshold-based dynamic replication in large-scale video-on-demand systems. In: Proc. of the Eighth International Workshop on Research Issues in Database Engineering (RIDE), pp. 52–59 (1998)
15. Shachnai, H., Tamir, G., Tamir, T.: Minimal Cost Reconfiguration of Data Placement in Storage Area Network, `http://www.cs.technion.ac.il/~hadas/PUB/reconf_jour.pdf`
16. Shachnai, H., Tamir, T.: On two class-constrained versions of the multiple knapsack problem. Algorithmica 29, 442–467 (2001)

17. Shachnai, H., Tamir, T.: Polynomial time approximation schemes for class-constrained packing problems. J. of Scheduling 4(6), 313–338 (2001)
18. Shachnai, H., Tamir, T.: Approximation Schemes for Generalized 2-dimensional Vector Packing with Application to Data Placement. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 165–177. Springer, Heidelberg (2003)
19. Shmoys, D., Tardos, E.: Scheduling unrelated machines with Costs. In: SODA (1993)
20. Wolf, J.L., Yu, P.S., Shachnai, H.: Disk load balancing for video-on-demand systems. ACM Multimedia Systems J. 5, 358–370 (1997)
21. Yu, H., Zheng, D., Zhao, B.Y., Zheng, W.: Understanding user behavior in large scale video-on-demand systems. In: ACM SIGOPS/EuroSys (2006)
22. Peyton Young, H.: Equity in theory and practice. Princeton University Press, Princeton (1995)
23. Zhou, X., Xu, C.Z.: Optimal video replication and placement on a cluster of video-on-demand servers. In: Int. Conference on Parallel Processing, ICPP (2002)

Competitive Multi-dimensional Dynamic Bin Packing via L-Shape Bin Packing*

Prudence W.H. Wong and Fencol C.C. Yung

Department of Computer Science, University of Liverpool, UK
pwong@liverpool.ac.uk, ccyung@graduate.hku.hk

Abstract. We study d-dimensional dynamic bin packing for general d-dimensional boxes, for $d \geq 2$. This problem is a generalization of the bin packing problem in which items may arrive and depart dynamically. Our main result is a 3^d-competitive online algorithm. We further study the 2- and 3-dimensional problem closely and improve the competitive ratios. Technically speaking, our d-dimensional result is due to a space efficient offline single bin packing algorithm, which is a variant of d-dimensional NFDH. We introduce an interesting notion of d-dimensional L-shape bin and show that effective offline packing into L-shape bin leads to effective online dynamic packing into unit-sized bins.

We also investigate the resource augmentation version of the problem where the online algorithm can use d-dimensional bins of size $s_1 \times s_2 \times \cdots \times s_d$ for $s_i \geq 1$ while the optimal offline algorithm uses unit-sized bins. We give conditions for the online algorithm to match the performance of the optimal offline algorithm, i.e., 1-competitive.

1 Introduction

Bin packing is a classical combinatorial optimization problem that has been studied since the early 70's and different variants continue to attract researchers' attention (see [9,7,5]). The problem was first studied in one-dimension (1-D) and has been extended to multi-dimension (d-D for $d \geq 1$). In d-D bin packing, the items are d-dimensional with length in the range $(0, 1]$ in each dimension and the bin is a d-dimensional bin with all lengths equal to 1. Items are oriented and cannot be rotated. The objective is to pack the items into a minimum number of unit-size bins such that the items do not overlap and do not exceed the boundary of the bin. The bin packing problem is NP-complete [13], even for 1-D.

The problem has been studied both in offline and online setting. In the offline setting, all the items and their sizes are given in advance. In the online setting, items may arrive at arbitrary time; item arrival time and item size are only known when an item arrives. The performance of an online algorithm is measured using competitive analysis [2]. Consider any online algorithm $\mathcal{A}$. Given an input I, let $OPT(I)$ and $\mathcal{A}(I)$ be the maximum number of bins used by the optimal offline algorithm and $\mathcal{A}$, respectively. Algorithm $\mathcal{A}$ is said to be c-competitive if there exists a constant b such that $\mathcal{A}(I) \leq c\,OPT(I) + b$ for all I.

* The work is partly supported by EPSRC Grant EP/E028276/1. We thank Tim Hartnack, Lars Prädel, Isaac K.K. To for discussion at the beginning of the work.

Table 1. Competitive ratios. Results in this paper are marked with [*].

	1-D	2-D	3-D	d-D
upper bound	2.788 [6]	7.788 [*]	22.788 [*]	3^d [*]
(previous)		8.5754 [11]	35.346 [11]	2×3.5^d [11]
lower bound	2.666 [19]	3.70301 [11]	4.85383 [11]	$d+1$ [11]

Dynamic bin packing. Most existing work focused on "static" bin packing in the sense that items do not depart. In some potential applications like warehouse storage, a more realistic model takes into consideration of dynamic arrival and departures of items. This natural generalization, known as *dynamic bin packing*, was introduced by Coffman, Garey and Johnson [6]. In this generalization, items arrive over time, reside for some period of time, and may depart at arbitrary time. Each item has to be assigned to a bin from the time it arrives until it departs. The objective is to minimize the maximum number of bins used over all time. Note that migration to another bin is not allowed yet rearrangement of items within a bin is allowed. One can imagine that warehouses (c.f. bins) may be geographically far from each other making migration infeasible but rearrangement within a warehouse is feasible.[1]

The dynamic bin packing problem was first studied in 1-D by Coffman, Garey and Johnson [6] who showed that a modified first-fit algorithm, which we called *FFM*, is 2.788-competitive. This algorithm works by classifying items into large (size larger than $\frac{1}{2}$) and small ones (size $\frac{1}{2}$ or less), then using a dedicated bin for each large item and using first-fit for the small ones. The 2.788 bound is derived from the 1.788-competitive ratio if all items are small [6]. Recently, Chan, Wong and Yung [4] have shown a lower bound that there is no algorithm better than 2.5-competitive and this is further improved to $8/3 \simeq 2.666$ very recently [19] .

Multi-dimensional dynamic bin packing has been studied by Epstein and Levy [11]. They gave a 2×3.5^d-competitive algorithm for d-D dynamic bin packing, i.e., 24.5 for $d = 2$ and 85.75 for $d = 3$. They further presented algorithms specifically for $d = 2$ and $d = 3$ and claimed that they are 8.5754- and 35.346-competitive, respectively. They also gave lower bounds of $d + 1$ for general d, 3.70301 and 4.85383 for $d = 2$ and $d = 3$.

Resource augmentation [14] has also been studied in 1-D dynamic bin packing [4] and online static bin packing [1, 12, 10] for various dimensions. In this setting, the online algorithm can use larger bins than the optimal offline algorithm. For 1-D dynamic bin packing, it is shown that using bins of double size is both necessary and sufficient to achieve 1-competitiveness [4].

Our contribution. In this paper, we study multi-dimensional dynamic bin packing and give the following results (see Table 1 for a summary).

– For d-D where $d \geq 2$, we present a 3^d-competitive algorithm (Theorem 1).

[1] If rearrangement within a bin is not allowed, one can show that there is no constant competitive deterministic online algorithm.

– For 2-D and 3-D, we further improve the above general ratio. We give 7.788- and 22.788-competitive algorithms, respectively (Theorems 2 and 3).
– We consider resource augmentation and give conditions for the online algorithm to match the offline algorithm, i.e., 1-competitive (Corollary 1).

For the d-D result, our algorithm classifies items into large and small items. Roughly speaking, we show that large items can be handled as small items of lower dimensions, hence, we can focus on small items. The main idea is a testing procedure to check whether a new small item can be packed into existing bins. This naturally involves a space efficient offline single bin packing procedure, which is indeed an interesting problem by itself. Multi-dimensional NFDH (next-fit-decreasing-height) is a common strategy to achieve this; in particular, a formula has been given in [16,15] for the minimum total volume of d-D cubes (i.e., all sides are equal) that can be packed without overflowing a bin. However, there is no matching results for d-D boxes of general size. We devise a single bin packing procedure using a variant of NFDH, which instead of packing items using the whole bin space, reserves space to accommodate the new item and tries to repack existing items into a so called L-shape space. At first glance, reserving space may be too pessimistic, yet it can be shown that packing boxes using full space may perform only as good as the L-shape approach (we skip the details due to space limit). Using this new packing procedure, we show that the same formula in [15] can be obtained even for packing boxes of general sizes.

Notations and definitions. We now give a precise definition of the problem and the necessary notations for further discussion. In general, a d-D object (item or bin) is called a d-D *cube* if all sides have the same length; and d-D *box* otherwise. A packing configuration is said to be *feasible* if all items do not overlap and the packing in each bin does not exceed the boundary of the bin; otherwise, the packing is said to *overflow* and is *infeasible*.

In d-D dynamic bin packing, d-D items arrive and depart at arbitrary time. When an item arrives, it must be assigned to a unit-sized bin immediately so that the resulting packing is feasible. The item then resides in the assigned bin until it departs, i.e., migration is not allowed. Rearrangement of items within a bin is allowed upon item arrival or departure. The objective is to minimize the maximum number of bins used over all time.

For 2-D packing, we call the two dimensions width and height. For general d-D packing, we name the d dimensions $x_1, x_2, \cdots, x_d$ and denote the length of an item R along dimension x_i by $x_i(R)$. When the context is clear, we may also call x_d the height. In the d-D packing algorithms, we use the concept of projection of higher dimension item to lower dimension item. We say that an item is *projected along* x_d when the item is projected on the hyperplane of dimensions $x_1, x_2, x_3, \cdots, x_{d-1}$.

Several of our algorithms involve reserving some space for a new item and repacking existing items in a bin to check if the new item can be packed to this bin. If such a repacking is not feasible, it is understood that we restore the packing to the original configuration.

2 d-Dimensional Dynamic Bin Packing

In this section, we consider d-D dynamic bin packing for any $d \geq 2$ and present a 3^d-competitive online algorithm, called DynamicPack(d). Roughly speaking, when an item R arrives, Algorithm DynamicPack(d) checks if an existing bin can accommodate R by reserving some space for R and repacking existing items into the remaining space. If there is any bin that such repacking is feasible, R is assigned to this bin. If no such bin exists, open a new bin for R. In other words, the algorithm involves a repacking procedure that rearranges items in a given bin. We present this repacking procedure in Section 2.1 and the overall bin assignment algorithm in Section 2.2. Furthermore, the algorithm distinguishes between small and large items. An item R is said to be *small* if $x_i(R) \leq \frac{1}{2}$ for all $1 \leq i \leq d$, and *large* otherwise. In Section 2.2, we show that large items can be handled as small items of lower dimensions and so we can focus on small items.

2.1 Repacking Procedure for Small Items into a Single Bin

In this section, we present a procedure to repack small items into a single bin and give a formula for the minimum total volume of small items that can be packed in the bin. This procedure is invoked when a new item arrives, some space is reserved and existing items are repacked into the remaining space. We call the remaining space an L-shape bin. The repacking makes use of a variant of NFDH (next-fit-decreasing-height) approach.

Below we first formally define a d-D L-shape bin and describe some property of NFDH. Then we show that if a set of items cannot be packed in a d-D L-shape, the volume of these items is at least 2^d (Lemma 1).

d-D L-shape bin. For any $d \geq 2$, a *d-D L-shape bin* is a unit d-D bin with a corner removed: the corner removed is a d-D cube with all sides equal to $\frac{1}{2}$. For the sake of reference, we say that the cube removed is the "bottom left-most" corner of the bin. See Figure 1 (a) and (b) for examples. We note the following property about a unit d-D bin and a d-D L-shape bin.

Property 1. Consider a unit d-D bin and a d-D L-shape bin.

(i) Any small item can be packed into the cube that is removed from the unit bin to form the L-shape bin.
(ii) Take a layer of the unit bin with length h in dimension x_d and length 1 in all other dimensions, if we remove a box of length h in x_d and length $\frac{1}{2}$ in other dimensions from bottom left corner of the layer and project along dimension x_d, we obtain a $(d-1)$-D L-shape bin. See Figure 1 (c) for an example.

NFDH. Our procedure to pack into an L-shape bin makes use of the idea of NFDH. In general, the d-D version of NFDH first sorts the items in descending order of the length of dimension x_d, say $R_1, R_2, R_3, \cdots$ such that $x_d(R_i) \geq x_d(R_{i-1})$. Then the items are packed into layers aligned to dimension x_d. The

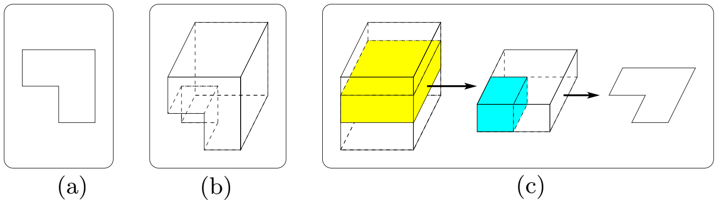


Fig. 1. (a) 2-D L-shape bin. (b) 3-D L-shape bin. (c) Removing a box at the bottom left corner from a layer of a 3-D bin and projecting on 2-D gives a 2-D L-shape bin.

first layer has a "height" equal to $x_d(R_1)$. Items, projected along x_d, are then packed into this layer using some $(d-1)$-D packing algorithm until a certain item, say R_j, cannot be packed. Then the next layer is constructed with height equals to $x_d(R_j)$. We then observe the following property about NFDH.

Property 2. Suppose NFDH has packed k layers with height $h_1 \geq h_2 \geq \cdots \geq h_k$.

(i) All the items in Layer-i have height at least h_{i+1}.
(ii) If the $(d-1)$-D packing algorithm guarantees each layer is packed with a $(d-1)$-D volume of at least V, then the total d-D volume of items in Layer-i $\geq h_{i+1}V$ and the total volume of all items in all layers $\geq \sum_{i=2}^{k} h_i V$.

Packing an L-shape bin. We now present the recursive procedure, called NFDH-LS(d), that packs into a d-D L-shape bin. As to be shown in Lemma 1, if NFDH-LS(d) cannot pack a set S of small items into a d-D L-shape bin, the total volume of S is more than $\frac{1}{2^d}$. We first describe the base case NFDH-LS(2).

Single-bin repacking procedure NFDH-LS(2): packing small items into a 2-D L-shape bin. We first sort the items in descending order of height. Items are packed into layers as follows (see Figure 2 for an example). *Layer-0* is the bottom square with height and width $\frac{1}{2}$. We pack the items (in order of height) to Layer-0 until a certain item, say Q, cannot be packed. Then the height of Q becomes the height of *Layer-1*. Layer-1 is divided into two equal partitions each with width $\frac{1}{2}$. We place Q into the first partition and continue packing the remaining items into the second partition until overflow. In general, the item that overflows from a layer is packed to the first partition of the next layer. The procedure returns whether all items can be packed in the L-shape bin (i.e., whether the last layer can be packed without overflow).

Single-bin repacking procedure NFDH-LS(d): packing small items into a d-D L-shape bin. We first sort the items in descending order of $x_d(\cdot)$. Items are then packed one by one in the sorted order into layers along dimension x_d (see Figure 3). *Layer-0* has height $\frac{1}{2}$ along dimension x_d. Projecting the layers along dimension x_d forms $(d-1)$-D L-shape bins (Property 1 (ii)). Items are packed with dimension x_d aligned to the height of the layer, and then projecting the items along dimension x_d, we use NFDH-LS($d-1$) to recursively pack the items into Layer-0. For Layer-i, if NFDH-LS($d-1$) reports feasible packing, we

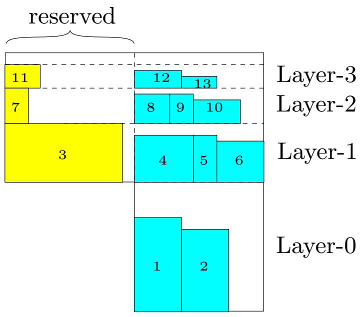


Fig. 2. Example for NFDH-LS(2). The number on the items is the order the items are packed. The item overflows from a layer (e.g., 3, 7 and 11) is packed in the first (reserved) partition of the next layer and the height of this item determines the height of the next layer. Packing then continues in the second partition.

try to include the next item into Layer-i; if packing is not feasible with the inclusion of a certain item R, we create Layer-(i+1) with $x_d(R)$ as the layer height along dimension x_d. We then reserve a box with height $x_d(R)$ along dimension x_d and length $\frac{1}{2}$ along all other dimensions; the box is at the bottom leftmost corner of the layer. Hence, projecting the layer along dimension x_d gives a (d−1)-D L-shape bin (Property 1 (ii)). We pack R into the reserved box and continue the process with the remaining items: adding them one by one and use NFDH-LS(d−1) recursively to pack into the (d−1)-D L-shape bin.

Whenever NFDH-LS(d−1) reports packing is not feasible with the next item included, a new layer is created. At the end, if all the layers constructed can be packed into the d-D L-shape bin without overflow, then NFDH-LS(d) reports a feasible packing, otherwise NFDH-LS(d) reports infeasible packing.

Lemma 1. *Let S be a set of items with sides bounded by $\frac{1}{2}$. If* NFDH-LS(d) *cannot pack S into a d-D L-shape bin, the total volume of S is more than $\frac{1}{2^d}$.*

Proof. We prove the lemma by induction on the dimension. Base case: when $d = 2$, Layer-0 together with the first item on Layer-1 has width greater than $\frac{1}{2}$. For $i > 0$, all items in Layer-i (except the first item in the reserved box) together with the first item in Layer-($i+1$) has width greater than $\frac{1}{2}$. All items in Layer-i has a height at least the height of Layer-($i+1$) (Property 2 (i)). If not all items can be packed, the total height of the layers (including the one that cannot fit) is more than 1. Since Layer-0 has height at most $\frac{1}{2}$, the total volume of all items is greater than $(1 - \frac{1}{2}) \times \frac{1}{2} = \frac{1}{2^2}$ (Property 2 (ii)).

Assume the lemma is true for dimension $d - 1$. For dimension d, we can use a similar argument as the base case to show that if not all items can be packed, the total height of all layers in dimension d is at least 1. Using the induction hypothesis, all items in each layer has volume greater than $\frac{1}{2^{d-1}}$. Hence, the total volume of all items is greater than $(1 - \frac{1}{2}) \times \frac{1}{2^{d-1}} = \frac{1}{2^d}$ (Property 2 (ii)).

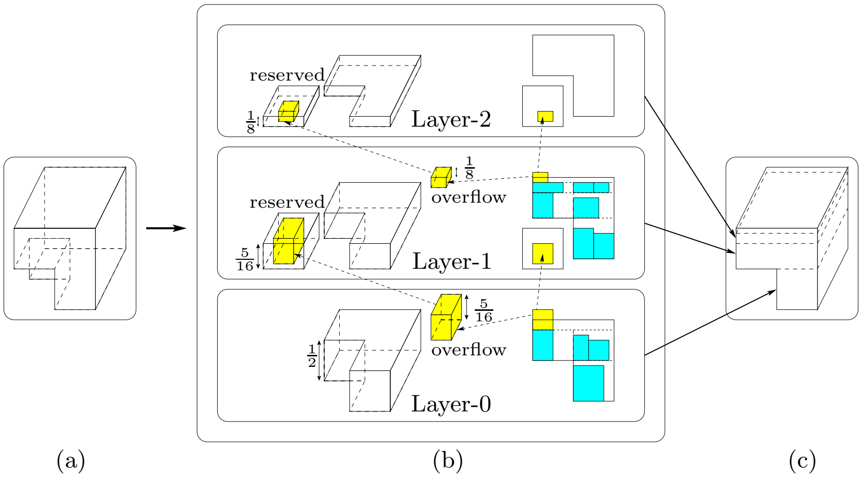


Fig. 3. Example for NFDH-LS(3). (a) 3-D L-shape bin. (b) In each layer, the 2-D L-shape on the right is the projection of the layer. The overflow item is packed in the reserved box in the next layer and the height of this item determines the height of the next layer. (c) The layers as shown in the 3-D L-shape.

The results in this section can be generalized as follows. Let S be a set of items with sides bounded by $\frac{1}{k}$. There is a packing algorithm such that if S cannot be packed into a d-D L-shape bin, then the total volume of S is more than $(1-\frac{1}{k})^d$. As mentioned in the introduction, this bound has been shown for packing d-D cubes and we show that it is also true for packing d-D boxes.

2.2 Bin Assignment Algorithm

We now present a 3^d-competitive online dynamic bin packing algorithm. We first present an algorithm AFReserveNFDH(d) for packing small items (AF for any-fit) and then an algorithm DynamicPack(d) for arbitrary items.

Bin assignment algorithm AFReserveNFDH(d) for small items. When an item R with sides bounded by $\frac{1}{2}$ arrives, we consider every bin in turn to reserve for R the bottom leftmost cube of all sides $\frac{1}{2}$ and check if all the existing items in that bin can be packed into the d-D L-shape bin using NFDH-LS(d). If NFDH-LS(d) finds a bin that there is feasible packing, R is packed in the reserved bottom leftmost corner of that bin and the other items are packed in the way NFDH-LS(d) packs them. If there is no such existing bin, open a new bin for R. Lemma 2 is a direct consequence of Lemma 1.

Lemma 2. *Algorithm* AFReserveNFDH(d) *is* 2^d*-competitive for packing d-D items with sides bounded by* $\frac{1}{2}$.

Algorithm DynamicPack(d) for arbitrary sized items. We distinguish items based on whether each dimension x_i is larger than $\frac{1}{2}$ or at most $\frac{1}{2}$, and classify the items into 2^d classes. Consider items of the same class. If $x_i(\cdot) > \frac{1}{2}$ in this class, then in any packing, a line drawn parallel to dimension x_i intercepts at most one item in that class. Suppose there are z dimensions with length at most $\frac{1}{2}$ and $(d - z)$ dimensions larger than $\frac{1}{2}$. Then items in this class can be considered as z-D small items. We use AFReserveNFDH(z) to pack the items by projecting along all the dimensions x_i such that $x_i(\cdot) > \frac{1}{2}$. These items will be packed to align to the facets for those dimensions with $x_i(\cdot) > \frac{1}{2}$.

For the class where all dimensions are larger than $\frac{1}{2}$, we pack each in a separate bin (no two such items can be packed into the same bin by any packing).

Theorem 1. *Algorithm* DynamicPack(d) *is* 3^d*-competitive.*

Proof. For each z, we run a couple of Algorithm AFReserveNFDH(z) and the number is at most $\binom{d}{z}$. By Lemma 2, the competitive ratio of DynamicPack(d) is at most the sum over all ratios of AFReserveNFDH(z), i.e., $\sum_{z=1}^{d} \binom{d}{z} 2^z = 3^d$.

3 Two- and Three-Dimensional Dynamic Bin Packing

In this section, we improve the results in Section 2 for $d = 2$ and 3 by more careful classification and assignment of items into bins. Similar to Section 2, the algorithms make use of reserve-and-repacking procedures to check if a new item can be assigned to an existing bin.

3.1 Two-Dimensional Dynamic Bin Packing

Repacking Procedures. The procedures described in this section will be reused later and so they are stated in a more general form. We assume we are given a bin with width u and height v throughout Section 3.1. Before we present our two-dimensional repacking procedures, we first note the result by Steinberg [18] for static bin packing. In particular, we will use the following lemma which is implied by Theorem 1.1 in [18].

Lemma 3 ([18]). *Given a bin with width u and height v, if all items have width at most $\frac{u}{2}$ and height at most v, then any set of these items with total area at most $\frac{uv}{2}$ can fit into the same bin by using Steinberg's algorithm.*

While Steinberg's result gives a condition for a set of items to be fit into a bin, we also need a slightly different condition which bounds the area of existing items in a bin if a new item cannot be packed into the bin. The latter is a typical notion required when the objective is to minimize the number of bins. Below we describe two such packing procedures.

Procedure 2DRepackNarrow: for items with width in $(0, \frac{u}{3}]$ and height in $(0, v]$. We divide each bin into two partitions both with height v, and the first one with width $\frac{u}{3}$, second with width $\frac{2u}{3}$. Note that any item can fit into the first partition. When an item R arrives, we reserve the first partition for R, and repack existing items using the second partition as a bin of width $\frac{2u}{3}$ and height v using Lemma 3. If repacking is feasible, the packing configuration is returned as solution.

Procedure 2DRepackMedium: for items with width in $(\frac{u}{3}, \frac{u}{2}]$ and height in $(0, v]$. Note that since the items have width more than $\frac{u}{3}$, in any packing including the optimal one, at most two items can be packed side by side along the width. We divide each bin into two equal partitions each with width $\frac{u}{2}$ and height v. When a new item R arrives, we reserve the first partition for R and stack the existing items (in arbitrary order) along the height of the second partition. Similar to 2DRepackNarrow, if repacking is feasible, the packing configuration is returned as solution.

Lemma 4. *Consider a bin with width u and height v.*

(i) Consider items with width in $(0, \frac{u}{3}]$ and height in $(0, v]$. If 2DRepackNarrow *returns an infeasible repacking, the total area of the existing items in the bin is more than $\frac{uv}{3}$.*

(ii) Consider items with width in $(\frac{u}{3}, \frac{u}{2}]$ and height in $(0, v]$. If 2DRepackMedium *returns an infeasible repacking, the total height of the existing items in the bin is more than v.*

Proof. (i) This means that existing items cannot be packed into the second partition (the larger one) of the bin. By Lemma 3, the total area of these items is more than $\frac{1}{2} \times \frac{2uv}{3} = \frac{uv}{3}$; otherwise, the items can be packed in the partition. (ii) This means the total height of the existing items exceed the height of the second partition, i.e., v.

Bin Assignment Algorithm. Using the above two procedures, we present an algorithm called 2DDynamicPack, which classifies items into three classes: narrow, medium-wide and wide according to their width. An item is said to be *narrow* if its width is in $(0, \frac{1}{3}]$; *medium-wide* if in $(\frac{1}{3}, \frac{1}{2}]$; and *wide* if in $(\frac{1}{2}, 1]$.

Algorithm 2DDynamicPack. Classify items into narrow, medium-wide, and wide as they arrive. Items of the same class are assigned to bins independently of other classes.

- **Narrow items.** When a narrow item R arrives, find any bin that existing narrow items in the bin can be repacked using procedure 2DRepackNarrow into the larger partition of the bin and pack R into the smaller partition of the bin. If no such bin exists, open a new bin for R.
- **Medium-wide items.** When a medium-wide item R arrives, find any bin that existing medium-wide items in the bin can be repacked using procedure 2DRepackMedium into the second partition of the bin and pack R into the first partition of the bin. If no such bin exists, open a new bin for R.
- **Wide items.** Items are packed so that the width of the item is aligned to the width of the bin, and then ignoring the width of the items, use the 1D algorithm FFM to pack according to the height of the items.

Theorem 2. *Algorithm* 2DDynamicPack *is* 7.788*-competitive.*

Proof. Let OPT denote the maximum number of bins used by the optimal offline algorithm, and n, n_1, n_2, and n_3 be that used by 2DDynamicPack for all, narrow, medium-wide and wide items, respectively, i.e., $n \le n_1 + n_2 + n_3$. When 2DDynamicPack opens the n_1-th bin for a new narrow item, the total area of all items is more than $\frac{n_1-1}{3}$ (Lemma 4 (i)). Hence, we have $OPT \ge \lfloor \frac{n_1-1}{3} \rfloor + 1 \ge \frac{n_1}{3}$. When 2DDynamicPack opens the n_2-th bin for a new medium-wide item, by Lemma 4 (ii), the total height of all items is more than $n_2 - 1$. Since the width of these items is more than $\frac{1}{3}$, in any packing of the items, every horizontal line drawn intercepts at most two items and hence, the total number of bins used is at least $\lfloor \frac{n_2-1}{2} \rfloor + 1$, i.e., $OPT \ge \frac{n_2}{2}$. Finally, for wide items, using FFM means $OPT \ge \frac{n_3}{2.788}$. In total, $n \le n_1 + n_2 + n_3 \le 7.788 OPT$ and the lemma follows.

3.2 Upper Bounds for 3-D Dynamic Bin Packing

Recall that the three dimensions are x_1, x_2 and x_3 and the length of an item R along dimension x_i is denoted as $x_i(R)$. Our algorithm, called 3DDynamicPack, classifies the items into four classes according to x_i of the items. An item R is said to be in

- **Class-1** if $x_1(R) > \frac{1}{2}$;
- **Class-2** if $x_1(R) \le \frac{1}{2}$ and $x_2(R) > \frac{1}{2}$;
- **Class-3** if $x_1(R) \le \frac{1}{2}$ and $\frac{1}{3} < x_2(R) \le \frac{1}{2}$; and
- **Class-4** if $x_1(R) \le \frac{1}{2}$ and $x_2(R) \le \frac{1}{3}$.

Classes 1 and 2 can be handled rather straightforwardly by using 2-D packing algorithm (details to be given later). Classes 3 and 4 need more attention. We describe two repacking procedures for handling these two classes.

Procedure 3DRepackClass3: for items with $x_1(R) \le \frac{1}{2}$ and $\frac{1}{3} < x_2(R) \le \frac{1}{2}$. Similar to an observation made in 2DRepackMedium, in any packing including the optimal one, at most two items can be packed side by side along dimension x_2. We divide a bin into two equal partitions along dimension x_2 both with length $\frac{1}{2}$ (the length along dimension x_1 and x_3 remains 1). When a new item R arrives, we reserve the first partition for R and check if existing items can be packed into the second partition: project existing items along dimension x_2, repack them by Lemma 3, treating them as rectangles with dimension x_1 as width and x_3 as height. If repacking is feasible, the packing configuration is returned as solution.

Procedure 3DRepackClass4: for items with $x_1(R) \le \frac{1}{2}$ and $x_2(R) \le \frac{1}{3}$. We first sort the existing items in descending order of $x_1(R)$. Items are then packed into layers constructed along dimension x_1, in an NFDH manner (next-fit-decreasing-height). The *first layer* has length $\frac{1}{2}$ along dimension x_1. Project the items along dimension x_1 and treating x_2 as width x_3 as height, we pack items into this layer using 2DRepackNarrow, i.e., create two partitions with width $\frac{1}{3}$ and $\frac{2}{3}$, and pack to the larger partition. If some item Q cannot be packed into this layer, we then create *a new layer* with length $x_1(Q)$ along dimension x_1 and pack Q into the first partition of this layer. Then we use 2DRepackNarrow similarly to pack items to the second partition. The item that overflows from a layer will be packed into the first partition of the next layer. Repeat this until all existing items are packed or a new layer overflows dimension x_1 of the bin. If repacking is feasible, the packing configuration is returned as solution.

Algorithm 3DDynamicPack. Classify the items as they arrive into the four classes defined above. Items in each class are assigned to bins independently of other classes.

– **Class-1:** Project the items along dimension x_1 treating them as rectangles, and then pack the items using Algorithm 2DDynamicPack.
– **Class-2:** Project each item along dimension x_2 and further classify the items into two sub-classes with $0 < x_1 \leq \frac{1}{3}$ and $\frac{1}{3} < x_1 \leq \frac{1}{2}$. For items of the first sub-class, find a bin such that existing items can be repacked using 2DRepackNarrow. If there is no such bin, open a new bin for the new item. Similarly for the second sub-class, use 2DRepackMedium.
– **Class-3:** When a new item R of Class-3 arrives, find any bin that existing items can be repacked using procedure 3DRepackClass3 into the second partition and pack R into the first partition. If there is no such bin, open a new bin for R.
– **Class-4:** When a new item R of Class-4 arrives, find any bin that existing items can be repacked using procedure 3DRepackClass4 and pack R into the reserved space in the first layer. If there is no such bin, open a new bin for R.

Theorem 3. 3DDynamicPack *is* 22.788*-competitive.*

Proof. Let OPT be the maximum number of bins used by the optimal offline algorithm, n, n_1, n_2, n_3, and n_4 be that by 3DDynamicPack for all, Classes 1, 2, 3 and 4 items, respectively, i.e., $n \leq n_1 + n_2 + n_3 + n_4$. When 3DDynamicPack opens the n_1-th bin for a new Class-1 item, by Theorem 2, we have $OPT \geq \frac{n_1}{7.788}$.

Let $n_{2,1}$ and $n_{2,2}$ be the maximum number of bins used for the two sub-classes of Class-2 items, i.e., $n_2 \leq n_{2,1} + n_{2,2}$. By Lemma 4 (i) and (ii) and a similar argument as Theorem 2, we have $OPT \geq \frac{n_{2,1}}{3}$ and $OPT \geq \frac{n_{2,2}}{2}$. So, $OPT \geq \frac{n_2}{5}$.

When 3DDynamicPack opens the n_3-th bin for a new Class-3 item, by Lemma 3, the total area of all Class-3 items is more than $\frac{n_3-1}{2}$. Furthermore, the length of these items along dimension x_2 is more than $\frac{1}{3}$, in any packing of the items, every line drawn parallel to dimension x_2 intercepts at most two items. Hence, $OPT \geq \lfloor \frac{n_3-1}{4} \rfloor + 1 \geq \frac{n_3}{4}$.

When 3DDynamicPack opens the n_4-th bin for a new Class-4 item, by Lemma 4 (i), in the repacking of each bin, the total area of all Class-4 items in the second partition in each layer and the first partition in the next layer is more than $\frac{1}{3}$. Furthermore, the height of all items in each layer is at least the height of the next layer. Since existing items cannot be repacked into the same bin using the procedure 2DRepackNarrow, the total height of all layers is more than 1 and that of all but the first layer is more than $\frac{1}{2}$ (since the first layer has height $\frac{1}{2}$). Therefore, the total volume of existing items in each bin is more than $\frac{1}{6}$ and the total volume of all existing items is more than $\frac{n_4-1}{6}$. Hence, $OPT \geq \lfloor \frac{n_4-1}{6} \rfloor + 1 \geq \frac{n_4}{6}$. In summary, we have $n \leq n_1 + n_2 + n_3 + n_4 \leq 22.788OPT$.

4 Concluding Remarks

In this paper, we have studied multi-dimensional dynamic bin packing. We have presented a general competitive ratio for $d \geq 2$ and improved the ratio further for $d = 2$ and $d = 3$. So far the competitive ratio for multi-dimensional bin packing (both static and dynamic, as well as for both cube and box) grows exponentially with d. Yet there is no matching lower bound that also grows exponentially with d. It is believed that this is the case [8] and any such lower bound would be of great interest. Furthermore, the general upper bound for d-dimension is usually worse than the corresponding 2-D or 3-D upper bound when substituting $d = 2$ or $d = 3$. It would be desirable to have d-dimensional packing algorithm that have a more accurate formula to reflect the ratio for lower dimension. As for lower dimension, an obvious open question is to close the gap between the upper bound and lower bound.

Another direction is to consider resource augmentation in which the online algorithm can use d-dimensional bins of size $s_1 \times s_2 \times \cdots \times s_d$ for $s_i \geq 1$ while the optimal offline algorithm uses unit-sized bins. As a first step, we give some simple conditions for the online algorithm to match the performance of the optimal offline algorithm, i.e., 1-competitive. The results here are obtained directly from those in Sections 2 and 3.

Corollary 1. *Consider the optimal offline algorithm that uses unit sized bins.*

(i) *For* 2*-D, there is a* 1*-competitive online algorithm using bins of size* 3×1.
(ii) *For d-D, there is a* 1*-competitive online algorithm using bins of size* $\{2\}^d$.
(iii) *For d-D, no online algorithm is* 1*-competitive using bins of size* $\{2 - \epsilon\}^d$ *for any* $\epsilon > 0$.

References

1. Bansal, N., Caprara, A., Sviridenko, M.: Improved approximation algorithms for multidimensional bin packing problems. In: FOCS 2006, pp. 697–708 (2006)
2. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
3. Chan, W.T., Lam, T.W., Wong, P.W.H.: Dynamic bin packing of unit fractions items. Theoretical Computer Science (TCS) 409(3), 521–529 (2008)
4. Chan, W.T., Wong, P.W.H., Yung, F.C.C.: On dynamic bin packing: An improved lower bound and resource augmentation analysis. Algorithmica 53(2), 172–206 (2009)
5. Coffman Jr., E.G., Galambos, G., Martello, S., Vigo, D.: Bin packing approximation algorithms: Combinatorial analysis. In: Du, D.Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization (1998)

6. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Dynamic bin packing. SIAM Journal on Computing Dynamic bin packing 12(2), 227–258 (1983)
7. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: A survey. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-Hard Problems, pp. 46–93. PWS Publishing (1996)
8. Coppersmith, D., Raghavan, P.: Multidimensional on-line bin packing: Algorithms and worst-case analysis. Operations Research Letters 8(1), 17–20 (1989)
9. Csirik, J., Woeginger, G.J.: Online packing and covering problems. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms—The State of the Art, pp. 147–177. Springer, Heidelberg (1996)
10. Csirik, J., Woeginger, G.J.: Resource augmentation for online bounded space bin packing. Journal of Algorithms 44(2), 308–320 (2002)
11. Epstein, L., Levy, M.: Dynamic multi-dimensional bin packing. Manuscript (2006)
12. Epstein, L., van Stee, R.: Online bin packing with resource augmentation. Discrete Optimization 4(3-4), 322–333 (2007)
13. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
14. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. of ACM 47(4), 617–643 (2000)
15. Kohayakawa, Y., Miyazawa, F.K., Raghavan, P., Wakabayashi, Y.: Multidimensional cube packing. Algorithmica 40(3), 173–187 (2004)
16. Meir, A., Moser, L.: On packing of squares and cubes. J. Comb. Theory Series A 5(2), 116–127 (1968)
17. Schiermeyer, I.: Reverse-fit: A 2-optimal algorithm for packing rectangles. In: ESA, pp. 290–299 (1994)
18. Steinberg, A.: A strip-packing algorithm with absolute performance bound 2. SIAM Journal on Computing 26(2), 401–409 (1997)
19. Wong, P.W.H., Yung, F.C.C.: A 2.666-lower bound on dynamic bin packing. Manuscript

Author Index