Klaus Jansen
Roberto Solis-Oba (Eds.)

LNCS 6534

# Approximation and Online Algorithms

8th International Workshop, WAOA 2010
Liverpool, UK, September 2010
Revised Papers

Springer

Lecture Notes in Computer Science 6534

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

Klaus Jansen  Roberto Solis-Oba (Eds.)

# Approximation and Online Algorithms

8th International Workshop, WAOA 2010
Liverpool, UK, September 9-10, 2010
Revised Papers

Springer

Volume Editors

Klaus Jansen
University of Kiel, Department of Computer Science
Olshausenstrasse 40, 24118 Kiel, Germany
E-mail: kj@informatik.uni-kiel.de

Roberto Solis-Oba
The University of Western Ontario, Department of Computer Science
London, Ontario N6A 5B7, Canada
E-mail: solis@csd.uwo.ca

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-18317-1 e-ISBN 978-3-642-18318-8
DOI 10.1007/978-3-642-18318-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2010942454

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

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



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

Printed on acid-free paper

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

# Organization

## Program Co-chairs

| | |
|---|---|
| Klaus Jansen | University of Kiel, Germany |
| Roberto Solis-Oba | University of Western Ontario, Canada |

## Program Committee

| | |
|---|---|
| Klaus Jansen (Co-chair) | University of Kiel, Germany |
| Roberto Solis-Oba (Co-chair) | University of Western Ontario, Canada |
| Evripidis Bampis | University of Evry, France |
| Jianer Chen | Texas A&M University, USA |
| Jose Correa | Universidad de Chile, Chile |
| Khaled Elbassioni | Max Planck Institut für Informatik, Germany |
| Rudolf Fleischer | Fudan University, China |
| Thomas Erlebach | University of Leicester, UK |
| Martin Fürer | The Pennsylvania State University, USA |
| Christos Kaklamanis | University of Patras, Greece |
| Jochen Könemann | University of Waterloo, Canada |
| Stefano Leonardi | Sapienza University of Rome, Italy |
| Alejandro López-Ortiz | University of Waterloo, Canada |
| Monaldo Mastrolilli | IDSIA Lugano, Switzerland |
| Hadas Shachnai | Israel Institute of Technology, Israel |
| Martin Skutella | TU Berlin, Germany |
| Clifford Stein | Columbia University, USA |
| Denis Trystram | Grenoble Institute of Technology, France |

## Additional Referees

Sivan Albagli
Susanne Albers
Eric Angel
Stavros Athanassopoulos
Nikhil Bansal
Vincenzo Bonifaci
Marin Bougeret
Mohamed Slim Bouguerra
David Bunde
Chandra Chekuri
Ning Chen
Janka Chlebikova
Daniel Cordeiro
Andy Curtis
Reza Dorrigiv
Christoph Dürr
Stephane Durocher
Leah Epstein
Bruno Escoffier
Samuel Fiorini
Mohamoud Fouz
Robert Fraser
Hiroshi Fujiwara
Leslie Ann Goldberg

Laurent Gourves
Fabrizio Grandoni
Martin Groß
Inge Li Gørtz
Angele Hamel
Martin Hoefer
Riko Jacob
Panagiotis Kanellopoulos
Nikos Karanikolas
Hans Kellerer
Alexandra Kolla
Ravishankar Krishnaswamy
Sven Krumke
Piotr Krysta
Raghav Kulkarni
Maria Kyropoulou
Van Bang Le
Lap-Kei Lee
Dimitris Letsios
Asaf Levin
Martin Matamala
Vardges Melkonian
Nikola Milosavljevic
Vahab Mirrokni
Jerome Monnot
Luca Moscardelli
Benjamin Moseley
Gregory Mounie
Alantha Newman
Kim Thang Nguyen
Zeev Nutov
Christina Otte
Andrea Pacifici
Evi Papaioannou
Ojas Parekh
Fanny Pascual
David Pritchard
Kirk Pruhs
Lars Prädel
Claude-Guy Quimper
Rajiv Raman
Thomas Rothvoss
Alejandro Salinger
Ulrich M. Schwarz
Rene Sitters
Alexander Souza
Georgios Stamoulis
Sebastian Stiller
Martin Strauss
Zoya Svitkina
Tami Tamir
Nicolas Thibault
Marc Uetz
Rob van Stee
Anke van Zuylen
José Verschae
Tjark Vredeveld
Andreas Wiese
Prudence W.H. Wong

# Table of Contents

## WAOA 2010

# Strategic Multiway Cut and Multicut Games*

Elliot Anshelevich, Bugra Caskurlu, and Ameya Hate

Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY

**Abstract.** We consider cut games where players want to cut themselves off from different parts of a network. These games arise when players want to secure themselves from areas of potential infection. For the game-theoretic version of Multiway Cut, we prove that the price of stability is 1, i.e., there exists a Nash equilibrium as good as the centralized optimum. For the game-theoretic version of Multicut, we show that there exists a 2-approximate equilibrium as good as the centralized optimum. We also give poly-time algorithms for finding exact and approximate equilibria in these games.

## 1 Introduction and Model

Networked systems for transport, communication, and social interaction have contributed to all aspects of life by increasing economic and social efficiency. However, increased connectivity also gives intruders and attackers better opportunities to maliciously spread in the network, whether the spread is of disinformation, or of contamination in the water supply [26]. Anyone participating in a networked system may therefore desire to undertake appropriate security measures in order to protect themselves from such malicious influences.

We introduce a *Network Cutting Game*, which is a game-theoretic framework where a group of self-interested players protect vertices that they own by cutting them off from parts of the network that they find untrustworthy. Cutting an edge should not be interpreted as destroying a part of the network: instead it can correspond to taking security measures on that edge such as placing sentries on lines of communication. These notions are applicable in areas such as airline security. Consider a situation where country A requires extra security screening of passengers or cargo from country B. Due to the networked multi-hop structure of international air travel, the optimal locations for carrying out such screenings may lie somewhere in between the two countries. In general, the goal of each player is to make sure that nothing can get to it from an "untrustworthy" part of the network without passing through an edge with installed security measures.

The purpose of each player is to protect themselves while spending as little money as possible for their security. We investigate the efficiency of the security actions taken by a group of agents by studying the *price of anarchy* and the *price of stability* — the ratios between the costs of the worst and best Nash

* This work supported in part by NSF CCF-0914782.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 1–12, 2011.

equilibrium[1] respectively, and that of the globally optimal solution. While the price of anarchy can be extremely high (see Section 5), we prove nice bounds on the price of stability.

*Game Definition.* We now formally define the *Network Cutting Game* as follows. We are given an undirected graph $G = (V, E)$, and a set $P$ of $k$ players. Each player $i$ corresponds to a single *player node* in the graph $G$, which we will also denote by $i$. The strategy set of every player $i$ is $2^E$; a strategy $S_i \subseteq E$ of player $i$ is the set of edges that player $i$ will cut. The outcome of the game is $G_S$, which is a subgraph of $G$ obtained by removing the edges of $\bigcup_i S_i$.

The objective of each player $i$ is to protect her node $i$ from a given subset of nodes $T_i$ of $V$. We say that player $i$ *satisfies her cut requirement* if $i$ is disconnected from all nodes of $T_i$ in $G_S$. Every player wants to satisfy her cut requirement, but also wants to minimize the number of edges she cuts, which we denote by $|S_i|$. If a player $i$ does not satisfy her cut requirement, she faces a penalty cost of $\beta_i$. We can think of $\beta_i$ as the maximum number of edges that $i$ would be willing to cut in order to satisfy her cut requirement. We conclude the definition of our game by defining the cost function for each player $i$ as:

- $cost(i) = |S_i|$ if player $i$ satisfies her cut requirements,
- $cost(i) = |S_i| + \beta_i$ otherwise.

*Nash Equilibrium and OPT.* A pure Nash equilibrium (NE) of the *Network Cutting Game* is a strategy vector $S = (S_1, \ldots, S_k)$ such that no player $i$ has an incentive for unilateral deviation from her strategy, i.e., no player can reduce her cost by changing her strategy from $S_i$ to another strategy $S_i'$, assuming all other players stay with their existing strategies. Notice that in an equilibrium no player will cut more than $\beta_i$ edges, since this player could change her strategy to $S_i' = \emptyset$, and reduce her cost to at most $\beta_i$. By the same reasoning, all players that do not satisfy their cut requirements will play $S_i = \emptyset$ at equilibrium. Therefore, in a Nash equilibrium, all edges must be cut by players that satisfy their cut requirements.

We analyze the quality of Nash equilibrium solutions by comparing them with the cost of the socially optimal solution, which we refer to as OPT. The socially optimal solution is an outcome of the *Network Cutting Game* that minimizes the total cost of all the players (equivalently, maximizes social welfare). Let $Q(S)$ denote the set of players whose cut requirements are not satisfied in $G_S$. The cost of solution $S$ is given by:

$$\text{cost}(S) = \left| \bigcup_i S_i \right| + \sum_{j \in Q(S)} \beta_j,$$

[1] Recall that a (pure-strategy) Nash equilibrium is a solution where no single player can switch her strategy and become better off, given that the other players keep their strategies fixed.

which is the same as $\sum_i cost(i)$ since for any equilibrium solution we can assume that all sets $S_i$ are disjoint. When all $\beta_i$ are large, OPT is exactly the smallest set of edges that satisfies all the cut requirements.

*Our Results.* In general, the *Network Cutting Game* may not have any pure Nash equilibrium[22], and the price of anarchy for even very simple instances can be as large as the number of players. Our main results are about the existence and computation of cheap, exact, and approximate equilibria for several important special cases of the *Network Cutting Game*. By an $\alpha$-approximate Nash equilibrium, we mean that no player in such a solution can reduce her cost by a factor of more than $\alpha$ by deviating. Specifically, we consider the following special cases of cut requirements.

In the *Single-Source Network Cutting Game,* each player $i$ wants to disconnect her node from a common node $t$, i.e., $T_i = \{t\}$ for all players $i$.

In the *Network Multiway Cut Game,* each player $i$ wants to disconnect her node from the nodes of all other players, i.e., $T_i = P \backslash \{i\}$ for all players $i$. Notice that when $\beta_i$ for all players is large enough, the socially optimal solution is exactly the *Minimum Multiway Cut*.

In the *Network Multi-Cut Game,* each player $i$ wants to disconnect her node $i$ from some specific node $t_i$. In other words, this is the case where $|T_i| = 1$ for all players. When $\beta_i$ for all players is large enough, the socially optimal solution is exactly the *Minimum Multicut* for the set of pairs $(i, t_i)$.

Our main results are as follows:

- In Section 2, we study the *Single Source Network Cutting Game* and show that there always exists a Nash equilibrium as cheap as OPT, i.e., the price of stability is 1. Furthermore, that Nash equilibrium can be computed in polynomial time. This analysis easily generalizes to the case when $T_i$ are not singleton sets, but $T_i = T_j$ for all $i, j$.
- In Section 3, we study the *Network Multiway Cut Game* and show that there always exists a Nash equilibrium as good as OPT, i.e., the price of stability is 1. Given an approximate solution to Multiway Cut (for example, a 1.34-approximation found by [23]), we show how to compute a Nash equilibrium with the same or smaller cost than this solution in polynomial time.
- In Section 4, we study the *Network Multi-Cut Game* and show that there always exists a 2-approximate Nash equilibrium as cheap as OPT.

In Section 5, we consider the above games on graphs with non-uniform edge costs, i.e., where each edge has some fixed cost $w(e)$ to cut it. If every edge must be bought entirely by a single player, the resulting Nash equilibria can be expensive. We show that if the players are allowed to pay for cutting only a portion of an edge, the above results extend to non-uniform edge costs.

*Related Work.* An extensively studied game related to cuts is the max-cut game [10,14,18]. In this game players are forming a bipartition on the graph, where the utility of each player is the total weight of the edges of the cut incident to her. The Max-cut game always admits Nash equilibria since it is a potential game [29] and it is recently shown that it always admits strong Nash equilibria [16].

A contrasting approach to the Network Cutting Game are Network Formation Games [1,13,27], where a set of players is collectively trying to built a network, i.e., players want to connect some subset of nodes, rather than cutting them apart. Players connect nodes to each other by building edges and sharing the cost of the edges. The most relevant network formation game to our model is the Connection Game [4,6,12], where each player $i$ wants to connect a pair of nodes to each other. Existence and quality of equilibria depend on the cost-sharing method used [9]. The cost-sharing scheme used for the Connection Game in [6] is commonly referred to as 'arbitrary-sharing' [3,19,20,21], which will be explained in detail in Section 5. Another popular cost-sharing scheme in Network Formation Games is commonly referred to as 'fair-sharing', which will also be explained in detail in Section 5. The Connection Game with fair sharing was first studied in [5] and later addressed in [8,15], among others.

There have been many interesting applications of game theory to network security models (eg. [17,28,30]). A notion of *interdependent security* (IDS) games was introduced by Kunreuther and Heal [25]. In these games, the decision to adopt a security measure by a player affects other players in the network. An algorithm for finding an approximate equilibrium on this model was later given by Kearns and Ortiz [24]. Work on a similar model by Aspnes et al [2] deals with players immunizing their nodes against infections that can spread in the network. There are several major differences between these models and the games we consider: (i) players in IDS games can only immunize themselves, while in our games players are allowed to add security to different parts of the network, and (ii) these models consider that an attack can occur randomly anywhere in the network, while in our games the players are trying to protect themselves from specific areas of the network which might be different for different players.

Finally, some of our questions are related to ones studied by Engelberg et al. [11], who look at bugeted versions of cut problems like Multiway Cut, Multicut, and k-cut. This work considers centrally optimal solutions, not equilibria, and while our values $\beta_i$ can be thought of as budgets, they are budgets for individual players, not global budgets on the cost of the network.

## 2 Preliminaries and Basic Results

Given a graph $G = (V, E)$, and two disjoint subsets $A \subseteq V$ and $B \subseteq V$, denote by $M(G, A, B)$ the set of minimum-size $A - B$ cuts. By an $A - B$ cut, we mean a set of edges $E'$ such that $A$ and $B$ are disconnected by removing $E'$ from $G$. Denote by $m(G, A, B)$ the size of a minimum $A - B$ cut in graph $G$. We will also abuse notation slightly, and for a strategy vector $S = (S_1, \ldots, S_k)$, we will let $S_{-i}$ denote $\cup_j S_j - S_i$, i.e., the set of edges cut by players other than $i$.

**Proposition 1.** *A strategy vector $S = (S_1, \ldots, S_k)$ of the Network Cutting Game is a Nash equilibrium if and only if $S_i$ are pairwise disjoint, and*

- $|S_i| = m(G - S_{-i}, i, T_i) \leq \beta_i$ *if cut requirement of player $i$ is satisfied on $G_S$,*
- $S_i = \emptyset$ *and* $m(G - S_{-i}, i, T_i) \geq \beta_i$ *otherwise.*

Due to lack of space most of our proofs appear in the full version of the paper. To add intuition about the structure of Nash equilibria, notice that when $\beta_i$ are large (say $\geq |E|$) for all players, then the above proposition says that a solution $S$ is a Nash equilibrium exactly when $|S_i| = m(G - S_{-i}, i, T_i)$ for all players.

### 2.1 Single-Source Network Cutting Game

In this section, we study the *Single Source Network Cutting Game*. Since for single-source cut games, the LP to compute OPT has an integrality gap of 1, the theorem below also follows from [19]. This analysis easily generalizes to the case when $T_i$ are not singleton sets, but $T_i = T_j$ for all $i, j$.

**Theorem 1.** *For the Single Source Cutting Game, Nash equilibrium is guaranteed to exist and the price of stability is* 1.

**Proof Sketch.** The proof idea is quite simple: consider a max-flow from a super-source $s$ to $t$, with $s$ connected to each player node $i \in P$ with an edge of capacity $\beta_i$. Then it is easy to see that OPT is an $s - t$ min-cut, and so is saturated by this flow. We set $S_i$ to be the set of edges of OPT that is used by the flow passing through node $i$. Then we can use Proposition 1 to show that this results in a Nash equilibrium. The full proof of this theorem appears in the full version. ■

Notice that the equilibrium constructed in the proof can be easily found in poly-time using standard flow algorithms.

## 3 Network Multiway Cut Game

In the Network Multiway Cut Game (NMCG) each player $i$ wants to disconnect itself from every other player provided that the cost of cutting edges does not exceed the player's budget $\beta_i$. When players all players have large $\beta_i$, the socially optimal solution is the well-studied Multiway Cut (MWC) problem. For the sake of simplicity in exposition, we will derive the main results assuming $\beta_i$ to be infinitely large. In the full version we show how to extend these results to the case where values of $\beta_i$ are bounded. Notice that in this scenario where $\beta_i = \infty$, each player ***must*** disconnect itself from every other player node.

*Proof Technique.* As we saw in Section 2.1 for the Single-Source Network Cutting Game, we can use flows to determine the payments for players in the game. One approach would be to create an analogous multi-commodity flow for the Network Multiway Cut Game, and assign payments based on this flow. Unfortunately, this approach *does not* yield the desired results, since for the payments to be stable, we would need that the size of the maximum flow is equal to the size of the minimum cut, which does not hold for multi-commodity flow problems.

Instead, in our approach we use the key observation proven in Lemma 1 to construct a *single-commodity* flow that lets us determine stable payments for the optimum solution. It should be noted that this construction does not find the

minimum multiway cut, but only calculates stable payments for this cut if it is given. We can, however, use the same ideas to find cheap Nash equilibria in polynomial time (see Theorem 3).

*Properties and Terminology.* The problem input consists of graph $G = (V, E)$, and a set of $k$ terminals $T \subseteq V$ such that every player $i \in P$ is assigned one terminal. For every player $i$, $T_i = T \backslash \{i\}$. We define the following terms with respect to an instance of a multiway cut problem $(G, T)$. Any valid MWC $X$ must divide $G$ into at least $k$ components. Let the component containing terminal $i$ be termed as $C_i(X)$. We define $\delta_i(X)$ as the set of edges $(u, v)$ such that $|C_i(X) \cap \{u, v\}| = 1$ and $\delta_{ij}(X)$ as the set of edges $(u, v)$ such that $u \in C_i(X)$ and $v \in C_j(X)$. Terminal $j$ is a neighbor of terminal $i$ (denoted by $j \in N_i(X)$) if $\delta_{ij}(X) \neq \emptyset$. Let $G_i(X)$ be the subgraph defined as follows: $G_i(X) = G - X + \delta_i(X)$.

Let $OPT$ be the set of edges in the optimal Multiway Cut. For the sake of brevity, when defined for $OPT$ we will refer to the above terms simply as $C_i$, $\delta_i$, $\delta_{ij}$, $N_i$ and $G_i$ respectively. It is well known that when an optimal MWC is made, the graph $G$ is divided into exactly $k$ components each containing one terminal. Also, given a graph $G$, let $E[G]$ correspond to the set of edges of $G$.

**Lemma 1.** *Recall that $M(G_i, \{i\}, N_i)$ is the set of minimum edge cuts between terminal $i$ and its neighboring terminals in graph $G_i$. Then, there exists $M \in M(G_i, \{i\}, N_i)$, such that the set of edges in $M$ is a subset of $E[C_i] \cup \delta_i$.*

Following is a useful observation about the optimal solution of the Multiway Cut.

**Observation 1.** *If $OPT$ is the optimal multiway cut for a graph $G$ with terminals $T$, then for any $S \subseteq OPT$, $OPT - S$ is the optimal multiway cut for the graph $G - S$ with terminals $T$.*

We can now state the main theorem of the section.

**Theorem 2.** *The price of stability for the Network Multiway Cut problem is* 1*, i.e. there exists an assignment of the edges of OPT to the players that forms a Nash Equilibrium.*

In order to prove the theorem we make use of the following construction. For a multiway cut $X$ of an instance $(G, T)$, we construct a directed flow graph $F(G, T, X)$ as follows: Construct source node $s$ and sink node $t$. For every edge $e \in X$, construct vertices $m_e, n_e$ and a directed edge $(m_e, n_e)$ of capacity 1. Construct a directed edge $(n_e, t)$ of capacity $\infty$ for every such $n_e$. Add components $C_i(X)$ to this graph and make all of their edges bi-directed with capacity 1. Add an edge $(s, t_i)$ for every terminal $i$ with capacity $\infty$. For every vertex $v \in C_i(X)$ that has an edge $e$ of $\delta_i(X)$ incident on it, construct a directed edge $(v, m_e)$ with capacity $\infty$. The proofs of lemmas in this section appear in the full version.

**Lemma 2.** *Any bounded $s-t$ cut in $F(G, T, X)$ corresponds to a valid multiway cut for $(G, T)$ of the same size.*

If $X = OPT$, then from the construction we can see that the edges representing $OPT$ in $F(G, T, OPT)$ form a bounded $s-t$ cut of the same size. If the minimum $s-t$ cut on $F(G, T, OPT)$ is smaller than size of $OPT$ then Lemma 2 would imply that there exists a MWC of size smaller than $OPT$, which is a contradiction. So the minimum bounded $s-t$ on $F(G, T, OPT)$ has to be of the same size as $OPT$. This means that the maximum $s-t$ flow in $F(G, T, OPT)$ will saturate edges of $OPT$. Then the assignment of edges $S_i$ to terminal/player $i$ can be made as shown in Algorithm 1.

**Data**: Flow Graph $F(G, T, OPT)$
**Result**: Assignment of edges to players
Mark the flows that originate through terminal $i$ as $f_i$;
All edges of $OPT$ that carry flows marked $f_i$ will be assigned to $S_i$;

**Algorithm 1.** Algorithm that assigns edges of $OPT$ to players

Since $OPT$ forms the minimum bounded $s-t$ cut, we know that all edges of $OPT$ will be assigned. Now all we need to prove is that this assignment will form a Nash Equilibrium, which we do in the following lemma.

**Lemma 3.** *The assignment made by Algorithm 1 forms a Nash Equilibrium.*

### 3.1 Poly-time Computable Nash Equilibrium

Our algorithm in the previous section depends on the knowledge of the optimal Multiway cut for a given problem. Since this is often computationally infeasible due to the NP-hardness of the Multiway Cut problem, we give an algorithm that efficiently computes a Nash equilibrium whose cost is no larger than any given Multiway Cut. For example, we can begin with an $\alpha-$approximate MWC solution that can be obtained in polynomial time [7,23], and obtain a Nash equilibrium that is no more expensive than this solution. Currently, the best known approximation algorithm for the Multiway Cut is given by [23] and returns a 1.34-approximate solution.

**Theorem 3.** *Given a multiway cut $X$, we can find a Nash equilibrium whose social cost is no more than the cost of $X$ in polynomial time.*

## 4 Network Multicut Game

In the *Network Multicut Game* each player $i$ wants to be cut from a particular node $t_i$ of $G$ provided that the cost of the strategy $S_i$ of player $i$ does not exceed $\beta_i$. Recall that Network Multicut Game is a special case of the Network Cutting Game, where $T_i = \{t_i\}$ is singleton for every player. As pointed out before, the socially optimal solution is the minimum cost multicut for the pairs $\{(1, t_1), (2, t_2), \ldots, (k, t_k)\}$ if $\beta_i$ values are large enough. In general, OPT is a solution that minimizes the total cost of all the players.

For the Network Multicut game, we don't know whether there always exists a Nash equilibrium or not. In this section, we prove the existence of a 2-approximate Nash equilibrium that is as cheap as OPT. We do this by giving an algorithm that takes the edges of OPT as the input, and returns an assignment of edges of OPT to players that is a 2-approximate Nash equilibrium. First, we will give some properties of Nash equilibria that are as cheap as OPT for the Network Multicut Game, which enables us to give a simple algorithm that constructs a 2-approximate Nash equilibrium on the edges of OPT. Specifically, we first form stable payments locally for every pair of neighboring connected components in OPT, and then combine these payments to form a global 2-approximate Nash equilibrium.

Let $C_j$ and $C_k$ be two arbitrary components of OPT and let $\delta_{jk}$ denote the set of edges of $G$ that are between $C_j$ and $C_k$. Since $C_j$ and $C_k$ are cut apart in OPT, all the edges of $\delta_{jk}$ are cut in OPT. We call that $C_j$ and $C_k$ are neighbor components if $\delta_{jk} \neq \emptyset$. With the help of this notation, we can write the following simple observations.

**Lemma 4.** *For any two neighbor components $C_j$ and $C_k$ of OPT, there exists a player $i$ such that either $i \in C_j$ and $t_i \in C_k$ or vice versa.*

**Lemma 5.** *Let $i$ be a player such that $i \in C_j$ and $t_i \in C_k$ for 2 arbitrary neighbor components $C_j$ and $C_k$. Then player $i$ cannot cut any edge that is not in $\delta_{jk}$ in any Nash equilibrium that cuts OPT.*

**Lemma 6.** *If a player $i$ is not assigned any edge by the algorithm, i.e., $S_i = \emptyset$, then $i$ cannot reduce her cost by unilateral deviation (even in the case that $\beta_i$ is finite).*

To prove the existence of a 2-approximate Nash equilibrium as cheap as OPT, we give an algorithm that assigns all the edges of OPT to the players and prove that no player can reduce her cost by more than half by unilaterally deviating from the strategy where she cuts the edges assigned to her by the algorithm. If a player $i$ is not cut from $t_i$ in OPT, then our algorithm does not assign any edge to player $i$, i.e., $S_i = \emptyset$. If $i$ and $t_i$ are in different connected components of OPT, say $C_j$ and $C_k$ respectively, then our algorithm assigns a subset of $\delta_{jk}$ to player $i$. Notice that if $C_j$ and $C_k$ are not neighbor components then $S_i = \emptyset$.

A player $i$ may have an incentive for unilateral deviation from her strategy $S_i$ only if $i$ and $t_i$ are in different neighboring connected components of OPT because of Lemma 6. Let $G - S_{-i}$ denote the subgraph of $G$ where the edges cut by other players are removed and let $i$ be a player such that $0 < |S_i| \leq \beta_i$. Observe that a best deviation of player $i$ from her strategy $S_i$, which is denoted by $\chi_i(S_i)$, is a cheapest strategy that cuts $i$ from $t_i$ in $G - S_{-i}$. The cost of the best deviation of player $i$ from strategy $S_i$, i.e., $|\chi_i(S_i)|$, is as much as $m(G - S_{-i}, i, t_i)$. Let $G_{jk}$ be the subgraph of $G$, which is composed of the connected components $C_j$ and $C_k$ of OPT and edges $\delta_{jk}$. Since player $i$ does not cut any edge of OPT that is not an element of $\delta_{jk}$, then $OPT - \delta_{jk} \subset S_{-i}$ and therefore, $m(G - S_{-i}, i, t_i) = m(G_{jk}, i, t_i) = |\chi_i(S_i)|$.

*Algorithm.* Let $L_{jk}$ be the set of players $i$ such that either $i \in C_j$ and $t_i \in C_k$ or vice versa. Without loss of generality let $L_{jk} = \{1, 2, ..., |L_{jk}|\}$. Recall that $L_{jk} \neq \emptyset$ by Lemma 4. Notice that the socially optimal solution of the Network Multicut Game for players $L_{jk}$ on $G_{jk}$ (which we denote by $OPT(G_{jk}, L_{jk})$) is $\delta_{jk}$. This is by the same argument as in Observation 1. For every $\delta_{jk}$, we make one pass over the players $L_{jk}$. For player 1 of $L_{jk}$, we send a max-flow from node 1 to $t_1$ on $G_{jk}$(If the size of the max-flow is more than $\beta_1$ then we just send an arbitrary flow of size $\beta_1$). Let $m_1$ denote the subset of edges of $\delta_{jk}$ that are used by that flow. Notice that $|m_1| = \min\{m(G_{jk}, 1, t_1), \beta_1\}$. The algorithm asks player 1 to cut the edges of $m_1$, i.e., sets $S_1 = m_1$ and proceeds with player 2. We send a max-flow from 2 to $t_2$ on $G_{jk} - m_1$(Similarly, if the size of the max-flow is more than $\beta_2$ then we just send an arbitrary flow of size $\beta_2$) and ask player 2 to cut the edges $m_2$, the subset of $\delta_{jk} - m_1$ that are used by that flow and so on.

Let $M$ denote the subset of the edges of $\delta_{jk}$ that are cut by the players at the end of the one pass described in detail above, i.e., $M = \bigcup_{i \in L_{jk}} m_i$. Notice that if $M = \delta_{jk}$ for all neighbor components $C_j$ and $C_k$ then the above algorithm will return a Nash equilibrium. This is because $|m_i| = m(G - OPT + m_i, i, t_i)$ for all $i$ since the flow we used to construct $m_i$ does not use any edges of OPT except $m_i$, and thus $|m_i| \leq \min\{m(G - S_{-i}, i, t_i), \beta_i\}$ which by Proposition 1 implies that $i$ is stable.

However, it may be that $M \neq \delta_{jk}$ and so the greedy algorithm given above does not always return a Nash equilibrium. Fortunately, we know that $|M| \geq |\delta_{jk}|/2$ by Lemma 7. We next assign the remaining edges of $\delta_{jk}$ to the players $L_{jk}$ proportionally to the number of edges they are assigned so far. Since $|M| \geq |\delta_{jk}|/2$, then any player $i \in L_{jk}$ which has been assigned $|m_i|$ edges is now assigned at most $|m_i|$ extra edges. The assignment given by the algorithm is a 2-approximate Nash equilibrium, since the cost of the best deviation of each player $i \in L_{jk}$ is $\min\{m(G - OPT + S_i, i, t_i), \beta_i\}$, which is at least $|m_i|$ by the above argument.

**Lemma 7.** $|M| \geq |\delta_{jk}|/2$.

**Proof.** The critical observation is that $|OPT(G_{jk} - M, L_{jk})| - |OPT(G_{jk} - M, L_{jk} - \{1\})| \leq m_1$. For the purpose of contradiction, assume the inequality does not hold. If the max-flow between 1 and $t_1$ on $G_{jk}$ is less than $\beta_1$, then $m(G_{jk}, 1, t_1) = |m_1|$. Observe $m(G_{jk} - M, 1, t_1) \leq |m_1|$ since player 1 cannot send a bigger flow in a smaller graph. Then one can obtain a cheaper solution for the Network Multicut Game for the set of players $L_{jk}$ on the graph $G_{jk} - M$ than $OPT(G_{jk} - M, L_{jk})$ by first cutting the edges of $OPT(G_{jk} - M, L_{jk} - \{1\})$ and then cutting a min-cut between 1 and $t_1$ on the remaining edges. If the max-flow between 1 and $t_1$ on $G_{jk}$ is at least $\beta_1$, then $|m_1| = \beta_1$. Then one can obtain a cheaper solution for the Network Multicut Game for the set of players $L_{jk}$ on the graph $G_{jk} - M$ than $OPT(G_{jk} - M, L_{jk})$ by only cutting the edges of $OPT(G_{jk} - M, L_{jk} - \{1\})$. Note that in this solution player 1 does not cut any edges and faces a cost of $\beta_1$.

With the same argument one can show $|OPT(G_{jk} - M, L_{jk} - \{1\})| - |OPT(G_{jk} - M, L_{jk} - \{1, 2\})| \leq m_2$ and so on. Finally, we have $|OPT(G_{jk} - M, L_{jk} - \{1, 2, ..., (|L_{jk}| - 1)\})| - |OPT(G_{jk} - M, \emptyset)| \leq m_{|L_{jk}|}$. Summing up all the telescoping inequalities, we obtain $|OPT(G_{jk} - M, L_{jk})| \leq |M|$. It is also clear that $|OPT(G_{jk}, L_{jk})| - |OPT(G_{jk} - M, L_{jk})| \leq |M|$.

Therefore, $|\delta_{jk}| = |OPT(G_{jk}, L_{jk})| \leq 2|M|$. As desired, this proves that at least half of the edges of $\delta_{jk}$ are cut after one pass over the players of $L_{jk}$. ■

## 5 Edges with Non-uniform Costs

In the previous sections, the cost to player $i$ for cutting edges $S_i$ was just the number of edges $|S_i|$. We now consider a generalized version of our games where the edges have positive edge weights/costs $w(e)$. In order for an edge $e$ to be cut, players will have to pay the weight $w(e)$ of the edge. That is, the cost to player $i$ is no longer just the number of edges $|S_i|$, but their total weight which we represent by $w(S_i)$. Similarly, the cost of a strategy vector $S$ is now $\text{cost}(S) = w(S) + \sum_{j \in Q(S)} \beta_j$, where $Q(S)$ is defined as in the Introduction.

For this more general model with non-uniform edge costs, we first show that if the weight of an edge cannot be split between players, then the price of stability can be very high. By allowing players to share the cost of edges, however, we are able to extend most of our results to this general case.

*No Cost Sharing.* The game defined above does not allow players to share the cost of an edge, since if an edge $e \in S_i$, then the cost of player $i$ increases by the full weight $w(e)$ of the edge. We can show that in this case the price of stability can be as high as $k$, the number of players.

*Fair Sharing.* In this model, the players that cut an edge $e$ split the cost of this edge equally among themselves. Specifically, for a given strategy vector $S$, define $k_e$ to be the number of players $i$ such that $e \in S_i$. Then, player $i$ only pays $w(e)/k_e$ for cutting edge $e$, i.e., player $i$'s cost is $\sum_{e \in S_i} w(e)/k_e$ when $i$'s cut requirements are satisfied. All the games that we considered in sections 2, 3, and 4 are congestion games [5] under the fair sharing scheme. Using standard techniques [5], it can be shown that the price of stability is $O(\log k)$. Unfortunately, it can also be as high as $\Omega(\log k)$: just consider the example in Figure **??** with weights $1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{k}$ on the bottom edges instead of 1.

*Arbitrary Cost Sharing.* In this model players can choose to pay for arbitrary fractions of edge weights. Specifically, the strategy of each player $i$ is a payment function $S_i$ where $S_i(e)$ is the amount player $i$ pays for the cost of edge $e$. An edge $e$ is considered cut if the total payment for $e$ exceeds its weight, i.e., $\sum_i S_i(e) \geq w(e)$. The cost of each player $i$ is the total amount of payment it makes for cutting the edges, i.e., $\sum_e S_i(e)$. Observe that the arbitrary sharing model gives the players much larger freedom in selecting their strategies since the strategy space of each player is the positive orthant of the $m$-dimensional Euclidean space where $m = |E|$.

The algorithms presented in previous sections work for edges with unit cost. In order to extend those results to non-uniform weighted edges in the arbitrary cost sharing scheme, we make the following changes: Scale up the weights of edges so that all weights are integers. Simultaneously increase the value of $\beta_i$ by the same factor. Now replace any edge having weight $w(e)$ with $w(e)$ parallel edges of unit cost. We can now use the algorithms in sections 2, 3, and 4 to assign these edges with unit cost to players. This assignment can be easily mapped to the original graph with weighted edges where the players pay for fractions of edge weights. This gives us a Nash equilibrium solution where the edge weights are arbitrarily shared.

*Poly-time computable NE for Network Multiway Cut Game under Arbitrary cost sharing.* Since the algorithm for poly-time computable NE for the NMWG considered in Section 3.1 may take $O(E)$ steps, for non-uniform edge weights this may result in exponential running time. So the same algorithm does not work for this case. However we show that it is possible to form an approximate NE for NMWG in polynomial time.

**Theorem 4.** *Suppose we have a weighted NMCG and an MWC $S^\alpha$ that is within a factor $\alpha$ of OPT. Then for any $\epsilon > 0$, there is a poly-time algorithm which returns a $(1+\epsilon)$-approximate NE for a MWC $S'$, where $w(S') \leq w(S^\alpha)$.*

## References


1. Albers, S., Eilts, S., Even-Dar, E., Mansour, Y., Roditty, L.: On Nash Equilibria for a Network Creation Game. In: SODA (2006)
2. Aspnes, J., Chang, K., Yampolskiy, A.: Innoculation Strategies for Victims of Viruses and the Sum-of-Squares Problem. Journal of Computer and System Sciences 72(6), 1077–1093 (2006)
3. Anshelevich, E., Caskurlu, B.: Exact and Approximate Equilibria for Optimal Group Network Formation. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 239–250. Springer, Heidelberg (2009)
4. Anshelevich, E., Caskurlu, B.: Price of Stability in Survivable Network Design. In: Mavronicolas, M., Papadopoulou, V.G. (eds.) SAGT 2009. LNCS, vol. 5814, pp. 208–219. Springer, Heidelberg (2009)
5. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, É., Wexler, T., Roughgarden, T.: The Price of Stability for Network Design with Fair Cost Allocation. SIAM Journal on Computing 38(4), 1602–1623 (2008)
6. Anshelevich, E., Dasgupta, A., Tardos, É., Wexler, T.: Near-Optimal Network Design with Selfish Agents. In: Theory of Computing, vol. 4, pp. 77–109 (2008)
7. Calinescu, G., Karloff, H., Rabani, Y.: An improved approximation algorithm for multiway cut. Journal of Computer and System Sciences 60(3), 564–574 (2000)
8. Chen, H., Roughgarden, T.: Network Design with Weighted Players. In: SPAA 2006 (2006)
9. Chen, H., Roughgarden, T., Valiant, G.: Designing Networks with Good Equilibria. In: SODA 2008 (2008)
10. Christodoulou, G., Mirrokni, V.S., Sidiropoulos, A.: Convergence and Approximation in Potential Games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 349–360. Springer, Heidelberg (2006)

11. Engelberg, R., Könemann, J., Leonardi, S., Naor, J.: Cut Problems in Graphs with a Budget Constraint. Journal of Discrete Algorithms 5(2) (June 2007)
12. Epstein, A., Feldman, M., Mansour, Y.: Strong Equilibrium in Cost Sharing Connection Games. In: EC 2007 (2007)
13. Fabrikant, A., Luthra, A., Maneva, E., Papadimitriou, S., Shenker, S.: On a Network Creation Game. In: PODC (2003)
14. Fabrikant, A., Papadimitriou, C.H., Talwar, K.: The complexity of pure Nash equilibria. In: STOC 2004 (2004)
15. 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)
16. Gourves, L., Monnot, J.: On Strong Equilibria in the Max Cut Game. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 608–615. Springer, Heidelberg (2009)
17. Hamilton, S.N., Miller, W.L., Ott, A., Saydjari, O.S.: Challenges to applying game theory to the domain of information warfare. In: 4th Information Survivability Workshop (ISW-2001/2002), Vancouver, Canada (2002)
18. Hoefer, M.: Cost Sharing and Clustering under Distributed Competition, Ph.D Thesis, Universitat Konstanz (2007)
19. Hoefer, M.: Non-cooperative Facility Location and Covering Games. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 369–378. Springer, Heidelberg (2006)
20. Hoefer, M.: Non-cooperative Tree Creation. Algorithmica 53(1), 104–131 (2009)
21. Hoefer, M., Krysta, P.: Geometric Network Design with Selfish Agents. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 167–178. Springer, Heidelberg (2005)
22. Hoefer, M.: Strategic Cooperation in Cost Sharing Games. In: Saberi, A. (ed.) WINE 2010. LNCS, vol. 6484, pp. 258–269. Springer, Heidelberg (2010)
23. Karger, D.R., Klein, P., Stein, C., Thorup, M., Young, N.E.: Rounding algorithms for a geometric embedding for minimum multiway cut. Mathematics of Operations Research 29(3), 436–461 (2004)
24. Kearns, M., Oritz, L.: Algorithms for Interdependent Security Games. In: Advances in Neural Information Processing Systems, vol. 16. MIT Press, Cambridge (2004)
25. Kunreuther, H., Heal, G.: Interdependent Security. Journal of Risk and Uncertainity (Special Issue on Terrorist Risks) (2003)
26. Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., Van Briesen, J., Glance, N.: Cost-effective Outbreak Detection in Networks. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD (2007)
27. Jackson, M.: A survey of models of network formation: stability and efficiency. In: Demange, G., Wooders, M. (eds.) Group Formation in Economics: Networks, Clubs and Coalitions. Cambridge Univ. Press, Cambridge
28. Mavronicolas, M., Michael, L., Papadopoulou, V.G., Philippou, A., Spirakis, P.G.: The Price of Defense. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 717–728. Springer, Heidelberg (2006)
29. Monderer, D., Shapley, L.: Potential Games. Games and Economic Behavior 14, 124–143 (1996)
30. Syverson, P.F.: A different look at secure distributed computation. In: IEEE Computer Security Foundations Workshop (CSFW 10) (June 1997)

# Approximating Directed Buy-at-Bulk Network Design

Spyridon Antonakopoulos

Bell Laboratories, Alcatel-Lucent,
600-700 Mountain Avenue, Murray Hill, NJ 07974, USA
spyros@research.bell-labs.com

**Abstract.** Buy-at-bulk network design is a well-known problem that has been researched extensively on undirected graphs. In this paper, we initiate the theoretical study of the same problem on directed graphs, thus capturing real-life situations where the cost of installing capacity on an edge is asymmetric with respect to direction, as e.g. in the design of wireless and satellite communication networks.

More specifically, we develop two approximation algorithms for directed buy-at-bulk network design in the non-uniform cost model. Combined, they achieve a ratio of $O\big(\min\big\{k^{1/2+\epsilon}, n^{4/5+\epsilon}\big\}\big)$ for any constant $\epsilon > 0$, where $n$ is the number of nodes of the network and $k$ is the number of demand pairs to be connected. Further, the above ratio is independent of the amount of traffic flow requested by each demand pair, which may vary arbitrarily, and it can be remarkably improved when all demand pairs share a common sink (or a common source, symmetrically).

To the best of our knowledge, this is the first non-trivial approximation factor established for the aforementioned problem, and naturally it also applies to more restricted cost models, such as the uniform and rent-or-buy models. Moreover, it essentially matches the best currently known approximation guarantees for directed Steiner forest, which may be viewed as a special case of directed buy-at-bulk network design.

## 1 Introduction

Network design has long been a fundamental area of combinatorial optimization. In a typical such problem, we are given a network topology and a list of pairs of network nodes, each with an associated traffic demand. The goal is to determine how to route in the network the traffic demand between each of those pairs, so that some objective function is optimized—such as minimizing the cost of installing the requisite capacity to support all traffic. For more than a decade, researchers have devoted a lot of attention to the buy-at-bulk aspect of network design, in which cost is a sub-additive function of capacity, motivated by the fact that actual costs of network components tend to exhibit economies of scale. However, the theoretical study of the above problem has been limited to undirected graphs so far; by contrast, in this work we focus on directed graphs.

*Asymmetric networks.* To model a real-life problem more accurately, directed graphs are often indispensable. Perhaps the most prominent such example is the

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 13–24, 2011.

asymmetric traveling salesman problem: among other things, it is straightforward to represent one-way streets by directed edges, whereas undirected edges would obviously be inappropriate for that purpose. Similar concerns arise in network design when the cost of carrying a unit of traffic across an edge varies substantially depending on direction, as is often the case in communication networks [21,5], e.g. if they contain satellite and/or wireless links. It is worth pointing out that as the popularity of these so-called *asymmetric networks* continues to increase, they have become the object of intense study by the networking community. For instance, see [20,22,15], among many others.

The typical cause of the aforementioned phenomenon is the disparity between the transmitter and receiver capabilities of network equipment, which in turn may be due to various technical, geographical, and even regulatory constraints. As an illustrative example, consider a cellular phone communicating with a base station. It is much easier, by comparison, to achieve some desired transfer rate from the latter to the former than vice versa, primarily because the transmission power of the phone is subject to heavy limitations. Hence, just as the buy-at-bulk framework was developed to reflect the economies of scale prevalent in network costs, a directed graph representation is needed to capture common situations where supporting a given bandwidth capacity in one direction along a network edge is markedly more expensive than in the opposite direction.

*Cost models.* The most general cost model considered in buy-at-bulk network design is called the *non-uniform* cost model. It stipulates that the cost of supporting $\ell \geq 0$ units of traffic on a given edge $e$ is given by a sub-additive, non-negative, and non-decreasing function $c_e(\ell)$, with $c_e(0) = 0$. On the other hand, the *uniform* cost model adds the restriction that there must exist some function $c(\ell)$ such that for every $e$ in the network we have $c_e(\ell) = C_e c(\ell)$, with $C_e \geq 0$ depending only on $e$. Moreover, the special case where $c(\ell) = 1$ for all $\ell > 0$ is equivalent to the well-known Steiner forest problem.

Any buy-at-bulk cost function $c_e(\ell)$ may be approximated by the lower envelope of polynomially many linear cost functions [2,9]. Namely, we can replace every edge $e$ with $p(e)$ parallel edges $e_1, e_2, \ldots, e_{p(e)}$, such that: (a) for each $i = 1, \ldots, p(e)$, $c_{e_i}(\ell) = \sigma(e_i) + \delta(e_i) \cdot \ell$ for $\ell > 0$, where $\sigma(e_i), \delta(e_i)$ are non-negative constants called *cost coefficients*; and (b) $c_e(\ell) \leq \min_{i=1,\ldots,p(e)} c_{e_i}(\ell) < 3 \cdot c_e(\ell)$. Note also that specifying each new edge's cost coefficients suffices to describe its cost function. Thus, this alternative representation is (approximately) equivalent to the original one, and we shall use it throughout the rest of the paper.

*Related work.* As mentioned above, there is a sizable body of research work devoted to buy-at-bulk network design, but only in the undirected setting. One of the earliest approximation algorithm formulations of the problem was due to Salman et al. [23]. Subsequently, Awerbuch and Azar [3] obtained a $O(\log n)$ approximation for uniform buy-at-bulk network design, where $n$ is the number of nodes of the network, using probabilistic embeddings into tree metrics [4,12]. For the *single-sink* case (i.e. when all demand pairs have a common sink, as opposed to the more general *multi-commodity* case that has no such restriction), Guha et al. [16] were the first to achieve an $O(1)$ approximation.

Non-uniform buy-at-bulk network design proved to be considerably more challenging to tackle. For the single-sink case, Meyerson et al. [19] presented a randomized $O(\log n)$ approximation algorithm, later derandomized in [10]; there is also an $O\big(\log^4 n\big)$ competitive online algorithm [18]. Charikar and Karagiozova [7] gave an $e^{O\left(\sqrt{\log n \log\log n}\right)}$ approximation for the multi-commodity case, and a polylogarithmic approximation was obtained fairly recently by Chekuri et al. [9], achieving an $O\big(\log^4 k\big)$ ratio, where $k$ is the number of demand pairs.

On the other hand, unlike buy-at-bulk network design, Steiner problems have been studied in the directed setting for quite some time. Zelikovsky [24] first gave an $O(k^\epsilon)$ approximation for directed Steiner tree on acyclic graphs. Charikar et al. [5] obtained a slightly better ratio on arbitrary directed graphs using a somewhat simpler algorithm. Furthermore, they achieved an $\tilde{O}\big(k^{2/3}\big)$ approximation for directed Steiner forest, which was improved to $O\big(k^{1/2+\epsilon}\big)$ by Chekuri et al. [8] almost a decade later. Very recently, Feldman et al. [13] provided another approximation guarantee for this problem, namely $O\big(n^{4/5+\epsilon}\big)$. Since $k$ may range from 1 to $O\big(n^2\big)$, the latter two results are mutually incomparable.

Additionally, it should be pointed out that directed Steiner problems are closely connected to group Steiner problems. In particular, group Steiner tree (on undirected graphs) is essentially a special case of directed Steiner tree. The relation between group Steiner forest and directed Steiner forest is analogous. Polylogarithmic approximations have been established for both group Steiner tree [14] (using LP-rounding) and group Steiner forest [8].

Finally, regarding lower bounds, Andrews [1] demonstrated that there is no $\Omega\big(\log^{1/2-\epsilon} n\big)$ approximation algorithm for non-uniform buy-at-bulk network design in the undirected setting, unless $\mathsf{NP} \subseteq \mathsf{ZPTIME}\big(n^{\mathrm{polylog}(n)}\big)$. This result clearly distinguishes buy-at-bulk network design from Steiner forest, which is 2-approximable. For directed Steiner forest, however, Dodis and Khanna [11] showed a hardness factor of $\Omega\big(2^{\log^{1-\epsilon} n}\big)$ assuming $\mathsf{NP} \nsubseteq \mathsf{TIME}\big(n^{\mathrm{polylog}(n)}\big)$, which obviously carries over to directed buy-at-bulk network design as well.

*Our results.* We present two approximation algorithms for directed buy-at-bulk network design in the non-uniform cost model, which together yield an approximation ratio of $O\big(\min\big\{k^{1/2+\epsilon}, n^{4/5+\epsilon}\big\}\big)$ for any constant $\epsilon > 0$, where $n$ is the number of nodes of the network and $k$ is the number of demand pairs to be connected. Note that the ratio is independent of the amount of traffic flow demanded by each such pair, which may vary arbitrarily, and also that it can be reduced to $O(k^\epsilon)$ (or $O(n^\epsilon)$, equivalently) in the single-sink case.

To the best of our knowledge, this is the first substantial improvement on the trivial approximation guarantee of $k$ for this problem. Of course, the above result also covers the uniform cost model as a special case, and furthermore it matches up to a constant factor the best currently known approximation ratios for directed Steiner forest from [8] and [13].

*Overview of techniques.* Our algorithms extend those in [8] and [13] for directed Steiner forest. More specifically, to obtain the $O\big(k^{1/2+\epsilon}\big)$ approximation, we first reduce our problem to what may be described as the "buy-at-bulk

generalization" of group Steiner forest, using the so-called *layering* and *path-splitting* transformations [8]. In turn, that is reduced to the buy-at-bulk generalization of group Steiner tree, following the *bucketing-and-scaling* method pioneered in [9]. We then apply a refined version of the LP-rounding scheme developed in [14] for group Steiner tree, and eventually produce a partial solution that connects at least some of the demand pairs, at a reasonable cost. The algorithm is then applied recursively to any demand pairs that remain unconnected.

For the analysis of the above algorithm, it is crucial to establish first a structural property of low-cost solutions, which we achieve by generalizing Zelikovsky's tree height-reduction procedure [24,17] in a suitable fashion. In particular, the original height-reduction procedure outputs a tree contained in the metric closure of the network graph. In the buy-at-bulk setting, then, we need to define the *bi-metric closure* of a graph, which extends the standard concept of metric closure and may also be of independent interest. We also take advantage of the fact that Zelikovsky's procedure preserves the lengths of all paths from the root of the tree to its leaves, under *any* metric. This property is usually overlooked, because it is not necessary for typical applications of the procedure to Steiner problems. However, it does play a pivotal role within the bi-metric closure framework that we introduce, since more than one metrics are involved.

On the other hand, to derive the $O\big(n^{4/5+\epsilon}\big)$ approximation ratio, we partition the demand pairs into two classes, namely the *good* and the *bad*. The good pairs are distinguished by the fact that they can all be connected in a relatively straightforward way, at a reasonable cost. Nevertheless, the situation becomes more complicated for the bad pairs. For these, we have to try out a number of options that include invoking the $O\big(k^{1/2+\epsilon}\big)$-approximation algorithm mentioned earlier, and rounding a near-optimal solution to a restricted LP relaxation of the problem. This approach resembles the one employed in [13]; however, the definition of good pairs is more specialized in our context, thus requiring a more elaborate case analysis to handle the bad pairs.

*Organization.* The next section presents several key concepts, including the formal definition of the directed buy-at-bulk network design problem. In Section 3, we show the structural result that is vital to the analysis of our algorithms, which are then described in Section 4. Lastly, Section 5 discusses some conclusions that may be drawn from our results and suggests directions for future research.

## 2 Preliminaries

An instance of the directed buy-at-bulk network design problem has the form $(G, D)$, where $G = (V, E)$ is a directed multigraph with two non-negative cost coefficients $\sigma(e), \delta(e)$ specified for every edge $e \in E$, and $D = \{d_{uv}\}$ is a $|V| \times |V|$ matrix with non-negative integer elements. An ordered pair $(u, v) \in V \times V$ for which $d_{uv} \geq 1$ is called a *demand pair*; furthermore, $u$ and $v$ are the *source* and *sink* of the demand pair, respectively. For simplicity of notation, we denote by $\mathcal{D} \subseteq V \times V$ the set of demand pairs, by $k$ the cardinality of $\mathcal{D}$, and by $n$ the cardinality of $V$. Additionally, for any path $P$, let $\sigma(P) = \sum_{e \in P} \sigma(e)$ and $\delta(P) = \sum_{e \in P} \delta(e)$.

A feasible solution to this instance consists of a collection $\mathcal{P} = \{P(u,v)\}$ of $k$ paths in $G$ such that for each demand pair $(u,v) \in \mathcal{D}$, $\mathcal{P}$ contains a path $P(u,v)$ from $u$ to $v$. We say that $P(u,v)$ *serves* the demand pair $(u,v)$ in the solution $\mathcal{P}$. The goal is to minimize the *cost* of $\mathcal{P}$:

$$\mathsf{c}(\mathcal{P}) = \sum_{e \in \mathcal{P}} \sigma(e) + \sum_{P(u,v) \in \mathcal{P}} \delta(P(u,v)) \cdot d_{uv} = \sum_{e \in \mathcal{P}} \big(\sigma(e) + \delta(e) \cdot \ell(e)\big)\,, \qquad (1)$$

where the notation $e \in \mathcal{P}$ indicates that edge $e$ belongs to at least one of the paths in $\mathcal{P}$, and $\ell(e)$ is the *load* of edge $e$ and is equal to $\sum d_{uv}$, with the sum taken over all $(u,v) \in \mathcal{D}$ such that $e \in P(u,v)$.

Without loss of generality, we henceforth assume that: (a) $d_{vv} = 0$ for all $v \in V$; (b) if a node of $G$ is the source of a demand pair, then it has in-degree zero; and (c) if a node is the sink of a demand pair, then it has out-degree zero. The latter two properties can be guaranteed easily by a simple transformation, so that the number of nodes of $G$ is at most tripled and the cost of the optimal solution remains unaffected.

A *partial solution* is a collection $\mathcal{P} = \{P(u,v)\}$ of paths that serve a non-empty subset $\mathcal{D}(\mathcal{P})$ of the demand pairs, where again $P(u,v)$ denotes the path serving the pair $(u,v) \in \mathcal{D}(\mathcal{P})$. The cost $\mathsf{c}(\mathcal{P})$ of the partial solution $\mathcal{P}$ is given by (1), similarly to the case of a feasible solution. Moreover, the quantity $\mathsf{den}(\mathcal{P}) = \mathsf{c}(\mathcal{P})/|\mathcal{D}(\mathcal{P})|$ is called the *density* of the partial solution $\mathcal{P}$. In particular, if $\mathcal{P}$ is a feasible solution then its density is simply $\mathsf{den}(\mathcal{P}) = \mathsf{c}(\mathcal{P})/k$.

Finally, we introduce a concept that generalizes the metric closure of a graph.

**Definition 1.** *Given a directed multigraph $G = (V,E)$ with two non-negative cost coefficients $\sigma(e), \delta(e)$ specified for every $e \in E$, we define the* bi-metric closure *$\bar{G} = (V, \bar{E})$ of $G$ as a directed multigraph such that: (a) for each $(u,v) \in V \times V$, $u \neq v$, and for every distinct (simple) path $P$ from $u$ to $v$ in $G$, there exists a directed edge $\bar{e} \in \bar{E}$ from $u$ to $v$ with cost coefficients $\sigma(\bar{e}) = \sigma(P)$ and $\delta(\bar{e}) = \delta(P)$; and (b) $\bar{E}$ contains no edges other than those required by (a).*

Observe that $\bar{G}$ may be exponential in size. Fortunately, for our purposes we can avoid computing it in its entirety, as we shall see below.

**Definition 2.** *Given an instance $(G,D)$ of directed buy-at-bulk network design, let $\ell_{\max} = \sum_{(u,v) \in \mathcal{D}} d_{uv}$, which is an upper bound of the load on any edge in any feasible solution. We define the* restricted bi-metric closure *$\hat{G} = (V, \hat{E})$ of $G$ as a subgraph of $\bar{G}$ that is minimal under the following property: for each integer $0 \leq z < \log \ell_{\max}$ and for every $(u,v) \in V \times V$, $u \neq v$, there exists an edge $\hat{e} \in \hat{E} \subseteq \bar{E}$ from $u$ to $v$ such that for any other $\bar{e} \in \bar{E}$ from $u$ to $v$, $\sigma(\hat{e}) + \delta(\hat{e}) \cdot 2^z \leq \sigma(\bar{e}) + \delta(\bar{e}) \cdot 2^z$.*

Clearly, $\hat{G}$ is computable by executing a standard all-pairs shortest path routine on $G$ with edge weights $\sigma(e) + \delta(e) \cdot z$, for each possible value of $z$. This takes time $\mathrm{poly}(n, \log \ell_{\max})$ overall, which is polynomial with respect to the size of the instance $(G,D)$.

## 3 Structural Results

The goal in this section is to show that there always exist partial solutions of a very specific form that have sufficiently low density.

**Definition 3.** *A* junction tree *$J$ is the union of an in-branching $J_I$ and an out-branching $J_O$, both rooted at some node $r$ which is called the* junction node*. $J_I$ and $J_O$ are not required to be edge-disjoint. The* height *of $J$ is the maximum of the heights of $J_I$ and $J_O$, i.e. the maximum length (in terms of the number of edges) of a simple path in $J$ having one endpoint at $r$.*

Furthermore, the *support* of a partial solution $\mathcal{P} = \{P(u,v)\}$ is the union of the paths it contains, i.e. $\bigcup P(u,v)$. The first lemma below, whose proof we omit due to space constraints, is analogous to Lemma 3.1 in [8]. Combining it with the second lemma leads to Corollary 1.

**Lemma 1.** *There exists a partial solution $\mathcal{P}^\circ = \{P^\circ(u,v)\}$ whose support is a junction tree $J^\circ \subseteq G$ and $\mathsf{den}(\mathcal{P}^\circ) \leq \sqrt{k} \cdot \mathsf{den}(\mathcal{P}^*)$, where $\mathcal{P}^*$ is the optimal solution to the instance $(G,D)$.*

**Lemma 2.** *Let $\mathcal{P} = \{P(u,v)\}$ be a partial solution to the instance $(G,D)$ with density $\mathsf{den}(\mathcal{P})$, whose support is a junction tree $J \subseteq G$. For any integer $h \geq 1$, there exists a partial solution $\hat{\mathcal{P}} = \{\hat{P}(u,v)\}$ to the instance $(\hat{G},D)$ whose support is a junction tree $\hat{J} \subseteq \hat{G}$ with height $\leq h$ and $\mathsf{den}(\hat{\mathcal{P}}) < 8h\sqrt[h]{k/2} \cdot \mathsf{den}(\mathcal{P})$.*

*Proof.* Recall that $J$ is the union of an in-branching $J_I$ and an out-branching $J_O$, both rooted at the junction node $r$. Denote by $\mathcal{D}(\mathcal{P}) \subseteq \mathcal{D}$ the subset of demand pairs connected by the paths in $\mathcal{P}$. Moreover, for each $(u,v) \in \mathcal{D}(\mathcal{P})$, denote by $P(u,r)$ and $P(r,v)$ the segments of $P(u,v)$ contained in $J_I$ and $J_O$ respectively. Observe that node $u$ must be a leaf of $J_I$ and $v$ must be a leaf of $J_O$, by our assumptions in Section 2.

Now, we shall invoke the height-reduction procedure proposed in [24,17]. We briefly explain its functionality below. Formally, the input must be an integer parameter $h \geq 1$, a branching $T$, and a non-negative weight function $w$ defined on the edges of $T$. However, since the output actually does not depend on those edge weights, they need not be specified beforehand. The procedure returns another branching $\bar{T}$ that has height at most $h$ and whose node set $V(\bar{T})$ is a subset of $V(T)$. In particular, $\bar{T}$ has the same root and leaves as $T$, but may be missing some of the latter's internal nodes. On the other hand, each directed edge $\bar{e} = (u,v)$ of $\bar{T}$ corresponds to the path $P_T(u,v)$ from $u$ to $v$ in $T$, and is assigned a weight $w(\bar{e}) = \sum_{e \in P_T(u,v)} w(e)$. The salient property of $\bar{T}$ is that

$$\sum_{\bar{e} \in \bar{T}} w(\bar{e}) \leq 2h\sqrt[h]{s/2} \cdot \sum_{e \in T} w(e)\ , \tag{2}$$

where $s$ is the number of leaves of $T$.

Let $\bar{J}_I$ and $\bar{J}_O$ be the results of applying the aforementioned procedure to $J_I$ and $J_O$ respectively. The junction tree $\bar{J} = \bar{J}_I \cup \bar{J}_O$ is a subgraph of the bi-metric

closure $\bar{G}$ of $G$. For any pair $(u,v) \in \mathcal{D}(\mathcal{P})$, there exists a unique path $\bar{P}(u,v)$ in $\bar{J}$ that consists of edges corresponding to paths in $J$ which, if concatenated, produce exactly the path $P(u,v)$. Therefore, $\delta(\bar{P}(u,v)) = \delta(P(u,v))$. If $\bar{\mathcal{P}} = \{\bar{P}(u,v) \mid (u,v) \in \mathcal{D}(\mathcal{P})\}$, then using inequality (2) with $\sigma(e)$ substituting for $w(e)$ we derive:

$$\begin{aligned}
\mathsf{c}(\bar{\mathcal{P}}) &= \sum_{\bar{e}\in\bar{\mathcal{P}}} \sigma(\bar{e}) + \sum_{\bar{P}(u,v)\in\bar{\mathcal{P}}} \delta(\bar{P}(u,v)) \cdot d_{uv} \leq \\
&\leq \sum_{\bar{e}\in\bar{J}_I} \sigma(\bar{e}) + \sum_{\bar{e}\in\bar{J}_O} \sigma(\bar{e}) + \sum_{P(u,v)\in\mathcal{P}} \delta(P(u,v)) \cdot d_{uv} \leq \\
&\leq 2h \sqrt[h]{\frac{|\mathcal{D}(\mathcal{P})|}{2}} \cdot \left( \sum_{e\in J_I} \sigma(e) + \sum_{e\in J_O} \sigma(e) \right) + \sum_{P(u,v)\in\mathcal{P}} \delta(P(u,v)) \cdot d_{uv} \leq \\
&\leq 2h \sqrt[h]{k/2} \cdot \left( 2 \sum_{e\in\mathcal{P}} \sigma(e) \right) + \sum_{P(u,v)\in\mathcal{P}} \delta(P(u,v)) \cdot d_{uv} \leq 4h \sqrt[h]{k/2} \cdot \mathsf{c}(\mathcal{P}) \,.
\end{aligned}$$

For an edge $\bar{e} \in \bar{J}$, define $\ell(\bar{e})$ as the sum of $d_{uv}$ for pairs $(u,v) \in \mathcal{D}(\mathcal{P})$ such that $\bar{e} \in \bar{P}(u,v)$, and set $\bar{z} = \lfloor \log \ell(\bar{e}) \rfloor$. Among the edges parallel to $\bar{e}$ in $\bar{G}$, let $\hat{e}$ be the one that minimizes $\sigma(\hat{e}) + \delta(\hat{e}) \cdot 2^{\bar{z}}$. Hence, $\hat{e}$ also belongs to the restricted bi-metric closure $\hat{G}$ of $G$, and furthermore it is easy to see that $\sigma(\hat{e}) + \delta(\hat{e}) \cdot \ell(\bar{e}) < 2\left(\sigma(\bar{e}) + \delta(\bar{e}) \cdot \ell(\bar{e})\right)$. We create $\hat{J} \subseteq \hat{G}$ from $\bar{J}$ (and the corresponding path collection $\hat{\mathcal{P}}$ from $\bar{\mathcal{P}}$) by replacing each edge $\bar{e} \in \bar{J}$ with $\hat{e}$ as above. Clearly, $\hat{\mathcal{P}}$ is a partial solution to the instance $(\hat{G}, D)$ with cost $\mathsf{c}(\hat{\mathcal{P}}) < 2 \cdot \mathsf{c}(\bar{\mathcal{P}}) \leq 8h \sqrt[h]{k/2} \cdot \mathsf{c}(\mathcal{P})$ and density $\mathsf{den}(\hat{\mathcal{P}}) < 8h \sqrt[h]{k/2} \cdot \mathsf{den}(\mathcal{P})$.

**Corollary 1.** *For any integer $h \geq 1$, there exists a partial solution $\hat{\mathcal{P}}^\circ = \{\hat{P}^\circ(u,v)\}$ to the instance $(\hat{G}, D)$ whose support is a junction tree $\hat{J}^\circ \subseteq \hat{G}$ with height $\leq h$ and $\mathsf{den}(\hat{\mathcal{P}}^\circ) < \sqrt{k} \cdot 8h \sqrt[h]{k/2} \cdot \mathsf{den}(\mathcal{P}^*) < 8h \cdot k^{1/2+1/h} \cdot \mathsf{den}(\mathcal{P}^*)$.*

## 4 Approximation Algorithms

*$O\big(k^{1/2+\epsilon}\big)$ approximation.* As implied by the discussion in the previous section, the principal task of our algorithm is to determine a low-density partial solution $\hat{\mathcal{P}} = \{\hat{P}(u,v)\}$ to the instance $(\hat{G}, D)$ whose support is a junction tree $\hat{J}$ in $\hat{G}$ with height $\leq h$. (Note that $h$ is an adjustable parameter that is specified in advance of the algorithm's execution.) Then, $\hat{\mathcal{P}}$ is straightforwardly converted into a partial solution $\mathcal{P} = \{P(u,v)\}$ to the original instance $(G, D)$ with at most the same density: for every path $\hat{P}(u,v)$, replace each of its edges with the corresponding path in $G$ and concatenate these paths to form $P(u,v)$. Once we have $\mathcal{P}$, we set $d_{uv} = 0$ for the demand pairs $(u,v) \in \mathcal{D}(\mathcal{P})$ and repeat the same process recursively on the residual instance, until there are no demand pairs left. The final solution is simply the union of all partial solutions produced.

In the following, we describe how to compute $\hat{\mathcal{P}}$. First of all, the algorithm *guesses* the junction node $r$ of the junction tree $\hat{J}^\circ$ in $\hat{G}$ that is implied by

Corollary 1. In practice, this means exhaustively trying out each of $n$ possible choices for $r$ and keeping the one that yields the best result. Assuming we know the correct $r$, we proceed as follows.

**Step 1: Layering and path splitting.** We construct an undirected tree $\mathcal{T} = (V(\mathcal{T}), E(\mathcal{T}))$ from $\hat{G}$ by applying the layering and path-splitting procedures presented in Section 3.3 of [8]. We refer the reader to that paper for details of the construction, which are omitted here to save space. Instead, we provide a high-level description of the resulting $\mathcal{T}$ and its main properties.

First of all, every node of $\mathcal{T}$ is a copy of some node of $\hat{G}$. For any $v \in V$, the number of its copies in $V(\mathcal{T})$ is polynomially bounded with respect to the size of $\hat{G}$ (which in turn is polynomial in the size of the problem instance) and exponentially bounded with respect to the parameter $h$ (which does not depend on the instance). Furthermore, each edge $\check{e}$ of $\mathcal{T}$ either (a) joins two distinct copies of the same node $v \in V$, or (b) is an undirected copy of some edge $\hat{e}$ of $\hat{G}$. In the former case, $\check{e}$ is assigned cost coefficients $\sigma(\check{e}) = 0$ and $\delta(\check{e}) = 0$, whereas in the latter we have $\sigma(\check{e}) = \sigma(\hat{e})$, $\delta(\check{e}) = \delta(\hat{e})$, and the endpoints of $\check{e}$ are copies of the endpoints of $\hat{e}$.

Overall, the tree $\mathcal{T}$ is rooted at node $r_0$, which is a copy of $r$, and all its leaves are at distance exactly $h$ from $r_0$. For each $v \in V$, denote by $X(v) \subseteq V(\mathcal{T})$ the set of leaves that are copies of $v$. The construction of $\mathcal{T}$ ensures that for every $(u, v) \in \mathcal{D}$, all paths that connect any node in $X(u)$ with any node in $X(v)$ must pass through the root $r_0$. Additionally, consider a collection $\check{\mathcal{P}}$ of paths in $\mathcal{T}$. Among these, let $\check{P}(u, v)$ be the path connecting any node in $X(u)$ with any node in $X(v)$ that minimizes $\delta(\check{P}(u, v))$, if any such path is contained in $\check{\mathcal{P}}$. Setting $\mathcal{D}(\check{\mathcal{P}}) = \{(u, v) \in \mathcal{D} \mid \check{P}(u, v) \text{ exists in } \check{\mathcal{P}}\}$, we define the cost of $\check{\mathcal{P}}$ as

$$\mathsf{c}(\check{\mathcal{P}}) = \sum_{\check{e} \in \check{\mathcal{P}}} \sigma(\check{e}) + \sum_{(u,v) \in \mathcal{D}(\check{\mathcal{P}})} \delta(\check{P}(u, v)) \cdot d_{uv} \ ,$$

and its density as $\mathsf{den}(\check{\mathcal{P}}) = \mathsf{c}(\check{\mathcal{P}})/|\mathcal{D}(\check{\mathcal{P}})|$, assuming $\mathcal{D}(\check{\mathcal{P}}) \neq \varnothing$. By the construction of $\mathcal{T}$, from any such path collection $\check{\mathcal{P}}$ we may easily derive a partial solution $\hat{\mathcal{P}}$ to the instance $(\hat{G}, D)$ whose support is a junction tree of height $\leq h$ and $\mathsf{den}(\hat{\mathcal{P}}) \leq \mathsf{den}(\check{\mathcal{P}})$. Conversely, for any given partial solution $\hat{\mathcal{P}}$ as above, there exists a collection of paths in $\mathcal{T}$ with density equal to $\mathsf{den}(\hat{\mathcal{P}})$.

**Step 2: Bucketing and scaling.** We now address the problem of finding a collection $\check{\mathcal{P}}$ of paths in $\mathcal{T}$ with minimum density. A linear programming relaxation of this problem is presented in **LP**1. Each $x_e$ variable indicates whether edge $e \in \check{\mathcal{P}}$; $y_{uv}$ indicates whether there is a path in $\check{\mathcal{P}}$ from any node in $X(u)$ to any node in $X(v)$; and $f_{e,uv}$ indicates whether $e$ is used by such a path. We normalize the sum of all $y_{uv}$ variables to 1, thus the objective function is linear.

**LP**1 is efficiently solvable, either by first reformulating it into an equivalent linear program with a polynomial number of flow-conservation constraints on the $f_{e,uv}$ variables, or by applying the Ellipsoid method on **LP**1 directly, since it admits a polynomial-time separation oracle. Denote by $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{f}^*)$ the resulting optimal fractional solution, whose value OPT is a lower bound on the minimum

$$
\begin{aligned}
\textbf{LP}1\colon \quad \min \quad & \sum_{e\in E(\mathcal{T})} \left( \sigma(e)x_e + \delta(e) \sum_{(u,v)\in\mathcal{D}} d_{uv}\, f_{e,uv} \right) \\
\text{s.t.} \quad & \sum_{(u,v)\in\mathcal{D}} y_{uv} = 1 \\
& x_e \geq f_{e,uv} && \forall e \in E(\mathcal{T}),\, \forall (u,v) \in \mathcal{D} \\
& \sum_{e\in\partial(U)} f_{e,uv} \geq y_{uv} && \forall U \subseteq V(\mathcal{T}),\, \forall (u,v) \in \mathcal{D}\colon (r_0 \in U) \wedge \\
& && \wedge \big((U \cap X(u) = \varnothing) \vee (U \cap X(v) = \varnothing)\big) \\
& x_e, y_{uv}, f_{e,uv} \geq 0 && \forall e \in E(\mathcal{T}),\, \forall (u,v) \in \mathcal{D}
\end{aligned}
$$

density of any collection of paths in $\mathcal{T}$. From the preceding discussion, we deduce that $\textsc{opt} \leq \mathsf{den}(\hat{\mathcal{P}}^\circ)$, where $\hat{\mathcal{P}}^\circ$ is the partial solution to the instance $(\hat{G}, D)$ implied by Corollary 1.

Let $q = \lceil \log k \rceil + 1$ and $\mathcal{D}_i = \big\{(u,v) \in \mathcal{D} \;\big|\; 2^{-i} < y^*_{uv} \leq 2^{-i+1}\big\}$, for $1 \leq i \leq q$. Note that $\sum_{i=1}^{q} \big( \sum_{(u,v)\in\mathcal{D}_i} y^*_{uv} \big) \geq \frac{1}{2}$, therefore at least one of the inner sums must be $\geq 1/(2q)$. If $i^*$ is the index value corresponding to that sum, then $|\mathcal{D}_{i^*}| \geq 2^{i^*-1}/(2q) = 2^{i^*-2}/q$. Hence, we formulate a new linear program **LP**2, by removing the constraint $\sum_{(u,v)\in\mathcal{D}} y_{uv} = 1$ from **LP**1 and fixing the $y_{uv}$ variables as follows: $y_{uv} = 1$ for $(u,v) \in \mathcal{D}_{i^*}$, and $y_{uv} = 0$ for $(u,v) \in \mathcal{D} \setminus \mathcal{D}_{i^*}$.

**Step 3: Rounding.** Next, we efficiently compute an optimal fractional solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{f}})$ to **LP**2. Its value $\widetilde{\textsc{opt}}$ is at most $2^{i^*}\textsc{opt}$, because $(\tilde{\mathbf{x}}', \tilde{\mathbf{f}}') = \big(2^{i^*}\mathbf{x}^*, 2^{i^*}\mathbf{f}^*\big)$ is a feasible solution to **LP**2.

Consider another feasible solution $(\mathring{\mathbf{x}}, \mathring{\mathbf{f}}) = (2\tilde{\mathbf{x}}, 2\tilde{\mathbf{f}})$. For each demand pair $(u,v) \in \mathcal{D}_{i^*}$, let $\Delta_{uv} = \sum_{e\in E(\mathcal{T})} \delta(e)\tilde{f}_{e,uv} = \frac{1}{2}\sum_{e\in E(\mathcal{T})} \delta(e)\mathring{f}_{e,uv}$. The $\mathring{f}_{e,uv}$ values represent a fractional 2-unit flow, with sources in $X(u)$ and sinks in $X(v)$. This flow can be decomposed into a collection $\Pi_{uv}$ of non-disjoint paths, such that each path carries a fraction of the total flow. Observe that all the paths in $\Pi_{uv}$ pass through $r_0$. If $\Pi'_{uv} = \{P \in \Pi_{uv} \mid \delta(P) \leq 2\Delta_{uv}\}$, then by Markov's inequality at least one unit of flow is carried by the paths in $\Pi'_{uv}$. We reduce the flow along the paths in $\Pi_{uv} \setminus \Pi'_{uv}$ to zero and, if necessary, we also reduce the flow along paths in $\Pi'_{uv}$ so that there is exactly one unit of flow remaining in total. Then, we adjust the $\mathring{f}_{e,uv}$ values accordingly and prune the $\mathring{x}_e$ values so that $\mathring{x}_e = \max_{(u,v)\in\mathcal{D}_{i^*}}\{\mathring{f}_{e,uv}\}$ for all $e$. This transformation does not affect the feasibility of $(\mathring{\mathbf{x}}, \mathring{\mathbf{f}})$.

Finally, we round $(\mathring{\mathbf{x}}, \mathring{\mathbf{f}})$ to an integral solution $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$ as follows. Consider the edges of $\mathcal{T}$ in breadth-first-search order, starting from the root $r_0$. For each edge $e \in E(\mathcal{T})$, either $e$ is incident to $r_0$ or it has a *parent* edge $e'$, which is the edge adjacent to $e$ but closer to $r_0$. In the first case, set $\bar{x}_e = 1$ with probability $\mathring{x}_e$, or $\bar{x}_e = 0$ otherwise. In the second case, if $\bar{x}_{e'} = 0$ then set $\bar{x}_e = 0$; else set $\bar{x}_e = 1$ with probability $\mathring{x}_e/\mathring{x}_{e'}$, or $\bar{x}_e = 0$ otherwise. Note that $\mathring{x}_e \leq \mathring{x}_{e'}$, because $\mathcal{T}$ is a tree and all flow paths pass through $r_0$. Moreover, denote by $\bar{\mathcal{T}} \subseteq \mathcal{T}$

the tree formed by the edges for which $\bar{x}_e = 1$, and let $\bar{\mathcal{D}}_{i^*} = \{(u,v) \in \mathcal{D}_{i^*} \mid \bar{\mathcal{T}}$ contains a path $\breve{P}(u,v) \in \Pi'_{uv}\}$. For each $(u,v) \in \bar{\mathcal{D}}_{i^*}$, set $\bar{f}_{e,uv} = 1$ along $\breve{P}(u,v)$ and $\bar{f}_{e,uv} = 0$ for all other edges. For each $(u,v) \in \mathcal{D}_{i^*} \setminus \bar{\mathcal{D}}_{i^*}$, set $\bar{f}_{e,uv} = 0$ for all edges. Now, we can efficiently convert $\breve{\mathcal{P}} = \{\breve{P}(u,v) \mid (u,v) \in \bar{\mathcal{D}}_{i^*}\}$ into a partial solution $\hat{\mathcal{P}}$ to the instance $(\hat{G}, D)$, with $\mathsf{den}(\hat{\mathcal{P}}) \leq \mathsf{den}(\breve{\mathcal{P}})$.

**Analysis.** It is straightforward to realize that the expected cost of $\breve{\mathcal{P}}$ is at most the expected value of $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$, which does not exceed $\sum_{e \in E(\mathcal{T})} \big(\sigma(e)\mathring{x}_e + \delta(e)\sum_{(u,v)\in\mathcal{D}_{i^*}} d_{uv}\,\mathring{f}_{e,uv}\big) \leq 2 \cdot \widetilde{\text{OPT}}$. For each $(u,v) \in \mathcal{D}_{i^*}$, with probability $\Omega\big(h^{-1}\big)$ there exists a path in $\bar{\mathcal{T}}$ from $r_0$ to any node of $X(u)$ that is an endpoint of a path in $\Pi'_{uv}$. This is deduced from the arguments in the proof of Theorem 3.4 in [14], adapted to the special case of a tree with height $\leq h$. The same is true for $X(v)$, by symmetry, and the two events are independent because any node in $C(u_0)$ and any node in $C(v_{2h})$ have no common ancestor in $\mathcal{T}$ except $r_0$. Hence, the probability that at least one path in $\Pi'_{uv}$ is contained in $\breve{\mathcal{P}}$ is $\Omega\big(h^{-2}\big)$, which is a constant. As a result, with high probability $|\bar{\mathcal{D}}_{i^*}| = \Omega(|\mathcal{D}_{i^*}|)$. Alternatively, we may perform the rounding using the deterministic techniques from [6], so that an analogous property holds with probability 1. Therefore $\mathsf{den}(\breve{\mathcal{P}}) \leq O\big(\widetilde{\text{OPT}}/|\mathcal{D}_{i^*}|\big) \leq O\big(2^{i^*}\,\text{OPT}/\big(2^{i^*-2}/q\big)\big) \leq O(\log k)\cdot\text{OPT}$, and thus $\mathsf{den}(\hat{\mathcal{P}}) \leq O(\log k)\cdot\text{OPT} \leq O(\log k)\cdot\mathsf{den}(\hat{\mathcal{P}}^\circ)$. Consequently, we have proven the following:

**Lemma 3.** *The algorithm computes a partial solution $\hat{\mathcal{P}}$ to the instance $(\hat{G}, D)$ with density $\mathsf{den}(\hat{\mathcal{P}}) \leq O\big(k^{1/2+1/h}\log k\big)\cdot\mathsf{den}(\mathcal{P}^*)$, where $\mathcal{P}^*$ is the optimal solution to the instance $(G, D)$.*

As mentioned earlier, $\hat{\mathcal{P}}$ is easily transformed into a partial solution $\mathcal{P}$ to the instance $(G, D)$ with $\mathsf{den}(\mathcal{P}) \leq \mathsf{den}(\hat{\mathcal{P}})$. By applying a well-known argument for recursive greedy covering (see e.g. [5,13]), we establish that the final solution produced by our algorithm has density at most $O\big(k^{1/2+1/h}\log k\big)\cdot\mathsf{den}(\mathcal{P}^*)$. Additionally, the overall running time is polynomial in the size of the instance. Setting the parameter $h$ appropriately, e.g. $h = \lceil 2/\epsilon\rceil$, guarantees that:

**Theorem 1.** *For any constant $\epsilon > 0$, the directed buy-at-bulk network design problem may be efficiently approximated within an $O\big(k^{1/2+\epsilon}\big)$ factor.*

*$O\big(n^{4/5+\epsilon}\big)$ approximation.* We start by guessing the value of $\mathsf{c}(\mathcal{P}^*)$, within a factor of 2 (so that we need to make only a polynomial number of guesses). Furthermore, for each demand pair $(u,v)$, define a node set $B(u,v)$ as follows. The node $w \in V$ belongs to $B(u,v)$ if and only if there exist in $G$ two paths $P_1$ and $P_2$, from $u$ to $w$ and from $w$ to $v$ respectively, such that (a) $\sigma(P_1), \sigma(P_2) \leq \mathsf{c}(\mathcal{P}^*)/n^{4/5}$, and (b) $\delta(P_1), \delta(P_2) \leq n^{4/5}\cdot\mathsf{c}(\mathcal{P}^*)/k$. We say that a demand pair $(u,v)$ is *good* if $|B(u,v)| \geq n^{2/5}$, and *bad* otherwise. Denote by $\mathcal{D}_g$ and $\mathcal{D}_b$ the sets of good and bad pairs respectively, with $k_g = |\mathcal{D}_g|$ and $k_b = |\mathcal{D}_b|$. We have:

**Lemma 4.** *For any $\tau \in \{g, b\}$, we can efficiently compute a partial solution $\mathcal{P}_\tau$ such that $\mathcal{D}(\mathcal{P}_\tau) = \mathcal{D}_\tau$ and $\mathsf{c}(\mathcal{P}_\tau) \leq O\big(n^{4/5+\epsilon}\big)\cdot\mathsf{c}(\mathcal{P}^*)$.*

This lemma, whose proof is omitted here due to space constraints, leads immediately to our second main result.

**Theorem 2.** *For any constant $\epsilon > 0$, the directed buy-at-bulk network design problem may be efficiently approximated within an $O\big(n^{4/5+\epsilon}\big)$ factor.*

*Approximations for the single-sink case.* If all demand pairs share a common sink (or source, symmetrically), then a simple argument enables us to assume that the support of the optimal solution is a junction tree. This signifies an improvement of the quantitative claim in Lemma 1 by a factor of $\sqrt{k}$, and said improvement propagates to Theorem 1, yielding the corollary below.

**Corollary 2.** *For any constant $\epsilon > 0$, the single-sink directed buy-at-bulk network design problem may be efficiently approximated within an $O(k^{\epsilon})$ factor.*

The above corollary also implies an $O(n^{\epsilon})$ approximation for the single-sink case, because in the latter the number $k$ of demand pairs cannot exceed $n - 1$.

## 5 Concluding Remarks and Open Problems

Recall that, in undirected graphs, buy-at-bulk network design is much harder to approximate than Steiner forest—and provably so, modulo a reasonable complexity assumption. By contrast, as a consequence of this work, the current picture in the directed case appears to be quite the opposite, i.e. the two problems have essentially identical approximation guarantees. A potential explanation may be that directionality is now the dominant factor determining problem hardness, overshadowing the effects of the cost model. On the other hand, our results could also be construed as a small piece of evidence that there exist, in fact, better approximation algorithms for directed Steiner forest, thus meriting further investigation. Last but not least, by the same token it may be possible to obtain a tighter inapproximability bound for directed buy-at-bulk network design than for directed Steiner forest, e.g. of the form $\Omega\big(n^{\lambda}\big)$ for some positive $\lambda$.

**Acknowledgments.** The author would like to thank Guy Kortsarz and Mihalis Yannakakis for numerous stimulating discussions, as well as Chandra Chekuri for some valuable comments. This work was performed at Columbia University, with support from NSF grant CCF-0728736 and an Alexander S. Onassis Foundation Scholarship.

## References

1. Andrews, M.: Hardness of buy-at-bulk network design. In: Proceedings of IEEE FOCS, pp. 115–124 (2004)
2. Andrews, M., Zhang, L.: Approximation algorithms for access network design. Algorithmica 34(2), 197–215 (2002)
3. Awerbuch, B., Azar, Y.: Buy-at-bulk network design. In: Proceedings of IEEE FOCS, pp. 542–547 (1997)

4. Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: Proceedings of IEEE FOCS, pp. 184–193 (1996)
5. Charikar, M., Chekuri, C., Cheung, T., Dai, Z., Goel, A., Guha, S., Li, M.: Approximation algorithms for directed Steiner problems. Journal of Algorithms 33, 73–91 (1999)
6. Charikar, M., Chekuri, C., Goel, A., Guha, S.: Rounding via trees: Deterministic approximation algorithms for group Steiner trees and k-median. In: Proceedings of ACM STOC, pp. 114–123 (1998)
7. Charikar, M., Karagiozova, A.: On non-uniform multicommodity buy-at-bulk network design. In: Proceedings of ACM STOC, pp. 176–182 (2005)
8. Chekuri, C., Even, G., Gupta, A., Segev, D.: Set connectivity problems in undirected graphs and the directed Steiner network problem. In: Proceedings of ACM-SIAM SODA, pp. 532–541 (2008)
9. Chekuri, C., Hajiaghayi, M.T., Kortsarz, G., Salavatipour, M.R.: Approximation algorithms for nonuniform buy-at-bulk network design. SIAM Journal on Computing 39(5), 1772–1798 (2010), http://link.aip.org/link/?SMJ/39/1772/1
10. Chekuri, C., Khanna, S., Naor, J.: A deterministic algorithm for the cost-distance problem. In: Proceedings of ACM-SIAM SODA, pp. 232–233 (2001)
11. Dodis, Y., Khanna, S.: Designing networks with bounded pairwise distance. In: Proceedings of ACM STOC, pp. 750–759 (1999)
12. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. Journal of Computer and System Sciences 69, 485–497 (2004)
13. Feldman, M., Kortsarz, G., Nutov, Z.: Improved approximation for the directed Steiner forest problem. In: Proceedings of ACM-SIAM SODA, pp. 922–931 (2009)
14. Garg, N., Konjevod, G., Ravi, R.: A polylogarithmic approximation algorithm for the group Steiner tree problem. Journal of Algorithms 37, 66–84 (2000)
15. Ge, F., Tan, L., Zukerman, M.: Throughput of FAST TCP in asymmetric networks. IEEE Communications Letters 12(2), 158–160 (2008)
16. Guha, S., Meyerson, A., Munagala, K.: A constant factor approximation for the single sink edge installation problem. SIAM Journal on Computing 38(6), 2426–2442 (2009)
17. Helvig, C.S., Robins, G., Zelikovsky, A.: Improved approximation scheme for the group Steiner problem. Networks 37(1), 8–20 (2001)
18. Meyerson, A.: Online algorithms for network design. In: Proceedings of ACM SPAA, pp. 275–280 (2004)
19. Meyerson, A., Munagala, K., Plotkin, S.: Cost-distance: Two metric network design. SIAM Journal on Computing 38(4), 1648–1659 (2008)
20. Obata, H., Ishida, K., Funasaka, J., Amano, K.: TCP performance analysis on asymmetric networks composed of satellite and terrestrial links. In: Proceedings of IEEE ICNP, pp. 199–206 (2000)
21. Ramanathan, S.: Multicast tree generation in networks with asymmetric links. IEEE/ACM Transactions on Networking 4(4), 558–568 (1996)
22. Ramasubramanian, V., Mossé, D.: BRA: A bidirectional routing abstraction for asymmetric mobile ad hoc networks. IEEE/ACM Transactions on Networking 16(1), 116–129 (2008)
23. Salman, F.S., Cheriyan, J., Ravi, R., Subramanian, S.: Buy at bulk network design: Approximating the single-sink edge installation problem. SIAM Journal on Optimization 11(3), 595–610 (2000)
24. Zelikovsky, A.: A series of approximation algorithms for the acyclic directed Steiner tree problem. Algorithmica 18, 99–110 (1997)

# New Lower Bounds for Certain Classes of Bin Packing Algorithms*

János Balogh, József Békési, and Gábor Galambos

Department of Applied Informatics, Gyula Juhász Faculty of Education,
University of Szeged
{balogh,bekesi,galambos}@jgypk.u-szeged.hu

**Abstract.** On-line algorithms have been extensively studied for the one-dimensional bin packing problem. In this paper we investigate two classes of the one- dimensional bin packing algorithms, and we give lower bounds for their asymptotic worst-case behaviour. For on-line algorithms so far the best lower bound was given by van Vliet in 1992 [13]. He proved that there is no on-line bin packing algorithm with better asymptotic performance ratio than 1.54014 .... In this paper we give an improvement on this bound to $\frac{248}{161} = 1.54037\ldots$ and we investigate the parametric case as well. For those lists where the elements are preprocessed according to their sizes in decreasing order Csirik et al. [1] proved that no on-line algorithm can have an asymptotic performance ratio smaller than $\frac{8}{7}$. We improve this result to $\frac{54}{47}$.

## 1 Introduction

The one-dimensional bin packing problem can be stated as follows. We are given a list $L$ of $n$ items – where the number of items is the length of the list – with sizes $a_i$, $i = 1, \ldots, n$, satisfying $0 < a_i \leq 1$. We need to pack these items into a minimal number of unit-capacity bins such that the total sum of the sizes in each bin is at most 1. The problem is known to be NP-hard [7]. So, substantial research has been focused on finding good approximation algorithms. One possibility to measure the performance of an algorithm $A$ is to give its asymptotic performance ratio $R_A$. For a list $L$ let $\mathrm{OPT}(L)$ denote the number of bins in an optimal packing and let $A(L)$ denote the number of bins that algorithm $A$ uses for packing $L$. If $R_A(l)$ denotes the supremum of the ratios $A(L)/\mathrm{OPT}(L)$ for all lists $L$ with $\mathrm{OPT}(L) = l$, then the asymptotic performance ratio is defined as

$$R_A := \limsup_{l \to \infty} R_A(l).$$

If an algorithm belongs to the class of *on-line algorithms* then it packs items immediately when they appear without any knowledge of subsequent items of the list. After an item has been placed in a bin, it must not be moved again. This lack of knowledge is such a severe handicap that no on-line algorithm can have an

* This research was supported by HSC-DAAD Hungarian-German Research Exchange Program (project P-MÖB/837).

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 25–36, 2011.

asymptotic performance ratio close to 1. In case of on-line algorithms it is more fashionable to use the phrase asymptotic competitive ratio instead of asymptotic performance ratio. The best known on-line algorithm is due to Seiden [10] with asymptotic performance ratio at most 1.58889.., while van Vliet [13] gave a lower bound 1.54014... for any on-line algorithm in 1992. He also investigated the parametric case, where for the sizes of the elements the inequality $0 < a_i \leq \frac{1}{r}$ is true for some $r > 1$ integer. To prove his result van Vliet considered the solution of a special linear program. The proof is rather complicated and assumes a fair amount of knowledge about linear programming.

It was observed very early that the asymptotic performance ratio of on-line algorithms becomes significantly better if one can suppose that the elements arrive in decreasing order. For this case the best known on-line algorithm is *First Fit Decreasing* (FFD) given by Garey et al. [8] with $R_{\mathrm{FFD}} = \frac{11}{9}$. For pre-ordered lists the best known lower bound is $\frac{8}{7}$. It was given by Csirik et al. [1]. So we have a very narrow gap [1.142857..., 1.22...] between the lower- and upper-bounds. In spite of great efforts, neither lower nor upper bound could be improved in the past 27 years.

This paper is organized as follows. In Section 2 we reformulate the packing pattern technique first introduced in [3]. In Section 3 we show that using this technique the $1.54014\ldots$ lower bound is also achievable with the right choice of the weights. Giving new sequences for the sizes of elements, in Section 4 we consider the parametric case and we slightly improve the van Vliet's lower-bound to $\frac{248}{161} = 1.54037\ldots$. In Section 5 for pre-ordered lists we improve the $\frac{8}{7}$ lower-bound to $\frac{54}{47} = 1.148936\ldots$. Some open problems conclude the paper.

## 2 Reformulated Packing Pattern Technique

In this section we reformulate the packing pattern technique which was first evaluated in [3]. Later the method was used by Galambos and Frenk in [4]. Both versions allowed only equal length lists in the construction of the proof. In his PhD thesis van Vliet [14] extended the technique for those constructions where one can use sub-lists with different sizes. Since we will use this basic theorem in our improvements we discuss the proof in detail. Firstly, we need some preliminaries and we also introduce some notations.

For an arbitrary large integer $n$ we consider lists $L_1, L_2, \ldots, L_k$ of lengths $n_j = c_j \cdot n$ for certain integers $c_j$, $j = 1, 2, \ldots, k$. Sub-list $L_j$ contains equally sized elements. We assume that the size of an item does not depend on $n$. In the concatenated list $(L_1 L_2 \ldots L_j)$ the elements of $L_1$ are followed by the elements of $L_2$ etc. and the list is terminated by the elements of $L_j$.

As a further notation, let $n \cdot U_j$ be an upper bound for the optimal packing of the concatenated list $(L_1 L_2 \ldots L_j)$, i.e.

$$U_j \geq \frac{\mathrm{OPT}(L_1 L_2 \ldots L_j)}{n}, \quad 1 \leq j \leq k.$$

Using the definition of the asymptotic performance ratio it is clear that for any on-line algorithm $A$

$$R_A \geq \max_{1\leq j\leq k} \limsup_{n\to\infty} \frac{A(L_1L_2\dots L_j)}{\mathrm{OPT}(L_1L_2\dots L_j)} \geq \max_{1\leq j\leq k} \limsup_{n\to\infty} \frac{A(L_1L_2\dots L_j)}{n\cdot U_j}.$$

In order to establish the theorems we introduce the definition of packing patterns, which was first defined in [3]. Suppose that some algorithm $A$ packs the elements of the concatenated list $L = (L_1L_2\dots L_k)$ into bins. A packing pattern $p = (p_1, p_2, \dots, p_k)$ is a vector that denotes the number of elements from every list $L_j$, $j = 1, 2, \dots, k$, while the algorithm places items into a bin according to that packing pattern. A packing pattern is feasible if $\sum_{i=1}^{k} a_i p_i \leq 1$. The set of all feasible packing patterns will be denoted by $P$. We define the subsets

$$P_i = \{p \in P \mid \ p_i > 0 \text{ and } p_j = 0, \text{ for } j < i\},\ i = 1, 2, \dots k.$$

Clearly, $P_i \cap P_j = \emptyset$ if $i \neq j$, and $P = \cup_{i=1}^{k} P_i$.

While we pack the elements of the concatenated list $L = (L_1L_2\dots L_k)$, every bin must be filled according to one feasible packing pattern. For a given type $p = (p_1, p_2, \dots, p_k)$ we denote the total number of bins which have been packed according to packing pattern $p$ by $n(p)$. The number of bins used by algorithm $A$ while successively packing the lists is

$$A(L_1\dots L_j) = \sum_{i=1}^{j} \sum_{p\in P_i} n(p),\ \text{ for } j = 1, 2, \dots, k, \tag{1}$$

and

$$n_j = \sum_{p\in P} p_j n(p),\ \text{ for } j = 1, 2, \dots, k. \tag{2}$$

Van Vliet stated the following theorem.

**Theorem 1.** *[14] Let $w_j$, $1 \leq j \leq k$, be some positive weights such that for every $p \in P_i$*

$$\sum_{j=i}^{k} w_j p_j \leq k - i + 1 \tag{3}$$

*holds. Then for every on-line algorithm $A$ we have that*

$$R_A \geq \frac{\sum_{j=1}^{k} w_j c_j}{\sum_{j=1}^{k} U_j}, \tag{4}$$

In this theorem van Vliet considered $k$ positive weights without any further condition, so if we apply this theorem for a special class of algorithms the weights can be arbitrary small. To avoid this inconvenience we can rescale the weights, and so we reformulate the above theorem as follows.

**Theorem 2.** *Let $\alpha_j$ and $\beta_j$ are $2k$ positive integers such that for every $p \in P_i$*

$$\sum_{j=i}^{k} \beta_j p_j \leq \sum_{j=i}^{k} \alpha_j. \tag{5}$$

*Then for every on-line algorithm $A$ we have that*

$$R_A \geq \max_{1 \leq j \leq k} \limsup_{n \to \infty} \frac{A(L_1 L_2 \dots L_j)}{OPT(L_1 L_2 \dots L_j)} \geq \frac{\sum_{j=1}^{k} \beta_j c_j}{\sum_{j=1}^{k} \alpha_j U_j}. \tag{6}$$

*Proof.* If we multiply, for $j = 1, 2, \dots, k$, the equations (1) and (2) by $\alpha_j$ and $\beta_j$, respectively, and sum all weighted equations, we get

$$\sum_{j=1}^{k} \alpha_j A(L_1 \dots L_j) = \sum_{j=1}^{k} \alpha_j \sum_{i=1}^{j} \sum_{p \in P_i} n(p) \tag{7}$$

and

$$\sum_{j=1}^{k} \beta_j n_j = \sum_{j=1}^{k} \beta_j \sum_{p \in P} p_j n(p). \tag{8}$$

Because of the property of the constants it follows that

$$\begin{aligned}
\sum_{j=1}^{k} \alpha_j \sum_{i=1}^{j} \sum_{p \in P_i} n(p) = & \sum_{p \in P_1} (\alpha_1 + \alpha_2 + \dots + \alpha_k)\, n(p) + \\
& + \sum_{p \in P_2} (\alpha_2 + \dots + \alpha_k)\, n(p) \dots + \sum_{p \in P_k} \alpha_k n(p) \\
\geq & \sum_{p \in P_1} (\beta_1 p_1 + \beta_2 p_2 + \dots + \beta_k p_k)\, n(p) \\
& + \sum_{p \in P_2} (\beta_2 p_2 + \dots + \beta_k p_k)\, n(p) + \dots + \sum_{p \in P_k} \beta_k p_k n(p) \\
= & \sum_{j=1}^{k} \beta_j \sum_{p \in P} p_j n(p).
\end{aligned}$$

So – using (1) and (2) – we get that

$$\sum_{j=1}^{k} \alpha_j A(L_1 \dots L_j) \geq \sum_{j=1}^{k} \beta_j n_j. \tag{9}$$

Therefore

$$\begin{aligned}
R_A \geq \max_{1 \leq j \leq k} \limsup_{n \to \infty} \frac{\alpha_j A(L_1 L_2 \dots L_j)}{\alpha_j \mathrm{OPT}(L_1 L_2 \dots L_j)} \geq \limsup_{n \to \infty} \frac{\sum_{j=1}^{k} \alpha_j A(L_1 \dots L_j)}{\sum_{j=1}^{k} \alpha_j OPT(L_1 \dots L_j)} \geq \\
\geq \limsup_{n \to \infty} \frac{\sum_{j=1}^{k} \beta_j n_j}{\sum_{j=1}^{k} \alpha_j OPT(L_1 \dots L_j)} \geq \limsup_{n \to \infty} \frac{n \sum_{j=1}^{k} \beta_j c_j}{n \sum_{j=1}^{k} \alpha_j U_j} = \frac{\sum_{j=1}^{k} \beta_j c_j}{\sum_{j=1}^{k} \alpha_j U_j}.
\end{aligned}$$

## 3 The Right Choice of the Weights

In [4] Galambos and Frenk did not give an explicit discussion of the packing pattern technique, but – using the idea of the packing pattern – they were able to give a simpler proof for the $1.5363\ldots$ lower bound for on-line bin packing algorithms given by Liang [9]. They investigated the parametric case as well. In [14] Van Vliet - using his generalization - improved the lower bound to $1.54014\ldots$. Here we will show that the right choice of the weights allows us to give the same lower bound using the packing pattern technique as van Vliet got with the help of the linear programming technique. During his proof he constructed a linear program, he solved it and defined two functions $f_k$ and $g_k$, both of them depending on $k$. He received his result as a limit of a function in $f_k$ and $g_k$ for $k \to \infty$. Since van Vliet proved that with the help of the applied sequences there is no possibility to get a better lower bound, our procedure will also justify that our approach has the same power as the LP method has.

In all of the above papers a specific sequence – mostly called as *Salzer sequence* – was applied to construct lists with equal sizes of elements. This sequence was first introduced by Sylvester in 1880 [12], therefore we will refer to this sequence as *Sylvester sequence.* We define for $k > 1$ and $r \geq 1$ integers the Sylvester sequence $m_1, \ldots, m_k$ by setting

- $m_1 = r + 1$,
- $m_2 = r + 2$,
- $m_j = m_{j-1}(m_{j-1} - 1) + 1$, for $j = 3, \ldots, k$.

**Table 1.** The first few elements of the parametric Sylvester sequences if $k \geq 5$

| | $r = 1$ | $r = 2$ | $r = 3$ | $r = 4$ | $r = 5$ |
|---|---|---|---|---|---|
| $m_1 = r + 1$ | 2 | 3 | 4 | 5 | 6 |
| $m_2 = r + 2$ | 3 | 4 | 5 | 5 | 7 |
| $m_3 = m_1 m_2 + 1$ | 7 | 13 | 21 | 31 | 43 |
| $m_4 = m_3(m_3 - 1) + 1$ | 43 | 157 | 421 | 931 | 1807 |
| $m_5 = m_4(m_4 - 1) + 1$ | 1807 | 24493 | 176821 | 865831 | 3263443 |

Now we define $k$ lists as follows. Let $n = c(m_k - 1)$ for some positive integer $c$. Each list $L_j$, $j = 1, \ldots, k-1$, contains $n$ elements, while $L_k$ contains $rn$ pieces of elements, i.e. $c_j = 1$, if $j = 1, 2, \ldots, k-1$, and $c_k = r$. The sizes of elements in $L_j$ are $a_j = 1/m_{k-j+1} + \varepsilon$, where $0 < \varepsilon < 1/(r+k)(m_k(m_k - 1))$. The following Lemma was proved in [9].

**Lemma 1**

*(i)* $OPT(L_1 L_2 \ldots L_j) = \frac{n}{m_{k-j+1}}$, *for all* $j = 1, \ldots, k-1$.
*(ii)* $OPT(L_1 L_2 \ldots L_k) = n$.

So for a fixed $k$ we set

$$U_j = \begin{cases} \frac{1}{m_{k-j+1}-1} & \text{, if } 1 \le j \le k-1, \\ 1 & \text{, if } j = k, \end{cases}$$

and we define the following constants.

$$\beta_j = \begin{cases} 1 & \text{, if } \quad j = 1 \\ (m_{k-j+1}-1)\beta_{j-1} & \text{, if } \quad 2 \le j \le k-1, \\ \beta_{k-1} & \text{, if } \quad j = k. \end{cases}$$

$$\alpha_j = \begin{cases} \beta_{j+1} & \text{, if } \quad 1 \le j \le k-1, \\ r\beta_k & \text{, if } \quad j = k. \end{cases}$$

Comparing our weights to the ones given in [14] we can realize the difference between them. So, although the formula is almost the same, our result is better than the one that van Vliet has got with the help of the packing pattern technique. On the other side, it is also easy to check that our proof is much simpler than the LP technique.

**Theorem 3.** *[13] There is no one-dimensional on-line bin packing algorithm A with worst case ratio*

$$R_A \ge \lim_{k\to\infty} \frac{\sum_{j=1}^{k} c_j\beta_j}{\sum_{j=1}^{k} \alpha_j U_j}.$$

As an example we show the case $r = 1$, $k = 3$, where $m_1 = 2$, $m_2 = 3$, $m_3 = 7$, $\beta_1 = 1$, $\beta_2 = 2$, $\beta_3 = 2$, $\alpha_1 = 2$, $\alpha_2 = 2$, $\alpha_3 = 2$, $U_1 = \frac{1}{6}$, $U_2 = \frac{1}{2}$, $U_3 = 1$. So we get

$$R_A \ge \frac{\sum_{j=1}^{3} c_j\beta_j}{\sum_{j=1}^{3} \alpha_j U_j} = \frac{5}{\frac{1}{3}+1+2} = \frac{3}{2}.$$

Table 2 displays the van Vliet's lower bounds for the asymptotic performance ratio of on-line algorithms for some values of $k$ and $r$, which where calculated by our formula.

## 4 The New Parametric On-Line Lower Bound

Proving his result van Vliet used the Sylvester sequence. This is a so-called double exponential sequence whose reciprocals tend very quickly to zero. That is the reason why constructing a lower bound for $k = 5$ the first five decimals have been reached by the appropriate choice of the sizes in the lists. During the last two decades a lot of efforts have been made to improve this result. It was already proved by van Vliet that his result was not improvable with the Sylvester sequence. Therefore we inquired to find other sequences which do not tend so

**Table 2.** van Vliet's lower bounds for on-line bin packing algorithms

| | $r=1$ | $r=2$ | $r=3$ | $r=4$ | $r=5$ |
|---|---|---|---|---|---|
| $k=3$ | 1,5000000 | 1,3793103 | 1,2878787 | 1,2283464 | 1,1880733 |
| $k=4$ | 1,5390070 | 1,3895759 | 1,2914337 | 1,2298587 | 1,1888167 |
| $k=5$ | 1,5401467 | 1,3896489 | 1,2914427 | 1,2298604 | 1,1888172 |
| $k=6$ | 1,5401474 | 1,3896489 | 1,2914427 | 1,2298604 | 1,1888172 |
| $k=7$ | 1,5401474 | 1,3896489 | 1,2914427 | 1,2298604 | 1,1888172 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $k=\infty$ | 1,5401474 | 1,3896489 | 1,2914427 | 1,2298604 | 1,1888172 |

quickly. Besides other approaches we attempted to give up the greedy choice of the next elements in the sequence. Among other – unsuccessful – shots we hit the following sequence. For any $r \geq 1$ integer let

- $b_{1,r} = r + 1$,
- $b_{2,r} = r + 2$,
- $b_{j,r} = (b_{1,r}b_{2,r} + 1)^{j-2}, \qquad 3 \leq j \leq k-1$,
- $b_{k,r} = b_{1,r}b_{2,r}b_{3,r}^{k-3} + 1$.

For the sake of simpler notation instead of $b_{i,r}$ we will use the notation $b_i$.

**Table 3.** The first few parametric values of the new sequence for $k = 6$

| | $r=1$ | $r=2$ | $r=3$ | $r=4$ | $r=5$ |
|---|---|---|---|---|---|
| $b_1 = r+1$ | 2 | 3 | 4 | 5 | 6 |
| $b_2 = r+2$ | 3 | 4 | 5 | 6 | 7 |
| $b_3 = b_1b_2 + 1$ | 7 | 13 | 21 | 31 | 43 |
| $b_4 = (b_1b_2+1)^2$ | 49 | 169 | 441 | 961 | 1849 |
| $b_5 = (b_1b_2+1)^3$ | 343 | 2197 | 9261 | 29791 | 79507 |
| $b_6 = b_1b_2b_3^3 + 1$ | 2059 | 26365 | 185221 | 893731 | 3339295 |

It is easy to prove that for any fixed $k < \infty$ integer

$$r\frac{1}{b_1} + \sum_{i=2}^{k} \frac{1}{b_i} = 1.$$

If we compare the contents of Table 1 and Table 3 it is conspicuous: we loose – in contrast to the greedy sequence – a bit at the fourth member, but – as we will see – our patience effects later improvement.

Using this new sequence we construct our lists as follows. Let $A$ be an on-line algorithm. In the first step we consider a concatenated list with sub-lists $L_1, L_2, \ldots, L_k$ for $k \geq 3$ as follows.

(i) $L_k$ contains $nr$ elements of size $a_k = \frac{1}{b_1} + \varepsilon$,
(ii) $L_{k-1}$ contains $n$ elements of size $a_{k-1} = \frac{1}{b_2} + \varepsilon$,
(iii) $L_j$ contains $n$ elements of size $a_j = \frac{1}{b_{k-j+1}} + \varepsilon$, where $2 \leq j \leq k-2$
(iv) $L_1$ contains $n$ elements of size $a_1 = \frac{1}{b_k} + \varepsilon$,

where $\varepsilon \leq \frac{1}{(k+r)b_k(b_k-1)}$, and $n = c(b_k - 1)$, for some integer $c \geq 1$. So, the constants what we will apply while we use the Theorem 2 are $c_j = 1$, if $j \leq k-1$, and $c_k = r$.

Note that for fixed $k \leq 4$ this definition gives the same lists, which are used in the proof of the van Vliet's lower bound.

If one tries to prove that these sequence of the lists results in a better lower bound, of course the LP method established by van Vliet in [13] is adaptable. Indeed, we also constructed this LP. But – as we mentioned above - the proof of the cited paper seemed to be rather complicated, and so we tried to apply our packing pattern technique. To do that, the only question was whether we could find the correct values of $\alpha$-s and $\beta$-s. (To find a good lower bound for the optimum was not difficult.) Before proving our main theorem we prove some lemmas.

**Lemma 2.** *For the optimum values of the concatenated lists the following relations hold*

*(i)* $OPT(L_1 \dots L_j) \leq \frac{n}{b_1 b_2 b_3^{k-j-2}}$, *for* $1 \leq j \leq k-2$,
*(ii)* $OPT(L_1 \dots L_{k-1}) \leq \frac{n}{b_1} = \frac{n}{r+1}$,
*(iii)* $OPT(L_1 ... L_k) \leq n$,

Based on the above Lemma, we can choose the values of $U_j^k$ as follows.

$$U_j^k = \begin{cases} \frac{1}{b_1 b_2 b_3^{k-j-2}} & \text{, if } j \leq k-2, \\ \frac{1}{b_1} = \frac{1}{r+1} & \text{, if } j = k-1, \\ 1 & \text{, if } j = k \end{cases}$$

For a given $k$ we define two $k$-dimensional vectors $\beta^k$ and $\alpha^k$, as follows.

– If $k \leq 4$ we use the same constants as in the new proof of the van Vliet's lower bound.
– If $k \geq 5$, then

$$\beta_j^k = \begin{cases} 1 & \text{, if } \quad j = 1 \\ b_1 b_2 & \text{, if } \quad j = 2 \\ b_3 \beta_{j-1}^k & \text{, if } \quad 3 \leq j \leq k-2, \\ b_1 \beta_{k-2}^k & \text{, if } \quad k-1 \leq j \leq k. \end{cases}$$

$$\alpha_j^k = \begin{cases} b_1 b_2 & \text{, if} \quad j = 1 \\ (b_1 b_2)^2 & \text{, if} \quad j = 2 \\ b_3 \alpha_{j-1}^k & \text{, if} \quad 3 \le j \le k-3, \\ \beta_{k-1}^k & \text{, if} \quad k-2 \le j \le k-1, \\ r\beta_k^k & \text{, if} \quad j = k. \end{cases}$$

Considering the above constants we need to prove for every feasible packing that inequality (5) holds. Let us suppose that the packing pattern $p = (0, ..., 0, p_i, ..., p_k)$ belongs to the subset $P_i$ of the feasible packings. The packing pattern $p$ is *dominant* in $P_i$ if

$$a_i + \sum_{j=1}^{k} a_j p_j > 1.$$

Let $D_i(p)$ be the set of those packing patterns for which $p$ is dominant in $P_i$. So, it is enough to investigate the dominant patterns for each $P_i$. See for example [11].

**Lemma 3.** *Let $L = L_1 \dots L_k$ be the above defined concatenated list for some $k \ge 5$. Then for every feasible dominant packing pattern $p \in P_i$*

$$\sum_{j=i}^{k} \beta_j^k p_j \le \sum_{j=i}^{k} \alpha_j^k. \tag{10}$$

Now we are ready to prove the new lower bound.

**Theorem 4.** *Let $r$ be a positive integer, and we consider the parametric bin packing problem, i.e. $a_i \le \frac{1}{r}$, if $a_i \in L$. Then there is no one-dimensional on-line bin packing algorithm $A$ with an asymptotic performance ratio*

$$R_A < \frac{r^6 + 8r^5 + 29r^4 + 60r^3 + 75r^2 + 55r + 20}{r^6 + 7r^5 + 22r^4 + 40r^3 + 45r^2 + 33r + 13}.$$

At the end of the section we give a table which displays the new lower bounds for the asymptotic competitve ratio of on-line algorithms for some values of $r$.

## 5 Improved Lower Bound for Decreasing Lists

For those lists where the elements are preprocessed according to their sizes in decreasing order Csirik et al. [1] proved that there is no on-line algorithm with better asymptotic performance ratio than $\frac{8}{7}$. Their construction based on 2 lists which contain elements with sizes $\frac{1}{3}+\varepsilon$ and $\frac{1}{3}-2\varepsilon$, respectively. The last 3 decades there was no success to give a better lower bound. The difficulty originates from the fact that the sizes of the last list in the concatenated list may not be too small, since they may fill up the opened bins, resulting a better packing than

**Table 4.** The new lower bounds for on-line bin packing algorithms

| | $r=1$ | $r=2$ | $r=3$ | $r=4$ | $r=5$ |
|---|---|---|---|---|---|
| $k=3$ | 1,5000000 | 1,3793103 | 1,2878787 | 1,2283464 | 1,1880733 |
| $k=4$ | 1,5390070 | 1,3895759 | 1,2914337 | 1,2298587 | 1,1888167 |
| $k=5$ | 1,5403448 | 1,3896631 | 1,2914442 | 1,2298607 | 1,1888172 |
| $k=6$ | 1,5403721 | 1,3896636 | 1,2914443 | 1,2298607 | 1,1888172 |
| $k=7$ | 1,5403726 | 1,3896636 | 1,2914443 | 1,2298607 | 1,1888172 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $k=\infty$ | 1,5403726 | 1,3896636 | 1,2914443 | 1,2298607 | 1,1888172 |
| $k=\infty$ | $\frac{248}{161}$ | $\frac{1694}{1219}$ | $\frac{7502}{5809}$ | $\frac{24992}{20321}$ | $\frac{68420}{57553}$ |

in the earlier step was. So, there is no point about investigation concatenated lists with $k$ different sub-lists with $k \to \infty$, while the sizes of elements getting progressively smaller and smaller. As a further application of the Theorem 2 here we give a construction with 3 different lists. (Using 4 sub-lists we were unsuccessful.) During our proof we will use again the condition that the sub-lists must not have the same lengths.

Let $A$ be an on-line algorithm. We consider a concatenated list with three sub-lists $L_1, L_2$ and $L_3$.

- $L_1$ contains $n_1$ elements of size $\frac{7}{24} - 4\varepsilon$,
- $L_2$ contains $n_2$ elements of size $\frac{5}{24} + \varepsilon$,
- $L_3$ contains $n_3$ elements of size $\frac{4}{24} + \varepsilon$,

where $\varepsilon < \frac{1}{96}$, $n_1 = n_2 = 6n$ and $n_3 = 18n$. It means that $c_1 = c_2 = 6$ and $c_3 = 18$. It is easy to see that the following inequalities are true.

$$\mathrm{OPT}(L_1) \leq 2n, \quad \mathrm{OPT}(L_1L_2) \leq 3n, \quad \mathrm{OPT}(L_1L_2L_3) \leq 6n.$$

So, we can set the upper bounds to

$$U_1 = 2, \qquad U_2 = 3, \qquad U_3 = 6.$$

In fact, these upper bounds are sharp. Let us now consider the following constants.

$$\alpha_1 = 4, \qquad \alpha_2 = 3, \qquad \alpha_3 = 5$$

and

$$\beta_1 = 4, \qquad \beta_2 = 2, \qquad \beta_3 = 1.$$

Considering the above constants we need to prove that for every dominant packing pattern inequality (5) holds. Since the number of dominant patterns is small we can investigate all of them. Three cases have to be distinguished.

(i) For $i = 1$, we consider the dominant patterns in $P_1$. We need to prove that any feasible packing pattern $p \in P_1$ satisfies

$$12 \geq 4p_1 + 2p_2 + p_3.$$

The dominant patterns are (3,0,0), (2,2,0), (2,1,1), (2,0,2), (1,3,0), (1,2,1), (1,1,3) and (1,0,4). It is easy to check that the inequality holds for all of them.

(ii) For $i = 2$, the dominant patterns of bins in $P_2$ have to be considered. These are (0,4,0), (0,3,2), (0,2,3) and (0,1,4). All of them satisfy

$$8 \geq 2p_2 + p_3.$$

(iii) Finally, we have to address the packing patterns in $P_3$. The only dominant pattern is (0,0,5) and the inequality

$$5 \geq p_3$$

trivially holds.

So, the conditions of Theorem 2 hold and therefore

$$R_A \geq \frac{\sum_{j=1}^{3} c_j \beta_j}{\sum_{j=1}^{k} \alpha_j U_j} = \frac{24 + 12 + 18}{8 + 9 + 30} = \frac{54}{47}.$$

We can summarize our calculation in the following theorem.

**Theorem 5.** *No on-line algorithm for the one-dimensional bin packing problem which packs the elements in decreasing order and can have better asymptotic performance ratio than* $\frac{54}{47} = 1.1489361\ldots$.

## 6 Conclusions

In this paper we improved two old lower-bound results for certain classes of one-dimensional bin packing algorithms. For on-line algorithms we considered the parametric case and the new lower bounds are summarized for some positive integers $r$ in Table 4. For those semi–online bin packing algorithms, which allow preordering, and they get the elements in decreasing order our new lower bound is $\frac{54}{47} = 1.1489361\ldots$ vs. $\frac{8}{7} = 1.142857\ldots$. As a "byproduct" we gave a simple combinatorial proof for van Vliet's lower bound for the performance of on-line algorithms.

For the latter case we note that if the size of the largest elements is in the interval $(\frac{8}{29}, \frac{1}{2}]$ then First Fit Decreasing yields the upper bound $\frac{71}{60}$ (see [8]). However, our efforts to get a better result were not successful so far. It is true, that our improvements are very small in absolute values. However we did an exhaustive search for possible lists and we have not found worse ones. We strongly believe that the gap might be decreased only by defining better algorithms or one needs to find a new method for proving lower bounds.

The packing pattern technique was used for the two- and three-dimensional bin packing problems [6] and the best known lower bound for the on-line vector packing algorithms operates also with this technique (see [5]). Since in these cases the Sylvester sequence were used during the proof it is plausible that the application of the new series will also improve these lower bounds. We are convinced that this technique is usable for other classes of algorithms.

## References

1. Csirik, J., Galambos, G., Turán, G.: Some Results on Bin Packing. In: Proceedings of EURO VI., Vienna (1983)
2. Galambos, G.: Parametric Lower Bound for On-line Bin Packing. SIAM J. Algebraic Discrete Methods, 362–367 (1986)
3. Galambos, G.: A 1.6 Lower Bound for the Two-dimensional On-line Rectangle Bin Packing. Acta Cybernetica 10, 21–24 (1991)
4. Galambos, G., Frenk, J.B.G.: A Simple Proof of Liang's Lower Bound for On-line Bin Packing and Extension to the Parametric Case. Discrete Applied Mathematics 41, 173–178 (1993)
5. Galambos, G., Kellerer, H., Woeginger, G.: A Lower Bound for On-line Vector-packing Algorithms. Acta Cybernetica 11, 23–34 (1993)
6. Galambos, G., van Vliet, A.: Lower Bounds for 1-, 2- and 3-dimensional On-line Bin Packing Algorithms. Computing 52, 281–297 (1994)
7. Garey, M.R., Johnson, D.S.: Computer and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman, New York (1979)
8. Johnson, D.S.: Fast Approximation Algorithms for Bin Packing. J. Comput. Syst. Sci. 8, 272–314 (1974)
9. Liang, F.M.: A Lower Bound for On-line Bin Packing. Inf. Proc. Letters 10, 76–79 (1980)
10. Seiden, S.: On the On-line Bin Packing Problem. Journal of ACM 49, 640–671 (2002)
11. Seiden, S., van Stee, R., Epstein, L.: New Bounds for Variable-sized Online Bin Packing. SIAM J. on Computing 32, 455–469 (2003)
12. Sylvester, J.: On a Point in the Theory of Vulgar Fractions. American Journal of Mathematics 3, 332–335 (1880)
13. van Vliet, A.: An Improved Lower Bound for On-line Bin Packing Algorithms. Inf. Proc. Letters 43, 274–284 (1992)
14. van Vliet, A.: Lower Bound and Upper Bounds for On-line Bin Packing and Scheduling Algorithms, PhD Thesis, Tinbergen Institute Research Series no. 93

# On the Approximation Complexity Hierarchy

Magnus Bordewich

School of Engineering and Computing Sciences, Durham University, U.K.
m.j.r.bordewich@durham.ac.uk

**Abstract.** This paper presents an extension of Ladner's Theorem to the approximation complexity hierarchy. In 1975 Ladner proved that if P≠NP, then there exists an incomplete problem A which is neither in P nor NP-complete. Here we show that if RP≠NP, then there is a counting problem $\pi_A$ which neither has a *fully polynomial randomised approximation scheme* (FPRAS), nor is as hard to approximate as #SAT.

This work is motivated by recent results showing that approximately counting $H$-colouring problems appears to fall into three complexity groups. Those problems which admit an FPRAS, those which are 'AP-interreducible' with #SAT and an intermediate class of problems all AP-interreducible with #BIS (counting independent sets in bipartite graphs). It has been asked whether this intermediate class in fact collapses into one of the former two classes, or whether it truly occupies some middle ground. Moreover, supposing it does occupy some middle ground, does it capture all the ground between?

Our results reveal that there are counting problems whose approximation complexity lies between FPRASable and #SAT, under the assumption that NP≠RP. Indeed, there are infinitely many complexity levels between. Moreover we show that if #BIS is genuinely in the middle ground (neither having an FPRAS, nor as hard to approximate as #SAT), then there are problems that do not admit an FPRAS, are not equivalent in approximation complexity to #BIS and are not 'AP-interreducible' with #SAT, thus also occupy the middle ground.

The proof is based upon Ladner's original proof that there are classes between P and NP, and suffers the same drawback that the problems constructed are not natural. In particular our constructed problems are not $H$-colourings. The question of the approximation complexity of #BIS remains open.

## 1 Introduction

One of the most intriguing aspects of complexity theory is that almost every 'natural' problem can be shown to be either in the class P (solvable in polynomial time), or to be NP-complete (as hard to solve as boolean satisfiability (SAT) and many other well known hard problems). Of the thousands of decision problems analysed over the years the notable exceptions to this, natural problems that have not been shown to be in either group, are graph isomorphism testing and factoring (input $(n, t)$: does $n$ have a factor between 2 and $t$?). One

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 37–46, 2011.

might reasonably wonder whether in fact all problems in NP are either in P or are NP-complete. In 1975 Ladner showed that this is not the case, in what has become one of the fundamental theorems in complexity theory [5]. He showed that under the assumption that P≠NP, then we can construct a problem which is neither in P nor is NP-complete, using a technique of delayed diagonalisation. Although Ladner's Theorem shows that there is an infinite hierarchy of distinct complexity levels between P and NP, it remains the case that almost every 'natural' problem that has been encountered has been shown to be either in P or NP-complete. The problem generated in Ladner's proof is a somewhat complex interweaving of a trivial and an NP-complete problem, taking enough of each to be neither NP-complete itself, nor polynomial time decidable.

In recent years there has been great interest in the class of counting problems #P introduced by Valiant in 1979 [6], and approximation algorithms for problems in #P. One of the triumphs of research in the area has been the steady progress towards classifying the complexity of counting graph homomorphisms, known as $H$-colouring problems. In this setting the graph $H$ is fixed and we consider the problem of counting the number of homomorphisms from an input graph $G$ to the fixed graph $H$ ($H$-colourings of $G$). This framework captures many standard graph problems. For example: if $H$ is a complete graph on $k$ vertices with no loops, then counting homomorphisms from $G$ to $H$ is equivalent to counting proper $k$-colourings of $G$; if $H$ is a graph on two vertices with an edge between and a loop on one vertex, then counting $H$-colourings of $G$ is equivalent to counting independent sets of $G$. See [4] for further background on $H$-colourings.

In 1990 Hell and Nešetřil [4] considered the decision problem of determining whether there are any homomorphisms from $G$ to $H$. They showed that there is a complexity dichotomy between graphs $H$ for which the decision problem in is P, and those for which it is NP-complete. In 2000 Dyer and Greenhill [3] considered the problem of counting $H$-colourings. They were able to completely characterise the graphs $H$ for which this problem is #P-complete. They defined a *trivial* connected component of $H$ to be one that is a complete graph with all loops present or a complete bipartite graph with no loops present, and showed that counting $H$-colourings is #P-complete if $H$ has a nontrivial component and that it is in P otherwise. In 2003 Dyer *et. al.* [2] turned their attention to the relative complexity of approximate counting problems, introducing the idea of approximation preserving reductions. They showed that there appear to be three distinct classes of problems relative to such reductions: those that can be approximated in polynomial time, those that are as hard to approximate as #SAT, and a logically defined intermediate class of problems that are equivalent in approximation complexity to approximately counting independent sets in a bipartite graph (#BIS).

In this paper we show that it cannot be the case that there is a neat trichotomy (or even dichotomy) of approximation complexity in general. That is, if we assume that we cannot approximate the number of solutions to an instance of SAT in random polynomial time (it is enough to assume RP≠NP, see [7]), then there must be an infinite hierarchy of approximation complexity classes between

those that may be approximated and #SAT. In particular, if #BIS lies in this intermediate complexity region, then there are problems that may be reduced to #BIS, but to which #BIS will not reduce, also in the intermediate region. We note that the problems constructed are not $H$-colouring problems. Therefore this work does not preclude the possibility of a trichotomy or dichotomy of approximation complexity for $H$-colouring problems.

The intuitive idea of Ladner's proof is to construct a specific decision problem, then enumerate all the polynomial time Turing machines and show that each in turn cannot either solve the decision problem or reduce SAT to the constructed problem. Here we proceed in the same manner, except that we are dealing with probabilistic Turing machines and require a bounded probability of erroneous response. Since there is no effective enumeration of bounded error probabilistic Turing machines, it may seem that we are stuck. However all we require is an effective enumeration of all polynomial time probabilistic Turing machines, and we show that none of these can give an FPRAS for our constructed problem, or the required reduction. Fortunately polynomial time probabilistic Turing machines, with no bound on the error, can be efficiently enumerated. For a discussion of enumeration of probabilistic Turing machines and the difficulty of enumerating bounded error probabilistic Turing machines see, for example, Du and Ko [1].

## 2 Preliminaries

A counting problem $\pi$ is in the class #P if there is some non-deterministic Turing machine $M$, constant $k$ and polynomial $p$ such that for any input $x$ the number of valid witnesses for $x$ is exactly $\pi(x)$, all valid witnesses have length exactly $|x|^k$ and $M$ runs in time bounded by $p(|x|)$. It follows that for any problem in NP, there is an associated problem of counting the number of solutions in #P. For example, determining whether a graph has a proper $k$-colouring is an NP problem; counting the number of proper $k$-colourings is a #P problem. A *randomised approximation scheme* (RAS) for a counting problem $\pi$ is an algorithm or oracle (black box), which given any input $(x, y)$ returns a value in $[(1-y^{-1})\pi(x), (1+y^{-1})\pi(x)]$ with probability at least $3/4$. Thus, we are taking $\epsilon = y^{-1}$ to be a relative error parameter. A *fully polynomial randomised approximation scheme* (FPRAS) for $\pi$ is an RAS algorithm for $\pi$ with running time polynomial in both $|x|$ and $y$. From here on, when referring to an input $(x, y)$ we assume that $y$ is encoded in unary, so that $|(x, y)| = |x| + y$ and hence polynomial time in the input size is the correct measure for FPRAS.

We will consider two types of special Turing machines: *probabilistic Turing machines* taking two inputs (PTMs) and *probabilistic oracle Turing machines* taking two inputs (POTMs). Here two inputs means the input is $(x, y)$, where $y^{-1}$ is a relative error parameter as above. Let $M_1, M_2, \ldots$ be an enumeration of *polynomial time* PTMs and $M'_1, M'_2, \ldots$ be an enumeration of *polynomial time* POTMs. For simplicity we assume that the machines are clocked so that $M_i$ and $M'_i$ run in time $n^i$ on all inputs of size $n$. For the probabilistic aspect of our Turing machines, we assume that each Turing machine has an auxiliary

input tape of length $n^i$ containing random bits. Thus a problem $\pi$ has an FPRAS if and only if there is some $i$ for which the machine $M_i$ applied to any input $(x, y)$ returns a value in $[(1 - y^{-1})\pi(x), (1 + y^{-1})\pi(x)]$ on at least 3/4 of the possible strings of random bits. We describe a response in this range as a *good* response for $\pi$.

A POTM operates exactly as a standard PTM, except that it can make oracle calls during its run. Each oracle call must be computed by the PTM, so has polynomially bounded size, and the response from the oracle must be read, so also has polynomially bounded size. For our purposes we are interested in RAS oracles for counting problems. An RAS oracle for $\pi$ is a black box about which we know only that when called with $(x, y)$, it returns an integer $z \in [0, 2^{|x|^k}]$ such that $\Pr[z \in [(1-y^{-1})\pi(x), (1+y^{-1})\pi(x)]] \geq 3/4$. Note that we include the possibility that the oracle is an omnipotent adversary who can see your other calculations and objectives in addition to the call $(x, y)$, and is only constrained by having to give a good response with probability at least 3/4.

We will also be interested in two sub-classes of RAS oracles: *binary* RAS oracles and *restricted* RAS oracles. A binary RAS oracle is an RAS oracle such that the oracle determines two possible responses, $z, z' \in [0, 2^{|x|^k}]$, and responds $z$ with probability 1/4 and $z'$ (necessarily in $[(1 - y^{-1})\pi(x), (1 + y^{-1})\pi(x)]$) with probability 3/4. (We allow $z = z'$, which then has probability 1.) This can be thought of as an adversary giving as much probability as permitted to the least helpful responses. Again, we describe a response of the oracle in the range $[(1-y^{-1})\pi(x), (1+y^{-1})\pi(x)]$ as a *good* response. The reader may be concerned that these definitions allow an oracle to use a different probability distribution to respond to a given input depending on external factors such as the state of the POTM making the call. As an alternative we also consider restricted RAS oracles. We suppose the RAS oracle for a given problem $\pi$ is assumed to be a probabilistic Turing machine (with possibly exponential running time) hidden in a black box. We assume that the running time for input $(x, y)$ is bounded by $p(|x|)2^{|x|^k}$ for some constant $k$ and polynomial $p$, since we can compute $\pi(x)$ exactly in this time by checking every possible witness. Since the probabilistic nature of a PTM can be captured by an auxiliary tape of random bits, in this case of length at most $2^{|x|^k}$, the probability of any specific response must be a multiple of $2^{-2^{|x|^k}}$. Hence we define a restricted RAS oracle for a counting problem $\pi$ as an RAS oracle for which the probability of any specific response depends only on the call $(x, y)$, and for some fixed constant $k$ is always a multiple of $2^{-2^{|x|^k}}$.

We next define *approximation preserving* reductions. A counting problem $\pi_1$ is AP-reducible to $\pi_2$ (denoted $\pi_1 \leq_{AP} \pi_2$) if for some $i$ and any $(x, y)$ the machine $M_i'$ applied to input $(x, y)$ returns a value in $[(1 - y^{-1})\pi_1(x), (1 + y^{-1})\pi_1(x)]$ on at least 3/4 of the possible strings of random bits, where the oracle calls made by $M_i'$ are to any RAS oracle for $\pi_2$. If $\pi_1 \leq_{AP} \pi_2$ and $\pi_2 \leq_{AP} \pi_1$ we say $\pi_1$ and $\pi_2$ are AP-interreducible. We will also consider binary AP-reductions ($\leq_{bAP}$) and restricted AP-reductions ($\leq_{rAP}$) where the oracle calls are to any

binary RAS oracle, and any restricted RAS oracle, respectively. We will also write $\pi_1 <_{AP} \pi_2$ ($\pi_1 <_{rAP} \pi_2$) to indicate that $\pi_1 \leq_{AP} \pi_2$ but $\pi_2 \not\leq_{AP} \pi_1$ ($\pi_1 \leq_{rAP} \pi_2$ but $\pi_2 \not\leq_{rAP} \pi_1$, respectively).

## 3 Results

We may now state the main theorem.

**Theorem 1.** *Let $\pi_1$ be a #P problem such that there is no FPRAS for $\pi_1$. Then we can construct a #P problem $\pi_A$ such that*

*(i) there is no FPRAS for $\pi_A$,*
*(ii) $\pi_A <_{AP} \pi_1$.*

*In words: $\pi_A$ is of intermediate approximation complexity between FPRASable and $\pi_1$.*

This general theorem leads to the following corollaries.

**Corollary 1.** *If NP≠RP then there are an infinite number of problems $\pi_{A_1}$, $\pi_{A_2}, \ldots$ in #P such that*

*(i) for all $i$, $\pi_{A_i}$ does not have an FPRAS,*
*(ii) for all $i$, #SAT is not AP-reducible to $\pi_{A_i}$, and*
*(iii) for all $i, j$ such that $i < j$, we have $\pi_{A_j} <_{AP} \pi_{A_i}$.*

*Proof.* If NP≠RP then there is no FPRAS for #SAT [7]. Thus we can apply Theorem 1, taking $\pi_1$ to be #SAT, to obtain $\pi_{A_1}$ of intermediate approximation complexity. We then iterate the arguments, for $i \geq 1$, taking $\pi_1 = \pi_{A_i}$ to obtain $\pi_{A_{i+1}}$ of intermediate complexity between FPRASable and $\pi_{A_i}$.

If we make the additional assumption that #BIS does not have an FPRAS and #SAT is not AP-reducible to #BIS then we obtain the following corollary.

**Corollary 2.** *If there is no FPRAS for* #BIS*, and* #SAT $\not\leq_{AP}$ #BIS*, then there exists a problem $\pi_A$ in #P that does not have an FPRAS and such that* $\pi_A <_{AP}$ #BIS $<_{AP}$ #SAT*.*

*Proof.* We apply Theorem 1 taking $\pi_1$ to be #BIS, to obtain $\pi_A$.

## 4 Proofs

Here we give the proof of Theorem 1. First we show that we may consider only binary RAS oracles in the proofs.

**Lemma 1.** *For two counting problems $\pi_1$ and $\pi_2$, $\pi_1 \leq_{AP} \pi_2$ if and only if $\pi_1 \leq_{bAP} \pi_2$.*

*Proof.* Suppose first that $\pi_1 \leq_{AP} \pi_2$. Then there is some POTM $M_i'$ which has probability at least 3/4 of returning a good answer for $\pi_1$ as long as $M_i'$ makes calls to any RAS oracle for $\pi_2$. Since binary RAS oracles are RAS oracles, $M_i'$ returns a good answer with sufficient probability whenever the calls are to any binary RAS oracle. Hence $\pi_1 \leq_{bAP} \pi_2$.

Suppose now that $\pi_1 \not\leq_{AP} \pi_2$, and that $\pi_2$ has witnesses of length $k$. Then for each $i$ there is some input $(x, y)$ and some RAS oracle O, such that $M_i'(x, y)$ has less than probability 3/4 of returning a good response when the oracle calls are to O. We may regard the calculation of $M_i'(x, y)$ as branching at each use of a random bit and at each response of the oracle. Down any computation path, consider the first oracle call $(x', y')$. For each response of the oracle, there is some probability that the calculation of $M_i'$ continued from this point results in a good response. There will be some possible response of the oracle $z \in [0, 2^{|x'|^k}]$ which minimises the probability that the final response of $M_i'$ will be good for $\pi_1$, and likewise some possible good response of the oracle $z' \in [(1 - y'^{-1})\pi_2(x'), (1 + y'^{-1})\pi_2(x')]$ which minimises this same probability. Consider an oracle O$'$ which responds exactly as O except in this specific call at this point of the computation of $M_i'$, when it responds $z$ with probability 1/4, and $z'$ with probability 3/4. Then O$'$ is still a valid RAS oracle for $\pi_2$, since it always gives a good response with probability at least 3/4. Moreover, the probability that the computation of $M_i'(x, y)$ fails to be good is at least as great using oracle O$'$ as O. Iterating this argument at all oracle calls down all computation paths, we see that if there is any RAS oracle such that $M_i'(x, y)$ fails to be good for $\pi_1$ with probability greater than 1/4 when calling this oracle, then there is a binary RAS oracle O$''$ such that $M_i'(x, y)$ fails to be good with probability greater than 1/4 when calling O$''$. Hence $\pi_1 \not\leq_{bAP} \pi_2$.

*Proof (Proof of Theorem 1).* We have a problem $\pi_1$ in #P such that $\pi_1$ does not admit an FPRAS. We take a fixed non-deterministic Turing Machine for $\pi_1$ which has witnesses of size exactly $|x|^{k_1}$, and a constant $c > 2k_1$ such that the running time of this NDTM is bounded by $|x|^c$. We define the counting problem $\pi_A$ to be

$$\pi_A(x) = \begin{cases} 0 & \text{if } f(|x|) \text{ is odd,} \\ \pi_1(x) & \text{if } f(|x|) \text{ is even,} \end{cases}$$

where $f : \mathbb{N} \mapsto \mathbb{N}$ is a polynomial time computable function which we will define shortly.

The problem $\pi_A$ is in #P since $f$ is polynomial time computable and hence there is a polynomial time NDTM which first computes $f(|x|)$ and then outputs either zero or proceeds exactly as the NDTM for $\pi_1$, as appropriate. Note that witnesses for $\pi_A$ therefore have length exactly $|x|^{k_1}$. It also follows from the definition that $\pi_A \leq_{AP} \pi_1$, since given an RAS oracle for $\pi_1$, we may approximate $\pi_A(x, y)$ by again computing $f(|x|)$, and then either outputting zero or returning the result of an oracle call for $\pi_1(x, y)$ as appropriate.

The function $f$ is defined recursively as follows. Set $f(n) = n$ for $n \leq 2$. For $n \geq 2$ define $f(n + 1)$ according to the cases below:

(i) If $(\log\log n)^{cf(n)} \geq \log n$ set $f(n+1) = f(n)$;

(ii) if $f(n) = 2i$ check to see if there is an input $(x, y)$, of size $|(x, y)| \leq \log\log n$ such that the probability that $M_i(x, y)$ gives a good response for $\pi_A$ is less than 3/4. If such an input exists set $f(n+1) = f(n) + 1$, otherwise set $f(n+1) = f(n)$.

(iii) if $f(n) = 2i + 1$ check to see if there is an input $(x', y')$, of size $|(x', y')| \leq \log\log n$ and an RAS oracle O′ for $\pi_A$, such that the probability that $M_i'(x', y')$ gives a good response for $\pi_1$ is less than 3/4 when the oracle calls are to O′. If such an input and oracle exist set $f(n+1) = f(n) + 1$, otherwise set $f(n+1) = f(n)$.

We must check that $f(n)$ may be computed in time polynomial in $n$. It is sufficient to show that we can check each of the conditions above in polynomial time. Checking condition (i) is polynomial in $n$. For (ii) there are $2^{\log\log n}$ inputs $(x, y)$ to be checked. For each input $|x|$ is much smaller than $n$, so we have already recursively calculated $f(|x|)$. Thus we can compute $\pi_A(x)$ by determining $f(|x|)$ and checking all possible witnesses for $\pi_1(x)$, if appropriate ($2^{(\log\log n)^{k_1}}$ possible witnesses, each checked in time at most $|x|^c \leq (\log\log n)^c$). Next we must determine whether the probability that $M_i(x, y)$ gives a good response (for $\pi_A$) is less than 3/4.

We shall do this by simulating the computation of $M_i(x, y)$, branching each time a random bit is used. Each random bit will lead to two branches, thus there are at most $2^{|(x,y)|^i}$ possible computation paths for the given input $(x, y)$. We follow each of these and compute the probability that $M_i(x, y)$ gives a good response. Thus (ii) can be checked in time $O(2^{(\log\log n)^{ic}})$. Since $i$ is bounded by $f(n)$, by (i) we have $2^{(\log\log n)^{ic}} \leq n$.

Similarly we can check condition (iii) in polynomial time: we first compute $\pi_1(x')$, and then we simulate $M_i'(x', y')$, now also branching at each oracle call of the computation, to see if it returns a good response with probability less than 3/4. By Lemma 1 we need only consider binary RAS oracles for $\pi_A$. Each oracle call has input $(x'', y'')$ of length bounded by $|(x', y')|^i$ and output in the range $[0, 2^{|x''|^{k_1}}]$. Thus the number of valid distributions of a binary RAS oracle for a given call is bounded by $2^{2|x''|^{k_1}}$. We can compute which are valid by determining $\pi_1(x'')$ exactly, through checking every possible witness of which there are at most $2^{|x''|^{k_1}}$. Hence we branch into at most $2^{2|x''|^{k_1}}$ possible distributions for the oracle at each oracle call, and then into two further branches for the randomly chosen response of the oracle. Any fixed choice of distributions at the oracle calls corresponds to a possible binary RAS oracle. For each fixed set of such choices, we may compute the probability of following a computation path which ends in a good response. Recalling that both the size and number of oracle calls are bounded by $|(x', y')|^i$, there are at most $(2^{2|(x',y')|^{ik_1}})^{|(x',y')|^i} \leq 2^{|(x',y')|^{ic}}$ possible binary RAS oracles we need to check for the given input $(x', y')$. Thus (iii) can be checked in time $O(2^{(\log\log n)^{ic}})$. Since $i$ is bounded by $f(n)$, by (i) we have $2^{(\log\log n)^{ic}} \leq n$.

We now conclude the proof by showing that $\pi_A$ does not admit an FPRAS and $\pi_1 \not\leq_{AP} \pi_A$. This will follow because if either part does not hold, then $f$ is bounded and in this case we can construct an FPRAS for $\pi_1$. Suppose first that there is an FPRAS for $\pi_A$. Then for some $i$ the machine $M_i$ gives a good response with probability at least 3/4 on every input $(x, y)$. By condition (ii) $f(n)$ will never grow larger than $2i$. Now suppose that $\pi_1 \leq_{AP} \pi_A$. Then for some $j$ the machine $M'_j$ equipped with any RAS oracle for $\pi_A$ gives a good response for problem $\pi_1$ with probability at least 3/4 on every input $(x', y')$. By condition (iii) $f(n)$ will never grow larger than $2j + 1$. In either case, $f(n)$ is bounded, and therefore constant for all sufficiently large $n$. If $f(n)$ is even for all sufficiently large $n$, then some PTM gives an FPRAS for $\pi_A$, but $\pi_A(x) \neq \pi_1(x)$ only on a constant number of inputs $x$. Thus we must also have an FPRAS for $\pi_1$, which contradicts the assumptions of the theorem. Alternatively if $f(n)$ is odd for all sufficiently large $n$, then some POTM gives an approximation preserving reduction from $\pi_1$ to $\pi_A$, but $\pi_A(x) \neq 0$ only on a constant number of inputs $x$. Thus $\pi_A$ is polynomial time computable, and all oracle calls to an RAS oracle for $\pi_A$ may be replaced with a polynomial time computation. So again there must be an FPRAS for $\pi_1$, contradicting the assumptions of the theorem.

### 4.1 Restricted Power Oracles

For the reader concerned that the definition of RAS oracle allows the potential use of a different probability distribution to respond to a given call depending on external factors such as the state of the POTM making the call, we now prove a version of our results only using restricted RAS oracles.

For a binary RAS oracle to problem $\pi$, taking witnesses of length $k$, there are at most $2^{2|x|^k}$ possible distributions of the oracle responses to a given call $(x, y)$. For restricted RAS oracles we can no longer assume that only two responses have non-zero probability; we are restricted to assuming that that the probability of each response is a multiple of $2^{-2^{|x|^k}}$. Thus there are at most $(2^{2^{|x|^k}})^{2^{|x|^k}} = 2^{2^{2|x|^k}}$ possible distributions for the oracle responses. However this additional exponential factor need cause us no concern. We can simply delay the diagonalisation a little further: when defining $f(n+1)$ we only check inputs up to size $\log\log\log n$ (rather than $\log\log n$) in conditions (ii) and (iii). This means we can do more checking and still compute $f$ in polynomial time. What we lose in doing this is the rate of growth of $f$. However all we require is that $f$ is unbounded, how quickly it grows is irrelevant. Indeed by using a parameter $l$ in the proof, for the height of an exponential tower, we demonstrate that $f$ could be defined to grow considerably more slowly if required.

**Theorem 2.** *Let $\pi_1$ be a #P problem such that there is no FPRAS for $\pi_1$. We can construct a #P problem $\pi_A$ such that $\pi_A <_{rAP} \pi_1$, and there is no FPRAS for $\pi_A$. In other words: $\pi_A$ is of intermediate (restricted) approximation complexity between FPRASable and $\pi_1$.*

*Proof.* The proof closely follows that of Theorem 1. We take a fixed non-deterministic Turing Machine for $\pi_1$ which has witnesses of size exactly $|x|^{k_1}$,

and a constant $c$ such that the running time of this NDTM is bounded by $|x|^c$ and $c > 2k_1$. Let $\pi_A$ and $f : \mathbb{N} \mapsto \mathbb{N}$ be defined as follows.

$$\pi_A(x) = \begin{cases} 0 & \text{if } f(|x|) \text{ is odd,} \\ \pi_1(x) & \text{if } f(|x|) \text{ is even,} \end{cases}$$

and, for some constants $c$ (defined below) and $l \geq 3$, $f$ is defined as follows. Set $f(n) = n$ for $n \leq 2$. For $n \geq 2$ define $f(n+1)$ according to the cases below:

(i) If $(\log^l n)^{cf(n)} \geq \log^{l-1} n$ set $f(n+1) = f(n)$;

(ii) if $f(n) = 2i$ check to see if there is an input $(x, y)$, of size $|(x, y)| \leq \log^l n$ such that the probability that $M_i(x, y)$ gives a good response for $\pi_A$ is less than 3/4. If such an input exists set $f(n+1) = f(n) + 1$, otherwise set $f(n+1) = f(n)$.

(iii) if $f(n) = 2i + 1$ check to see if there is an input $(x', y')$, of size $|(x', y')| \leq \log^l n$ and a restricted RAS oracle O′ for $\pi_A$, such that the probability that $M_i'(x', y')$ gives a good response for $\pi_2$ is less than 3/4 when the oracle calls are to O′. If such an input and oracle exist set $f(n+1) = f(n)+1$, otherwise set $f(n+1) = f(n)$.

Again it follows that $\pi_A$ is in #P, as long as we can show that $f(n)$ may be computed in time polynomial in $n$. For this it is sufficient to show that we can check each of the conditions in the definition of $f$ in polynomial time. Conditions (i) and (ii) are essentially unchanged from Theorem 1, except that we have fewer possible inputs to consider in (ii), so may be checked in time polynomial in $n$. In order to check condition (iii) we again proceed as in Theorem 1, but in simulating the computation of $M_i'(x', y')$ for each possible input $(x', y')$ of size $|(x', y')| \leq \log^l n$, we must consider restricted oracles. The branching at each oracle call $(x'', y'')$ is in two phases. Firstly, we branch into at most $2^{2^{2|x''|^{k_1}}}$ computations, one for each possible distribution of responses to the oracle call (distribution branches). Secondly, we branch into at most $2^{|x''|^{k_1}}$ branches depending on the randomly chosen response of the oracle given the distribution (response branches). We then determine whether any choice of distribution branches at the oracle calls, and hence any restricted RAS oracle, results in a weight of probability of less than 3/4 down computation paths (response branches) leading to a good response. Note we must take the same distribution branch each time a given call is made, should any calls be repeated. Since there are at most $|(x, y)|^i$ oracle calls, each of size at most $|(x, y)|^i$, there at most $(2^{2^{2|(x,y)|^{ik_1}}})^{|(x,y)|^i} \leq 2^{2^{|(x,y)|^{ic}}}$ possible oracles to consider. Hence we can check condition (iii) in time $O(2^{2^{|(x,y)|^{ic}}})$. Since $i \leq f(n)$, by condition (i) we have $2^{2^{|(x,y)|^{ic}}} \leq 2^{2^{\log^{l-1} n}} \leq n$. Thus condition (iii) can be checked in polynomial time.

The remainder of the proof follows that of Theorem 1. If $f(n)$ is bounded, then $f$ is constant for large enough $n$ and therefore only differs from 0 or $\pi_1$ in a constant number of places. Either case implies that there is an FPRAS for $\pi_1$,

which is a contradiction. Hence $f(n)$ is unbounded. Therefore conditions (ii) and (iii) are met for every $i$, which in turn implies that there can not be any restricted approximation preserving reduction from $\pi_1$ to $\pi_A$.

## References

1. Du, D.-Z., Ko, K.-I.: Theory of Computational Complexity. Wiley-Interscience, New York (2000)
2. Dyer, M., Goldberg, L.A., Greenhill, C., Jerrum, M.: On the relative complexity of approximate counting problems. Algorithmica 38(3), 471–500 (2003)
3. Dyer, M., Greenhill, C.: The complexity of counting graph homomorphisms. Random Structures and Algorithms 17, 260–289 (2000)
4. Hell, P., Nešetřil, J.: On the complexity of H -coloring. Journal of Combinatorial Theory, Series B 48, 92–110 (1990)
5. Ladner, R.E.: On the Structure of Polynomial Time Reducibility. Journal of the ACM 22(1), 155–171 (1975)
6. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing 8, 410–421 (1979)
7. Zuckerman, D.: On unapproximable versions of NP-complete problems. SIAM Journal on Computing 25, 1293–1304 (1996)

# The Power of Uncertainty: Bundle-Pricing for Unit-Demand Customers

Patrick Briest[1] and Heiko Röglin[2]

[1] Department of Computer Science, University of Paderborn, Germany
patrick.briest@upb.de
[2] Department of Computer Science, University of Bonn, Germany
heiko@roeglin.org

**Abstract.** We study an extension of the unit-demand pricing problem in which the seller may offer bundles of items. If a customer buys such a bundle she is guaranteed to get one item out of it, but the seller does not make any promises of how this item is selected. This is motivated by the sales model of retailers like hotwire.com, which offers bundles of hotel rooms based on location and rating, and only identifies the booked hotel after the purchase has been made.

As the selected item is known only in hindsight, the buying decision depends on the customer's belief about the allocation mechanism. We study strictly pessimistic and optimistic customers who always assume the worst-case or best-case allocation mechanism relative to their personal valuations, respectively. While the latter model turns out to be equivalent to the pure item pricing problem, the former is fundamentally different, and we prove the following results about it: (1) A revenue-maximizing pricing can be computed efficiently in the uniform version, in which every customer has a subset of items and the same non-zero value for all items in this subset and a value of zero for all other items. (2) For non-uniform customers computing a revenue-maximizing pricing is APX-hard. (3) For the case that any two values of a customer are either identical or differ by at least some constant factor, we present a polynomial time algorithm that obtains a constant approximation guarantee.

## 1 Introduction

Algorithmic pricing deals with the problem of efficiently determining revenue-maximizing ways of selling a collection of items given information about the preferences of the potential customers in the target market. The traditional way of selling items consists of posting a price for each individual item, then letting customers pick the bundle of items they prefer and charging them the sum of prices of items they select. The problem of finding such an *item pricing* under various kinds of customer preferences has received a lot of attention recently [1,2,6,7,14] and in many cases, its approximation complexity is quite well understood.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 47–58, 2011.

Two fundamental classes of customer preferences have been particularly well investigated. Customers are referred to as *single-minded*, if items constitute *strict complements* and each customer is interested in purchasing one particular subset of the items. We say that customers are *unit-demand*, if items are *strict substitutes* and each customer is interested in buying a single item out of some set of alternatives. Assuming unlimited supply of all the items, it is known that in both of these settings the revenue-maximizing item pricing can be approximated within factors that are logarithmic in the number of customers or linear in the number of items [14] and, under appropriate complexity theoretic assumptions, no essential improvement beyond these guarantees is possible [4,11].

In a recent paper Briest et al. [5] consider the unit-demand pricing problem, but instead of trying to find a revenue-maximizing item pricing, allow to sell items via a so-called *system of lotteries*. Here, rather than posting prices for individual items, the seller may offer a collection of *lottery tickets*, each such ticket representing a probability distribution over items and an associated price. A customer purchasing some given ticket will be asked to pay its price and in turn receive an item randomly sampled from the ticket's distribution. It is shown in [5] that depending on the details of the underlying model of customer behavior this larger class of selling mechanisms can lead to a significant increase in revenue while simultaneously allowing for much better algorithmic solutions.

While, apart from its intrinsic connection to the process of haggling in price negotiations [16,17], we are not aware of any lottery-like pricing mechanism being applied directly in practice at this point, a related - yet different - method is being employed by several companies. This method also consists of *bundling* subsets of items and pricing bundles of items rather than individual items, and again the understanding is that a unit-demand customer will receive a single item from the bundle she picks. The crucial difference to the lottery-based system described above lies in the fact that the seller does not make any promises as to how the item allocated to the customer will be selected. This might be done according to some probability distribution unknown to the customers, but it might also be done in any other conceivable fashion, e.g. guided by production costs or availability. One prominent example of a retailer employing this kind of pricing scheme is the website hotwire.com, where hotel rooms are bundled according to their location and star rating. Customers can book offers of the form "3 nights in the Philadelphia downtown area, 4 stars and up for $279" and will receive information about the exact hotel they will be staying at only after payment has been made.

Formalizing this kind of unit-demand bundle-pricing problem brings up some modeling issues. Rational unit-demand customers are commonly modeled as assigning a *value* to each of the items and upon observing the item prices selecting the item maximizing their *utility*, defined as the difference between the customer's respective value and its price. While this concept of rationality extends quite naturally to lottery-based pricing (every lottery holds a fixed *expected* utility to a customer), it is not obvious what to do in the bundle pricing setting. A customer's value for a given bundle depends on the actual item she receives

and, thus, will only be known in hindsight and, consequently, the buying decision itself has to depend on the customer's *belief* about the seller's allocation mechanism. In this paper, we will investigate the two most basic ways of modeling these beliefs and assume that customers assign to each bundle either the minimum or maximum value of any item contained in it. Intuitively, this corresponds to strictly *pessimistic* or *optimistic* customers who will always assume the worst-case or best-case allocation mechanism relative to their personal valuations, respectively. Before we give an overview of the results presented in this paper, let us introduce the problem more formally.

### 1.1 Preliminaries

The standard *unit-demand pricing problem* (UDP) is defined as follows. Given a set of items $\mathcal{I}$, each available in unlimited supply, and a collection of customers $\mathcal{C}$, each described by a *valuation function* $v_c : \mathcal{I} \to \mathbb{R}_0^+$, we want to assign prices to the items such as to maximize the overall revenue. More precisely, we assume that given prices $p(i)$ for all $i \in \mathcal{I}$, a customer $c \in \mathcal{C}$ will choose to purchase item

$$i_c(p) = \text{argmax}_{i \in \mathcal{I}}(v_c(i) - p(i)),$$

whenever that item's price does not exceed her respective value. To avoid technicalities, we assume that there is a special item $\varnothing$ with $v_c(\varnothing) = 0$, which is always assigned price 0. The quantity $v_c(i) - p(i)$ is termed customer $c$'s *utility* from purchasing item $i$ at price $p(i)$. We will also assume that whenever there are multiple items yielding identical utility, a customer will pick the one with highest price among them.[1] In this way, item $i_c(p)$ is well defined for any set of prices $p$. The revenue of a price assignment $p$ is

$$\text{rev}(p) = \sum_{i \in \mathcal{I}} p(i_c(p)).$$

In the *unit-demand bundle-pricing problem* (UDBP) considered in this paper we are again given a ground set $\mathcal{I}$ of items and a collection $\mathcal{C}$ of unit-demand customers. The output is a collection $\mathcal{B} \subseteq 2^{\mathcal{I}}$ of *bundles* of items and prices $p(B)$ for all bundles $B \in \mathcal{B}$. If a customer decides to purchase a bundle $B$ of items, she is guaranteed to receive an item from $B$. However, a customer does not have any information regarding the details of how the particular item she will receive is selected once the bundle is bought. Consequently, a customer's value for any given bundle has to depend on her belief about the selection procedure.

Most of this paper will be focused on the case of pessimistic customers who will assign to each bundle its worst-case value (UDBP-MIN). Formally, a customer with (unit-demand) valuation function $v_c : \mathcal{I} \to \mathbb{R}_0^+$ will value bundle $B \subseteq \mathcal{I}$ at

$$\bar{v}_c(B) = \min_{i \in B} v_c(i).$$

[1] This assumption is w.l.o.g., since in case of a tie decreasing all prices by a factor of $(1 - \varepsilon)$ for an arbitrary value of $\varepsilon$ ensures that for each customer the utility-maximizing item is one of maximal price.

As in standard unit-demand pricing, given prices $p$ each customer will purchase her utility-maximizing bundle

$$B_c(p) = \text{argmax}_{B \in \mathcal{B}}(\bar{v}_c(B) - p(B)),$$

where we assume that the empty bundle $\emptyset \in \mathcal{B}$ has price $p(\emptyset) = 0$ and is valued at 0 by all customers. Furthermore, ties are again broken in favor of more expensive bundles.

There are of course numerous other ways of extending unit-demand valuation functions to the set of all bundles. In this paper, we will also briefly look at the complementing case of customers assigning each bundle its best-case value (UDBP-MAX), formally,

$$\bar{v}_c(B) = \max_{i \in B} v_c(i).$$

Other models, in particular those assuming some kind of probabilistic selection method, are beyond the scope of this paper, but might also be of much interest, particularly as some of them are essentially variations of the *lottery* concept investigated in [5] and might have applications in the design of truthful revenue-maximizing auction mechanisms [3,12,15].

By *uniform* UDBP we refer to the restricted problem version in which customers' valuation functions assign identical (positive) values to some subset of the items and value 0 to all items in the complement of this subset. Formally, every customer is characterized by the set $S_c \subseteq \mathcal{I}$ of items she desires and her value $v_c \in \mathbb{R}^+$ for receiving any such item.

### 1.2 Contributions

We will first consider UDBP-MIN and present a number of algorithmic and complementing hardness results. In Section 2 we present a polynomial time algorithm for uniform UDBP-MIN, which is essentially based on two main ingredients. First, we observe that the number of bundles that might be part of an optimal bundle-pricing is small and, in fact, we can derive the set of bundles we need to consider immediately from the given set of customers. We then show that the extension of the valuation functions to this collection of bundles is very nicely structured, as a consequence of which one can apply techniques from [8] to solve the problem. More precisely, if we define a relation between bundles depending on whether there exists a customer who strictly prefers one of them to the other, this relation turns out to be transitive, as a consequence of which we can reduce the bundle-pricing problem to solving a weighted independent set problem in a perfect graph.

We proceed by considering the general (i.e., non-uniform) case of UDBP-MIN which turns out to be significantly more complex. In Section 3 we show that general UDBP-MIN is APX-hard. This is true even if all customers have non-zero values for at most 2 items and there are only 3 distinct values among all of them. The main distinction of our reduction from previous hardness results for unit-demand pricing problems stems from the fact that because of the enlarged

solution space (containing all possible bundles of items) we need to argue about a significantly larger set of potential solutions to prove that the reduction is indeed approximation preserving.

On the algorithmic side, we introduce the concept of $\alpha$-*coarse* instances, in which any two values of a single customer must be identical or differ by a factor of at least $\alpha$. We present a polynomial time algorithm that obtains a constant approximation guarantee for any given constant value of $\alpha > 1$. This is an interesting distinction from the several related item pricing problems, where the known inapproximability results suggest that coarse instances in particular seem to form the hard core of the problem [4]. The algorithm is based on a novel reduction of the general to the uniform problem, in the process of which each general valuation function is simulated by a carefully tailored collection of uniform valuation functions yielding similar revenue under all relevant pricings. This reduction is also interesting in its own right, as it can be applied to other unit-demand pricing problems as well, yielding similar algorithmic results as long as the uniform problem version allows for a good approximation. In particular, we can obtain constant factor approximation algorithms for $\alpha$-coarse instances of UDP with *price-ladder constraint* [1], i.e., when the relative order of item prices is predetermined, as the uniform version of this problem is known to be solvable in polynomial time via dynamic programming. These results are found in Section 4.

Finally, we briefly turn to UDBP-MAX and show that this problem behaves fundamentally different from UDBP-MIN. In Section 5 we argue that the problem turns out to be equivalent to the pure item pricing problem and, thus, all results known for UDP carry over in this case.

## 2 A Polynomial-Time Algorithm for Uniform UDBP-MIN

The first main ingredient for our polynomial-time algorithm for uniform UDBP-MIN are the following observations regarding the structure of the optimal collection of bundles and their prices. Note, that Definition 1 and Proposition 2 also apply to non-uniform UDBP-MIN.

**Definition 1.** *For a customer $c$ with valuation function $v_c : \mathcal{I} \to \mathbb{R}_0^+$ we let*

$$L_c^v = \{i \in \mathcal{I} \mid v_c(i) \geq v\}.$$

*We say that $L_c^v$ is customer $c$'s* level-$v$ *set.*

**Proposition 2.** *Let $(\mathcal{I}, \mathcal{C})$ be an instance of* UDBP-MIN. *Then there exists a revenue-maximizing collection $\mathcal{B}$ of bundles with corresponding prices $p$, such that $\mathcal{B} \subseteq \{L_c^v \mid c \in \mathcal{C}, v \in \mathbb{R}_0^+\}$.*

**Proposition 3.** *Let $(\mathcal{I}, \mathcal{C})$ be an instance of uniform* UDBP-MIN *and $(\mathcal{B}, p)$ a revenue-maximizing solution. It holds w.l.o.g. that $\mathcal{B} \subseteq \{S_c \mid c \in \mathcal{C}\}$ and $p(i) \in \mathcal{P}$ for all $i \in \mathcal{I}$, where $\mathcal{P} = \{v_c \mid c \in \mathcal{C}\}$.*

The proofs of Propositions 2 and 3 are left for the full version of this paper. Proposition 3 states that in the uniform case of UDBP-MIN both the set of possible bundles and the set of possible prices that can appear as part of an optimal solution are quite managable. In particular, the problem of computing an optimal bundle pricing reduces to deciding which customers should purchase their respective bundles $S_c$ and at which price from $\mathcal{P}$.

Similar to the approach first introduced in [8], we will transform the problem of computing the optimal bundle pricing into a weighted independent set problem and argue that the resulting graph is perfect, which allows us to solve the independent set problem in polynomial time [13]. We use $(c, S_c, p)$ to denote the fact that customer $c$ purchases bundle $S_c$ at price $p$. We create a vertex with label $(c, S_c, p)$ and weight $p$ for every $c \in \mathcal{C}$ and $p \in \mathcal{P}$ with $p \leq v_c$. Then we create a directed edge from the vertex with label $(c, S_c, p)$ to the vertex with label $(d, S_d, q)$, if and only if $S_d \subseteq S_c$ and $q < p$. Let us refer to the resulting directed graph as $G$ and let $\tilde{G}$ refer to the same graph but with undirected edges.

**Lemma 4.** *Graph $\tilde{G}$ as constructed above is perfect.*

The proof of Lemma 4, which is an application of the strong perfect graph theorem [9] and essentially similar to the proof given in [8], is omitted due to space limitations. Lemma 4 immediately yields a polynomial time algorithm for uniform UDBP-MIN.

**Algorithm 1.** Poly-Time Algorithm for Uniform UDBP-MIN.

**(1)** Given instance $(\mathcal{I}, \mathcal{C})$, construct the perfect weighted graph $\tilde{G}$ containing a vertex with label $(c, S_c, p)$ and weight $p$ for all $c \in \mathcal{C}$, $p \in \mathcal{P}$ with $p \leq v_c$, and an edge between vertices with labels $(c, S_c, p)$ , $(d, S_d, q)$ iff either $S_c \subseteq S_d$ and $p < q$ or $S_d \subseteq S_c$ and $q < p$.

**(2)** Find a maximum weight independent set in $\tilde{G}$.

**(3)** For each vertex in the independent set, if it has label $(c, S_c, p)$, offer bundle $S_c$ at price $p$.

**Theorem 5.** *Algorithm 1 returns a revenue-maximizing bundle pricing in polynomial time.*

*Proof.* By Proposition 3 there always exists a revenue-maximizing bundle pricing in which every customer $c \in \mathcal{C}$ buys bundle $S_c$ or nothing at all and all prices are chosen from the set $\mathcal{P}$. Clearly, for any customer purchasing bundle $S_c$ at price $p$ it must be the case that no bundle $B \subseteq S_c$ is offered at a price below $p$, as buying this bundle would yield higher utility for customer $c$. Consequently, the bundle pricing corresponds to an independent set in $\tilde{G}$ of total weight equal to the revenue obtained by the bundle pricing. Similarly, setting prices according to an independent set in $\tilde{G}$ ensures that customer $c$ purchase bundle $S_c$ at price $p$ whenever the vertex with label $(c, S_c, p)$ is part of the independent set. Thus, we have a one-to-one correspondence between bundle pricings and weighted independent sets in $\tilde{G}$ and it follows that Algorithm 1 returns a revenue-maximizing pricing.

Polynomial running time follows from the observation that graph $\tilde{G}$ has at most a polynomial number $|\mathcal{C}| \cdot |\mathcal{P}|$ of vertices and the fact that it is perfect by Lemma 4, so finding a maximum weight independent set can be done in polynomial time. □

## 3 Hardness of Approximation of UdBp-Min

In this section we show that UdBp-Min is APX-hard. In particular, we show that there is no polynomial time approximation algorithm achieving a revenue of at least $(428/429 + \epsilon)$ times the revenue of the optimal bundle pricing, unless P=NP. This is even true if every customer has non-zero value for at most two items and there are only three different values among all of them.

**Theorem 6.** *It is NP-hard to approximate* UdBp-Min *within* $c = \frac{428}{429} + \epsilon$ *for any* $\epsilon > 0$.

*Proof.* Our reduction is from the unweighted Max Dicut problem. An instance of this problem is a directed graph $G = (V, E)$, and the goal is to find a partition of $V$ into $(S, V \setminus S)$, where $S \subseteq V$, such that the number of edges that cross this cut, i.e., edges $(u, v)$ such that $u \in S$ but $v \notin S$, is maximized. This problem is not $(12/13 + \epsilon)$-approximable, for any constant $\epsilon > 0$, unless P=NP [10].

Given an instance $G = (V, E)$ of the unweighted Max Dicut problem, we create an instance of UdBp-Min by introducing one item for each node in $V$ and 48 customers for each edge in $E$. For the edge $(u, v) \in E$, we introduce customers with value 0 for all items in $V \setminus \{u, v\}$ and with the following values for $u$ and $v$:

| number of customers $c$ | 9 | 3 | 3 | 15 | 1 | 3 | 6 | 2 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| $v_c(u)$ | 0 | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 4 |
| $v_c(v)$ | 1 | 2 | 4 | 0 | 4 | 0 | 1 | 4 | 0 |

Given these customers, the only bundles for which there exist customers with non-zero valuation are singleton bundles and bundles $\{u, v\}$ for edges $(u, v) \in E$. Hence, we can focus on setting prices for these bundles. Furthermore, any pricing can be transformed into a pricing using only the prices 1, 2, 3, and 4 and achieving at least the same revenue as follows: If we have a pricing with prices below 1, we can first increase all these prices to 1 without decreasing the revenue, then we can increase all prices strictly between 1 and 2 to 2 without decreasing the revenue and so on.

If we have already set prices for the singleton bundles $\{u\}$ and $\{v\}$ and there is an edge $(u, v) \in E$, then the (not necessarily unique) price for the bundle $\{u, v\}$ maximizing the revenue is determined. A case analysis yields the following table showing an optimal price for $\{u, v\}$ and the total revenue obtained from all customers belonging to edge $(u, v)$ for the different choices of $p(\{u\})$ and $p(\{v\})$:

| $p(\{u\})$ | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p(\{v\})$ | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| opt. price for $p(\{u, v\})$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| total revenue | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 54 | 42 | 42 | 42 | 42 | 48 | 48 | 48 | 48 |

The following information from the previous table is crucial:

- For any pricing with $p(\{u\}) \neq 3$ and $p(\{v\}) \neq 3$, there is a choice for $p(\{u,v\})$ for which all customers belonging to edge $(u,v)$ yield a total revenue of 48.
- For any pricing with $p(\{u\}) \neq 2$ or $p(\{v\}) \neq 4$, there is no choice for $p(\{u,v\})$ for which they yield a larger revenue than 48.
- If $p(\{u\}) = 2$ and $p(\{v\}) = 4$, then there is a choice for $p(\{u,v\})$ for which all customers belonging to edge $(u,v)$ yield a total revenue of 54.

Based on this information, we can relate the maximum directed cut in the graph $G$ and the revenue-maximizing pricing for the instance of UDBP-MIN that we have constructed. If the graph $G$ has a cut $(S, V \setminus S)$ crossed by $\ell$ edges, then there exists a pricing for the instance of UDBP-MIN with a revenue of $48 \cdot (|E| - \ell) + 54 \cdot \ell$. For this, we assign a price of 2 to every set $\{u\}$ with $u \in S$ and a price of 4 to every set $\{u\}$ with $u \notin S$. The prices for the sets $\{u,v\}$ for edges $(u,v) \in E$ are chosen according to the previous table.

If the optimal pricing of the instance of UDBP-MIN yields a revenue of $48 \cdot (|E| - \ell) + 54 \cdot \ell$ for some $\ell \in \mathbb{N}$, then there exists a cut $S$ in $G$ that is crossed by $\ell$ edges. To see this, we can assume w.l.o.g. that in the optimal pricing all singleton bundles have a price of either 2 or 4 because replacing every price of 1 or 3 by a price of 4 does not decrease the revenue. Then if $S$ consists of exactly those nodes whose corresponding singleton bundle has a price of 2, there are $\ell$ edges crossing the cut $(S, V \setminus S)$.

An optimal directed cut of any graph $G = (V, E)$ is crossed by at least $|E|/4$ edges. In order to see this, consider the undirected (multi)-graph $G'$ obtained from $G$ by removing the directions of the edges. In a maximum undirected cut $(S, V \setminus S)$ of $G'$ at least half of the edges have one endpoint in $S$ and one endpoint in $V \setminus S$. This means that at least a quarter of the edges go from $S$ to $V \setminus S$ or at least a quarter of the edges go from $V \setminus S$ to $S$.

Assume there was an algorithm achieving a $(\frac{428}{429} + \epsilon)$-approximation for UDBP-MIN. Let $\ell^*$ denote the maximum number of edges crossing any cut in graph $G$, then $48 \cdot (|E| - \ell^*) + 54 \cdot \ell^* = 48 \cdot |E| + 6 \cdot \ell^*$ is the revenue of the optimal pricing in the instance of UDBP-MIN described above. Hence, the algorithm computes a pricing with revenue $48 \cdot |E| + 6 \cdot \ell$ with

$$\frac{48 \cdot |E| + 6 \cdot \ell}{48 \cdot |E| + 6 \cdot \ell^*} \geq c = \frac{428}{429} + \epsilon.$$

From this, we derive

$$\ell \geq c \cdot \ell^* - 8 \cdot |E| \cdot (1-c) \geq c \cdot \ell^* - 32 \cdot \ell^* \cdot (1-c) = \ell^* \cdot (33c - 32) \geq \ell^* \cdot \left(\tfrac{12}{13} + \epsilon\right).$$

Hence, $\frac{\ell}{\ell^*} \geq \frac{12}{13} + \epsilon$, contradicting the hardness of the MAX DICUT problem. □

## 4 Approximation Algorithm for Non-Uniform UDBP-MIN

**Definition 7.** *For a customer $c$ let $\mathcal{V}_c = \{v \mid \exists i \in \mathcal{I} : v_c(i) = v\}$ denote the range of her valuation function. We say that a* UDBP-MIN *instance $(\mathcal{I}, \mathcal{C})$ is* $\alpha$-coarse *for some $\alpha > 1$, if for every $c \in \mathcal{C}$ and all $v, v' \in \mathcal{V}_c$ with $v \neq v'$ it holds that either $v \geq \alpha v'$ or $v' \geq \alpha v$.*

**Algorithm 2.** Approximation Algorithm for $\alpha$-coarse UDBP-MIN.

**(1)** Given an $\alpha$-coarse instance $(\mathcal{I}, \mathcal{C})$, construct a uniform instance $(\mathcal{I}, \mathcal{C}')$ as follows: For every $c \in \mathcal{C}$ and every $v \in \mathcal{V}_c$, add a customer $c(v)$ with value $v$ and set of desired items $S_{c(v)} = L_c^v$.

**(2)** Compute an optimal solution $(\mathcal{B}, p)$ on this uniform instance.

**(3)** Return $\big(\mathcal{B}, (1-\alpha^{-1})p\big)$.

**Theorem 8.** *Algorithm 2 achieves approximation guarantee* $(1/4)(1-\alpha^{-1})^2$ *on* $\alpha$*-coarse instances of* UDBP-MIN.

Theorem 8 above is an immediate consequence of the following two lemmas.

**Lemma 9.** *Let* $R^*$ *denote the optimal revenue obtainable on* $\alpha$*-coarse instance* $(\mathcal{I}, \mathcal{C})$ *and* $R'$ *the maximum revenue obtainable on the uniform instance* $(\mathcal{I}, \mathcal{C}')$ *constructed by the algorithm. It holds that* $R' \geq R^*$.

*Proof.* We have seen in Proposition 2 that the collection of bundles $\mathcal{B}^*$ sold in the revenue-maximizing solution of instance $(\mathcal{I}, \mathcal{C})$ consists only of level sets of the customers from $\mathcal{C}$. Now assume that we offer all bundles from $\mathcal{B}^*$ at the same prices to the uniform customers constructed by the algorithm. For each customer $c \in \mathcal{C}$ purchasing her level-$v$ set $L_c^v$ at price $p \leq v$, we have a uniform customer $c(v)$ with value $v$ for any item in $L_c^v$ by construction, both of which experience utility $v - p$ from buying bundle $L_c^v$. On the other hand, customer $c(v)$'s values for all items are no larger than those of customer $c$ and, consequently, she values no bundle higher than $c$. Since buying $L_c^v$ at price $p$ is the utility maximizing choice for $c$, so it is for $c(v)$ and it follows that we collect as much revenue from $c(v)$ in the uniform instance as we do from $c$ in the original instance. Summing over all $c \in \mathcal{C}$ yields the claim. □

**Lemma 10.** *Let* $(\mathcal{B}, p)$ *be an optimal solution to the uniform* UDBP-MIN *instance* $(\mathcal{I}, \mathcal{C}')$ *constructed by the algorithm resulting in revenue* $R'$. *Then solution* $(\mathcal{B}, (1-\alpha^{-1})p)$ *yields revenue at least* $(1/4)(1-\alpha^{-1})^2 R'$ *on the original* $\alpha$*-coarse instance* $(\mathcal{I}, \mathcal{C})$.

*Proof.* Let $(\mathcal{B}, p)$ be an optimal solution to the uniform UDBP-MIN instance $(\mathcal{I}, \mathcal{C}')$. By Proposition 3 we may w.l.o.g. assume that $\mathcal{B}$ is a subset of the desired sets of customers from $\mathcal{C}'$ and so it is also a subset of the level sets of the original non-uniform customers from $\mathcal{C}$. We can also assume w.l.o.g. that every customer who buys a set buys her desired set and no subset.

Let $\mathcal{C}'_+ = \{c(v) \in \mathcal{C}' \mid L_c^v \in \mathcal{B} \text{ and } v/2 \leq p(L_c^v) \leq v\}$ denote the set of customers who purchase their set of desired items at a price of at least half their value. Note, that it must be the case that customers in $\mathcal{C}'_+$ contribute total revenue of at least $R'/2$. It is easy to argue that if this was not the case, multiplying all prices by a factor of 2 would increase overall revenue, contradicting the optimality of $(\mathcal{B}, p)$.

Let us refer to the original set of non-uniform customers which have at least one corresponding customer in $\mathcal{C}'_+$ as $M$ and sort the customers in $\mathcal{C}'_+$ according

to the non-uniform customer they represent and their values. For a customer $c \in M$, we define $\ell_c$ to be the number of corresponding customers in $\mathcal{C}'_+$. Formally, let us denote

$$\mathcal{C}'_+ = \bigcup_{c\in M} \bigcup_{i=1}^{\ell_c} \left\{ c\big(v_i^c\big) \right\},$$

where $v_1^c > v_2^c > \cdots > v_{\ell_c}^c$ for all $c \in M$. Now let $\mathcal{C}'_* = \bigcup_{c\in M}\{c(v_1^c)\}$ be the thinned out version of $\mathcal{C}'_+$ which only contains the uniform customer with highest value for each original customer $c \in M$. Let $R'_*$ be the total revenue collected from customers in $\mathcal{C}'_*$. It holds that

$$\begin{aligned} R'_* &\geq \frac{1}{2}\sum_{c\in M} v_1^c = \frac{1}{2}\big(1-\alpha^{-1}\big)\sum_{c\in M}\left(\sum_{i=0}^{\infty}\alpha^{-i}\right)v_1^c \\ &= \frac{1}{2}\big(1-\alpha^{-1}\big)\sum_{c\in M}\sum_{i=0}^{\infty}(\alpha^{-i}v_1^c) \geq \frac{1}{2}\big(1-\alpha^{-1}\big)\sum_{c\in M}\sum_{i=1}^{\ell_c} v_i^c \geq \frac{1}{2}\big(1-\alpha^{-1}\big)\frac{1}{2}R', \end{aligned}$$

where we use the facts that $v_i^c \leq \alpha^{-i+1}v_1^c$ since instance $(\mathcal{I},\mathcal{C})$ is $\alpha$-coarse and customer $c(v_i^c)$ cannot contribute more than $v_i^c$ to the overall revenue of at least $R'/2$ collected from customers in $\mathcal{C}'_+$.

Finally, let us fix a single uniform customer $c(v) \in \mathcal{C}'_*$ purchasing bundle $L_c^v$ at price $p$. We observe that it must be the case that all bundles $B \subset L_c^v$ must have a price of at least $p$, as otherwise purchasing $L_c^v$ could not be $c(v)$'s utility maximizing choice. Now consider the non-uniform customer $c$ corresponding to $c(v)$ and the effect of reducing all prices by a factor of $(1-\alpha^{-1})$. Customer $c$ has utility

$$v - \big(1-\alpha^{-1}\big)p \geq v - \big(1-\alpha^{-1}\big)v = \alpha^{-1}v$$

from purchasing bundle $L_c^v$ at price $(1-\alpha^{-1})p$. Her value for any bundle containing items from outside $L_c^v$ is at most $\alpha^{-1}v$ by $\alpha$-coarseness, so none of these bundles can yield higher utility even at price 0. Bundles strictly contained in $L_c^v$ could potentially yield higher utility, but by our argument above the price of any such bundle is at least $(1-\alpha^{-1})p$ after decreasing prices and we conclude that customer $c$ contributes at least as much revenue as $c(v)$ under the decreased prices.

It follows that when offered bundles $\mathcal{B}$ at prices $(1-\alpha^{-1})p$, customers $\mathcal{C}$ generate overall revenue of at least $(1-\alpha^{-1})R'_* \geq (1/4)(1-\alpha^{-1})^2R'$, which completes the proof. □

Finally, we briefly mention that the reduction described above has interesting applications in other varaiants of unit-demand pricing, as well. By UDP-PL we refer to the item pricing problem as defined in Section 1.1 with an additional *price ladder constraint* [1] $\pi$, i.e., a predefined relative order of item prices $p_{\pi(1)} \leq \cdots \leq p_{\pi(n)}$. It is known that uniform UDP-PL can be solved in polynomial time via a dynamic programming approach and, by our reduction, we obtain a $(1/4)(1-\alpha^{-1})^2$-approximation for general $\alpha$-coarse UDP-PL. This stands in sharp contrast to UDP without price ladder contraint, which does not allow for constant approximation gurantees even on coarse instances [4].

## 5 Approximability of UdBp-Max

In this section we turn to UdBp-Max where customers are strictly optimistic and assign to every bundle the maximum value of any item contained in it. We will see that this model is fundamentally different from UdBp-Min as it turns out to be equivalent to the pure item pricing problem.

Let $(\mathcal{I}, \mathcal{C})$ be an instance of UdBp-Max and let $(\mathcal{B}, p)$ be an optimal solution to that instance. Then we can transform $(\mathcal{B}, p)$ into a solution $(\mathcal{B}', p')$ yielding the same revenue where $\mathcal{B}'$ consists of singleton sets only. For this, we just need to replace any non-singleton bundle in $\mathcal{B}$ that is bought by a subset $\{c_1, \ldots, c_\ell\} \subseteq \mathcal{C}$ of customers by a set of bundles $\{i_1\}, \ldots, \{i_\ell\}$ where $i_j$ denotes the item from $\mathcal{B}$ that customer $c_j$ values the most. All these new bundles are offered for price $p(\mathcal{B})$. For a customer $j$ bundle $\{i_j\}$ has the same value as bundle $\mathcal{B}$ and for every other customer it has at most the same value. Hence, customer $j$ will buy $\{i_j\}$ and the other customers in $\mathcal{C} \setminus \{c_1, \ldots, c_\ell\}$ are not affected by replacing $\mathcal{B}$.

This implies that UdBp-Max and the pure item pricing problem are essentially the same problem. Hence, known results for the latter problem apply also to UdBp-Max. In particular, the revenue-maximizing item pricing can be approximated within factors that are logarithmic in the number of customers or linear in the number of distinct items [14] and, under appropriate complexity theoretic assumptions, no essential improvement is possible [4,11].

## 6 Conclusions

We have introduced an extension of the unit-demand pricing problem in which bundles may be offered. This problem is interesting because it models the sales model of retailers like hotwire.com. We have seen that different assumptions about the customers' beliefs yield very different conclusions. While for the case of pessimistic customers we presented novel algorithmic results, the case of optimistic customers boils down to the pure item pricing problem.

There are many interesting questions open. One question arising directly from our results is whether there exists a constant factor approximation algorithm for general non-uniform instances of UdBp-Min without the additional assumption of $\alpha$-coarseness. It would also be very interesting to extend our model to different beliefs of customers. One could, e.g., study a model in which customers believe that an item is chosen uniformly at random from the set they buy.

## Acknowledgements

We thank Bobby Kleinberg for several insightful discussions.

## References

1. Aggarwal, G., Feder, T., Motwani, R., Zhu, A.: Algorithms for Multi-product Pricing. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 72–83. Springer, Heidelberg (2004)

2. Balcan, M.-F., Blum, A.: Approximation Algorithms and Online Mechanisms for Item Pricing. In: Proc. of the 7th ACM Conference on Electronic Commerce, EC (2006)
3. Balcan, M.-F., Blum, A., Hartline, J.D., Mansour, Y.: Mechanism Design via Machine Learning. In: Proc. of the 46th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 605–614 (2005)
4. Briest, P.: Uniform Budgets and the Envy-Free Pricing Problem. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 808–819. Springer, Heidelberg (2008)
5. Briest, P., Chawla, S., Kleinberg, R., Weinberg, M.: Pricing Randomized Allocations. In: Proc. of the 21st ACM-SIAM Symposium on Discrete Algorithms, SODA (2010)
6. Briest, P., Krysta, P.: Buying Cheap is Expensive: Hardness of Non- Parametric Multi-Product Pricing. In: Proc. of the 18th ACM-SIAM Symposium on Discrete Algorithms, SODA (2007)
7. Chawla, S., Hartline, J.D., Kleinberg, R.: Algorithmic Pricing via Virtual Valuations. In: Proc. of the 8th ACM Conference on Electronic Commerce (EC), pp. 243–251 (2007)
8. Chen, N., Ghosh, A., Vassilvitskii, S.: Optimal Envy-Free Pricing with Metric Substitutability. In: Proc. of the 9th ACM Conference on Electronic Commerce (EC), pp. 60–69 (2008)
9. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The Strong Perfect Graph Theorem. Annals of Mathematics 164, 51–229 (2006)
10. Crescenzi, P., Silvestri, R., Trevisan, L.: On Weighted vs Unweighted Versions of Combinatorial Optimization Problems. Inf. Comput. 167(1), 10–26 (2001)
11. Demaine, E., Feige, U., HajiAghayi, M.T., Salavatipour, M.: Combination Can Be Hard: Approximability of the Unique Coverage Problem. In: Proc. of the 17th ACM-SIAM Symposium on Discrete Algorithms, SODA (2006)
12. Goldberg, A.V., Hartline, J.D., Karlin, A.R., Saks, M., Wright, A.: Competitive Auctions. Games and Economic Behavior 55(2), 242–269 (2006)
13. Grötschel, M., Lovász, L., Schrijver, A.: The Ellipsoid Method and its Consequences in Combinatorial Optimization. Combinatorica 1(2), 169–197 (1981)
14. Guruswami, V., Hartline, J.D., Karlin, A.R., Kempe, D., Kenyon, C., McSherry, F.: On Profit-Maximizing Envy-Free Pricing. In: Proc. of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1164–1173 (2005)
15. Myerson, R.: Optimal Auction Design. Mathematics of Operations Research 6, 58–73 (1981)
16. Riley, J., Zeckhauser, R.: Optimal Selling Strategies: When to Haggle, When to Hold Firm. Quarterly J. Economics 98(2), 267–289 (1983)
17. Thanassoulis, J.: Haggling Over Substitutes. J. Economic Theory 117, 217–245 (2004)

# Tradeoff between Energy and Throughput for Online Deadline Scheduling

Ho-Leung Chan*, Tak-Wah Lam, and Rongbin Li

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

**Abstract.** We consider dynamic speed scaling on a single processor and study the tradeoff between throughput and energy for deadline scheduling. Specifically, we assume each job is associated with a user-defined value (or importance) and a deadline. We allow scheduling algorithms to discard some of the jobs (i.e., not finishing them) and the objective is to minimize the total energy usage plus the total value of jobs discarded. We give new online algorithms under both the unbounded-speed and bounded-speed models. When the maximum speed is unbounded, we give an $O(1)$-competitive algorithm. This algorithm relies on a key notion called the profitable speed, which is the maximum speed beyond which processing a job costs more energy than the value of the job. When the processor has a bounded maximum speed $T$, we show that no $O(1)$-competitive algorithm exists and more precisely, the competitive ratio grows with the penalty ratio of the input, which is defined as the ratio between the maximum profitable speed of a job to the maximum speed $T$. On the positive side, we give an algorithm with a competitive ratio whose dependency on the penalty ratio almost matches the lower bound.

## 1 Introduction

Energy efficiency is a major concern not only for mobile devices, but also for large-scale server farms like those operated by Google [13]. Recently, it has been reported that the average energy cost for running a server exceeds the purchase cost of the server [9]. To improve energy efficiency, major chip manufacturers like Intel and AMD now produce processors equipped with a technology called *dynamic voltage scaling*. Specifically, it allows operating systems or application software to dynamically vary the processor speed so as to manage the energy usage. Running at a low speed reduces energy usage drastically, yet we still want to maintain some kind of quality of service (QoS). These conflicting objectives have imposed new challenges to the research on scheduling. In this paper, the QoS concerned is the throughput, i.e., total size or value of jobs completed by their deadlines.

**The history.** The theoretical study of energy-efficient online scheduling was initiated by Yao, Demers and Shenker [15]. They considered online deadline scheduling on a processor that can vary its speed dynamically between $[0, \infty)$. When the processor runs at speed $s$, the rate of energy usage, denoted by $P(s)$,

* Ho-Leung Chan is partially supported by HKU Seed Funding 201002159001.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 59–70, 2011.

is modeled as $s^\alpha$, where $\alpha > 1$ is a constant commonly believed to be 2 or 3 (determined by the physical properties of the hardware technology). Jobs with different sizes and deadlines arrive online over time. Jobs are preemptive and a preempted job can be resumed later at the point of preemption. The objective is to minimize the total energy usage subject to completing all jobs by their deadlines. [15] proposed two online algorithms AVR and OA, and showed that AVR is $(2^{\alpha-1}\alpha^\alpha)$-competitive. After about a decade, Bansal, Kimbrel and Pruhs [7] showed that OA is indeed better and is $\alpha^\alpha$-competitive. They also gave another algorithm BKP which is $O(e^\alpha)$-competitive (i.e., better than OA when $\alpha$ is large). Recently, Bansal et al. [6] showed that no algorithm can have a competitive ratio better than $e^{\alpha-1}/\alpha$, and they also gave an algorithm qOA that is $4^\alpha/(2\sqrt{e\alpha})$-competitive. When $\alpha = 3$, the competitive ratio of qOA can be fine tuned to 6.7.

All the above work assumes that the processor has unbounded maximum speed and can always complete every job on time. Chan et al. [10] extended the study of energy-efficient scheduling to a more realistic setting where a processor can only vary its speed between 0 to some fixed maximum speed $T$. Since the maximum speed is bounded, it is possible that no algorithm can complete all the given jobs. It is natural to consider the case where the optimal algorithm maximizes the throughput (which is the total size of jobs completed by their deadlines), and minimizes the energy usage subject to this maximum throughput. They gave an online algorithm that is 14-competitive on throughput and $(\alpha^\alpha + 4^\alpha\alpha^2)$-competitive on energy. Later, Bansal et al. [4] gave an improved algorithm that is 4-competitive on throughput, while the competitive ratio on energy remains the same. This algorithm is optimal in terms of throughput since any algorithm is at least 4-competitive on throughput even if we ignore the energy concern [11].

**Tradeoff between energy and throughput.** Note that all the above studies assume throughput is the primary concern. That is, the objectives require a scheduling algorithm to first maximize the throughput and then minimize the energy usage subject to the maximum throughput. With the growing importance of energy saving, this assumption may not be valid and some systems may actually prefer to trade throughput for better energy efficiency. For example, imagine the following scenario. There is a web server whose users are divided into different levels of importance. During the peak period, it may be desirable to drop the requests from less important users if the extra energy used for speeding up the processor to serve these requests costs more than the revenue generated by these requests. Note that when the server load is low, requests from less important users could be served at a low speed. The energy usage is much smaller and could make these jobs profitable.

**Our results.** To cater for the above situations, we initiate studying the tradeoff between throughput and energy. Specifically, we assume that each job is associated with a deadline and a user-defined *value*, the latter is about the importance of the job (e.g., the value can be the job size or simply any fixed constant). A scheduling algorithm may choose to finish only a subset of the given jobs by their deadlines and discard the rest. The objective is to minimize the total energy usage plus the total value of jobs discarded. The objective of minimizing the

total energy usage and value discarded has the following interpretation. From an economic point of view, a user would estimate the cost for one unit of energy and the revenue generated for each job. By normalizing the cost for one unit of energy to be one and assigning the normalized revenue for each job as its value, minimizing the total energy usage plus value discarded is equivalent to maximizing the total profit of the system.

We first study the tradeoff in the unbounded speed model. Notice that the problem of minimizing the total energy usage plus value discarded is a generalization of the classical problem of minimizing the total energy usage for completing all jobs, thus inheriting any lower bound result from the latter. The argument is as follows. Consider a set of jobs whose values are set to be sufficiently large, then the optimal offline algorithm and any competitive online algorithm will not discard any jobs, and the problem of minimizing energy plus value discarded is reduced to the problem of minimizing the energy usage subject to completing all jobs. Furthermore, since the value discarded is zero in this case, any $c$-competitive algorithm for the new objective gives a $c$-competitive algorithm for the classical objective. Recall that for the classical objective, no online algorithm has a competitive ratio better than $e^{\alpha-1}/\alpha$ [6]. This lower bound is also valid for the new objective of minimizing energy plus value discarded.

On the positive side, when the maximum speed is unbounded, we give an $O(1)$-competitive algorithm called PS. Precisely, the competitive ratio of PS is $\alpha^\alpha + 2e\alpha$. The main idea is about a notion called *profitable speed* for each job, which is the maximum speed beyond which processing the job costs more energy than the value of the job. Roughly speaking, the algorithm works as follows. When a job is released, PS calculates the OA schedule for all admitted jobs together with the new job. The new job is admitted if the OA schedule processes the new job with a speed at most $c$ times the profitable speed, where $c$ is a carefully chosen constant; otherwise the new job is discarded immediately. Though PS might look simple, the analysis is non-trivial. We first upper bound the value discarded by PS in terms of the energy used by PS plus the energy usage and value discarded of the optimal schedule. Then we bound the energy usage of PS using a potential function analysis.

For the bounded speed model, we show that the new objective becomes more difficult by giving a non-constant lower bound on the competitive ratio of any online algorithm. In particular, we define the *penalty ratio* of an input instance as the ratio of the maximum profitable speed of a job to the maximum processor speed $T$. We show that the competitive ratio of any algorithm is $\Omega(\max\{e^{\alpha-1}/\alpha, \Gamma^{\alpha-2+1/\alpha}\})$, where $\Gamma$ is the penalty ratio. The lower bound holds even if all jobs have the value equal to the size. On the other hand, we adapt the algorithm PS to the bounded-speed setting and show that its competitive ratio is $\alpha^\alpha + 2\Gamma^{\alpha-1}(\alpha+1)^{\alpha-1}$. Note that the dependency on the penalty ratio almost mathes the lower bound.

**Remark on an alternative objective.** Another and perhaps a more natural approach for studying the tradeoff between throughput and energy is to consider the objective of maximizing the total value of jobs completed by their deadlines

minus the total energy usage. However, we first notice that this objective, unlike the one for minimizing the total energy usage plus value discarded, is no longer a generalization of the classical model of minimizing total energy subject to completing all jobs. That is, a $c$-competitive algorithm for this maximization objective no longer gives a $c$-competitive algorithm for the classical model. More importantly, even in the unbounded-speed setting with the restriction that job value equals job size, this maximization objective is intractable as we can easily construct an instance where any online algorithm has total throughput minus energy arbitrarily close to zero or even zero, while an offline algorithm can obtain at least a finite throughput minus energy. We consider optimizing the total energy plus value discarded to avoid this singularity issue of getting a zero or close to zero value in the objective function. Recently and independently, Pruhs and Stein [14] studied the maximization objective. They consider the resource augmentation model where the online algorithm is given a processor that can run faster than that of the optimal with the same rate of energy usage; and they show that an $O(1)$-competitive algorithm exists.

**Other related work.** Energy efficiency has attracted a lot attention from the scheduling community in the past few years, see, e.g., [1] for a survey. Besides the related work already mentioned, there is another well-studied problem with similar flavor as ours, which is about energy-efficient flow time scheduling. In that problem, jobs with arbitrary sizes, but with no deadlines, arrive over time. The flow time of a job is the length of the duration from its arrival until it is completed. The objective is to complete all jobs and to minimize the total energy usage plus the total flow time of the jobs. The objective defined in this paper is motivated in part by this energy-plus-flow-time objective. Albers and Fujiwara [2] were the first to study this energy plus flow time objective. Following a chain of works [8, 12, 5], Andrew et al. [3] have finally given a 2-competitive algorithm for minimizing energy plus flow time.

## 2 Preliminaries

We first define the problem formally and review the algorithm OA [15, 7].

**Problem definition.** We consider online scheduling of jobs on a single processor. Each job $j$ has a release time $r(j)$, size $p(j)$, deadline $d(j)$ and a value $v(j)$. Let $J$ and $v(J)$ denote a sequence of jobs and their total values. Preemption is allowed. The processor can run at any speed in $[0, \infty)$ in the unbounded speed model and can run at speed in $[0, T]$ in the bounded speed model, where $T$ is a fixed constant. In any case, the rate of energy usage of the processor is $s^\alpha$, where $s$ is the running speed and $\alpha > 1$ is a constant. Let $s(t)$ be the speed of the processor at time $t$. Then the total energy usage is $\int_0^\infty (s(t))^\alpha dt$. Let $s(j,t)$ denote the speed at which a job $j$ is being processed at time $t$. The algorithms in this paper do not use time sharing; yet, if time sharing is allowed, we require that $\sum_j s(j,t) \leq s(t)$ for all $t$. A job $j$ is completed by $d(j)$ if $\int_{r(j)}^{d(j)} s(j,t)dt \geq p(j)$; and $j$ is discarded otherwise. The objective is to minimize the total energy usage plus the total value of jobs discarded. We denote Opt as the optimal offline

schedule which minimizes the objective for any input $J$. An algorithm is said to be $\gamma$-competitive if for any input $J$, the total energy usage plus the total value discarded is at most $\gamma$ times that of Opt.

**Review of algorithm OA.** At any time $t$, OA defines a sequence of times $t_0, t_1, \ldots$ as follows. Let $S$ be the jobs remaining at time $t$. Let $t_0 = t$. For $i = 1, 2, \ldots,$, let $t_i$ be the latest time after $t_{i-1}$ such that $\frac{w(t_{i-1},t_i)}{t_i - t_{i-1}}$ is maximized, where $w(t_{i-1}, t_i)$ is the total remaining size for jobs in $S$ with deadline in $(t_{i-1}, t_i]$. The interval $I_i = (t_{i-1}, t_i]$ is called the *i-th critical interval*, and the quantity $\rho_i = \frac{w(t_{i-1},t_i)}{t_i - t_{i-1}}$ is called the *density* of $I_i$. OA processes the jobs by EDF (earliest deadline first) and the speed during each critical interval $(t_{i-1}, t_i]$ is $\rho_i$. Note that $\rho_i$ is decreasing. It can been shown that OA uses the minimum energy to complete $S$ if no new jobs arrive. If a new job $j$ arrives after time $t$, the OA schedule will be recomputed starting from the time $r(j)$. Below are some known properties about OA which will be used by our algorithms.

*Property 1.* Consider an OA schedule and assume a job $j$ arrives at time $r(j)$. Let $S$ be the jobs remaining just before $j$ arrives and let $OA(S)$ be the OA schedule just before $j$ arrives. Let $OA(S \cup \{j\})$ be the re-calculated OA schedule just after $j$ arrives. Then,

(i) In $OA(S \cup \{j\})$, $j$ is processed by a constant speed $s(j)$. Furthermore, the speed of $OA(S \cup \{j\})$ during the period $[r(j), d(j)]$ is at least $s(j)$.
(ii) Let $I$ be any set of disjoint intervals after time $r(j)$. The total amount of work scheduled in $I$ by $OA(S \cup \{j\})$ is at least the total amount of work scheduled in $I$ by $OA(S)$, but at most the total amount of work scheduled in $I$ by $OA(S)$ plus $p(j)$.

## 3 Unbounded Speed Model

This section considers the unbounded speed model where the processor can run at any speed in $[0, \infty)$. We present an algorithm PS($c$), which stands for Profitable Speed with parameter $c$, and show that it is $(\alpha^\alpha + 2e\alpha)$-competitive when setting $c = \alpha^{(\alpha-2)/(\alpha-1)}$. First, we define the notion of *profitable speed*. For any job $j$, let $u(j) = v(j)/p(j)$ be the *value density* of $j$.

**Definition 1.** *The* profitable speed *of Job $j$, denoted $\tilde{s}(j)$, equals* $(u(j))^{1/(\alpha-1)}$.

**Fact 1.** *If we complete a job $j$ at a constant speed equal to $\tilde{s}(j)$, then the energy usage on processing $j$ equals the value of $j$.*

*Proof.* The energy usage is $(\tilde{s}(j))^\alpha \frac{p(j)}{\tilde{s}(j)} = (\tilde{s}(j))^{\alpha-1} p(j) = u(j)p(j) = v(j)$. □

Intuitively, the profitable speed $\tilde{s}(j)$ is a "boundary speed" suggesting whether we should complete or discard $j$. If the speed needed to complete $j$ is larger than $\tilde{s}(j)$, the energy usage on processing $j$ will be larger than its value and discarding it (instead of completing it) is more beneficial. On the other hand, if the speed needed is smaller than $\tilde{s}(j)$, completing $j$ is "profitable". Roughly speaking, our algorithm completes $j$ only when it can be completed at speed at most $c \cdot \tilde{s}(j)$.

### 3.1 Algorithm PS($c$)

Algorithm PS($c$) ($c$ is a parameter) maintains a list $Q$ of admitted jobs, which is empty initially. When a job arrives, it is immediately admitted into $Q$ or discarded. PS($c$) only processes and completes jobs in $Q$. Details are below.

**Algorithm PS**($c$)
- **Job execution.** At any time, PS($c$) uses OA to schedule the jobs in $Q$. (Note that in the literature, the OA schedule is defined and analyzed based on the entire input rather than a subset.)
- **Job admission.** When a job $j$ arrives at time $r(j)$, let $S$ be the set of jobs remaining in $Q$ just before $j$ arrives. PS($c$) calculates the OA schedule for $S \cup \{j\}$. Let $s(j)$ be the speed of $j$ in this OA schedule. PS($c$) admits $j$ into $Q$ if $s(j) \leq c \cdot \tilde{s}(j)$; and $j$ is discarded immediately otherwise.
- **Job Completion.** When a job is completed, remove it from $Q$.

By definition, OA always completes the jobs given to it no later than their respective deadlines. Thus, PS($c$) also meets the deadline of every job in $Q$. The main result of this section is about the competitiveness of PS($c$) on minimizing energy plus value discarded.

**Theorem 1.** *$\forall c>0$, PS($c$) is $\left((1+\frac{b^{\alpha-1}}{(cb-1)^{\alpha}})\max\{\alpha^{\alpha}, \alpha^2 c^{\alpha-1}\}+\max\{b^{\alpha-1},1\}\right)$-competitive on energy plus value discarded, for any $b > \frac{1}{c}$.*

By choosing the parameter $c$ to be $\alpha^{\frac{\alpha-2}{\alpha-1}}$ and considering $b = \frac{\alpha+1}{c} = \frac{\alpha+1}{\alpha^{(\alpha-2)/(\alpha-1)}}$, the competitive ratio becomes $\alpha^{\alpha} + 2\alpha(1+\frac{1}{\alpha})^{\alpha-1}$. Since $(1+\frac{1}{\alpha})^{\alpha-1} < e$, we conclude that PS($\alpha^{\frac{\alpha-2}{\alpha-1}}$) is $(\alpha^{\alpha} + 2e\alpha)$ -competitive.

To prove Theorem 1, we analyze the energy and the value discarded separately. Consider any input job sequence $J$ and parameter $c > 0$. Let $E_a$ and $E_o$ be the total energy usage of PS($c$) and Opt, respectively. Similarly, let $D_a$ and $D_o$ be the value discarded by PS($c$) and Opt, respectively. We will prove the following two lemmas concerning the value discarded and energy usage of PS($c$), whose proofs are given in the following subsections.

**Lemma 1.** *$D_a \leq \frac{b^{\alpha-1}}{(cb-1)^{\alpha}} E_a + b^{\alpha-1} E_o + D_o$, for any $b > \frac{1}{c}$.*

**Lemma 2.** *$E_a \leq \max\{\alpha^{\alpha}, \alpha^2 c^{\alpha-1}\}(E_o + D_o)$.*

Lemmas 1 and 2 together imply Theorem 1 as follows. For Opt, the total energy usage plus value discarded is $E_o + D_o$. Therefore, the total energy usage plus discard of PS($c$) is

$$\begin{aligned} E_a + D_a &\leq \left(1 + \tfrac{b^{\alpha-1}}{(cb-1)^{\alpha}}\right) E_a + b^{\alpha-1} E_o + D_o \\ &\leq \left(1 + \tfrac{b^{\alpha-1}}{(cb-1)^{\alpha}}\right) \max\{\alpha^{\alpha}, \alpha^2 c^{\alpha-1}\}\Big(E_o + D_o\Big) + b^{\alpha-1} E_o + D_o \end{aligned}$$

and Theorem 1 follows.

### 3.2 Value Discarded by PS($c$)

This section analyzes $D_a$ and proves Lemma 1. Let $J_D \subseteq J$ be the subset of jobs discarded by PS($c$). We further divide $J_D$ into $J_{D1}$ and $J_{D2}$, which include the jobs that are completed and discarded by Opt, respectively. $D_a = v(J_{D1}) + v(J_{D2}) \leq v(J_{D1}) + D_o$. To prove Lemma 1, it is sufficient to show that $v(J_{D1}) \leq \frac{b^{\alpha-1}}{(cb-1)^\alpha} E_a + b^{\alpha-1} E_o$.

Let $j$ be an arbitrary job in $J_{D1}$. Let $I(j)$ be the set of maximal time intervals during which Opt processes $j$. Denote $|I(j)|$ as the total length of the intervals in $I(j)$. Denote $E_a(I(j))$ and $E_o(I(j))$ as the energy usage by PS($c$) and Opt during $I(j)$, respectively. We will bound $v(j)$ by $E_a(I(j))$ and $E_o(I(j))$. Intuitively, if $|I(j)|$ is small, Opt completes $p(j)$ units of work in a short period of time and $E_o(I(j))$ should be relatively large. On the other hand, if $|I(j)|$ is large, then $E_a(I(j))$ is relatively large since PS($c$) discards $j$ and PS($c$) must run at relatively high speed during $I(j)$. Details are as follows.

**Lemma 3.** *Let $j$ be any job in $J_{D1}$, then $v(j) \leq \frac{b^{\alpha-1}}{(cb-1)^\alpha} E_a(I(j)) + b^{\alpha-1} E_o(I(j))$ for any $b > \frac{1}{c}$.*

*Proof.* To ease the discussion, let us denote $\tilde{\ell}(j)$ as the time to complete $j$ if at speed $\tilde{s}(j)$, i.e., $\tilde{\ell}(j) = p(j)/\tilde{s}(j)$. Note that $p(j) = \tilde{s}(j) \cdot \tilde{\ell}(j)$. Let $b_j = |I(j)|/\tilde{\ell}(j)$.

Note that Opt completes exactly $p(j)$ units of work in $I(j)$ and Opt runs at the speed $p(j)/|I(j)|$ throughout $I(j)$. Therefore,

$$E_o(I(j)) = \left(\frac{p(j)}{|I(j)|}\right)^\alpha |I(j)| = \left(\frac{\tilde{\ell}(j) \cdot \tilde{s}(j)}{|I(j)|}\right)^{\alpha-1} p(j) = \frac{u(j)}{b_j^{\alpha-1}} \cdot p(j) = \frac{v(j)}{b_j^{\alpha-1}} \quad (1)$$

where the last equality comes from the definition that $u(j) = v(j)/p(j)$.

Since $j$ is discarded by PS($c$), consider the time $r(j)$ when $j$ arrives. Let $S$ be the set of jobs remaining in $Q$ just before $j$ arrives. Let OA($S$) and OA($S \cup \{j\}$) be the OA schedules starting from time $r(j)$ for $S$ and $S \cup \{j\}$, respectively, assuming no other jobs arrive. Since $j$ is discarded, the speed of $j$ in OA($S \cup \{j\}$) is at least $c \cdot \tilde{s}(j)$. Since all intervals in $I(j)$ are completely inside $[r(j), d(j)]$, by Property 1 (i), the speed of OA($S \cup \{j\}$) throughout these intervals is at least $c \cdot \tilde{s}(j)$. Hence, the total work done by OA($S \cup \{j\}$) during $I(j)$ is at least $c \cdot \tilde{s}(j) \cdot |I(j)|$. By Property 1 (ii), the work done by OA($S$) in the intervals in $I(j)$ is at least $c \cdot \tilde{s}(j) \cdot |I(j)| - p(j)$. Again by Property 1 (ii), if some more jobs arrive after $j$, the amount of work scheduled to the intervals in $I(j)$ may only increases. Therefore,

$$\begin{aligned} E_a(I(j)) &\geq \left(\frac{c \cdot \tilde{s}(j) \cdot |I(j)| - p(j)}{|I(j)|}\right)^\alpha |I(j)| \\ &= \left(\frac{c \cdot \tilde{s}(j) \cdot b_j \tilde{\ell}(j) - \tilde{s}(j) \cdot \tilde{\ell}(j)}{b_j \tilde{\ell}(j)}\right)^\alpha b_j \cdot \tilde{\ell}(j) = \frac{(c \cdot b_j - 1)^\alpha}{b_j^\alpha} \cdot (\tilde{s}(j))^\alpha \cdot b_j \tilde{\ell}(j) \\ &= \frac{(cb_j - 1)^\alpha}{b_j^{\alpha-1}} u(j) \tilde{s}(j) \tilde{\ell}(j) = \frac{(cb_j - 1)^\alpha}{b_j^{\alpha-1}} v(j) \end{aligned} \quad (2)$$

Finally, for $b > 1/c$, there are two cases. If $b > b_j$, by (1), $v(j) = b_j^{\alpha-1} E_o(I(j)) < b^{\alpha-1} E_o(I(j))$. Otherwise, $b \le b_j$, then by (2), $v(j) \le \frac{b_j^{\alpha-1}}{(cb_j-1)^\alpha} E_a(I(j)) \le \frac{b^{\alpha-1}}{(cb-1)^\alpha} E_a(I(j))$, where the last inequality comes from the fact that function $f(x) = \frac{x^{\alpha-1}}{(cx-1)^\alpha}$ decreases when $x > \frac{1}{c}$. Hence for all $b > \frac{1}{c}$, Lemma 3 holds. □

Next, note that for any two jobs $j$ and $j'$ in $J_{D1}$, $I(j)$ and $I(j')$ are disjoint. Hence, by summing up the inequality in Lemma 3 over all jobs in $J_{D1}$, we obtain $v(J_{D1}) \le \frac{b^{\alpha-1}}{(cb-1)^\alpha} E_a + b^{\alpha-1} E_o$. Hence, Lemma 1 follows immediately.

### 3.3 Energy Usage of PS($c$)

This section analyzes $E_a$ and proves Lemma 2. We will use a potential function, which is similar to the one used in analyzing OA [7]. However, a major difference in our problem is that both PS($c$) and Opt may discard jobs, so the set of jobs scheduled by the two algorithms can be different. In particular, when a job $j$ is admitted by PS($c$) but discarded by Opt, our analysis needs to relate the extra energy usage of PS($c$) on processing $j$ to the value of $j$ discarded by Opt. Intuitively, this extra energy can be bounded because PS($c$) admits $j$ only if its speed is at most $c$ times the profitable speed. Details are as follows.

W.L.O.G., we assume that Opt admits a job $j$ at $r(j)$ if Opt will complete $j$; otherwise, Opt discards $j$ immediately. Let $E_a(t)$ and $E_o(t)$ be the energy usage of PS($c$) and Opt, respectively, by time $t$. Let $D_o(t)$ be the total value of jobs discarded by Opt by time $t$. Let $s_a(t)$ and $s_o(t)$ be the speed of PS($c$) and Opt, respectively, at time $t$. We will define a potential function $\Phi(t)$ satisfying the following conditions.

- *Boundary condition:* $\Phi(t) = 0$ before any job arrival and after all deadlines.
- *Running condition:* At any time $t$ without job arrival, $\frac{d}{dt}E_a(t) + \frac{d}{dt}\Phi(t) \le \max\{\alpha^\alpha, \alpha^2 c^{\alpha-1}\} \frac{d}{dt}(E_o(t) + D_o(t))$.
- *Arrival condition:* When a job $j$ arrives at time $t$, let $\Delta\Phi(t)$ and $\Delta D_o(t)$ denote the change of $\Phi(t)$ and $D_o(t)$, respectively, due to the arrival of $j$. Then $\Delta\Phi(t) \le \max\{\alpha^\alpha, \alpha^2 c^{\alpha-1}\}\Delta D_o(t)$.

Similar to [7, 10], we can then prove by induction on time that

$$\forall t, \qquad E_a(t) + \Phi(t) \le \max\{\alpha^\alpha, \alpha^2 c^{\alpha-1}\}(E_o(t) + D_o(t))$$

which implies Lemma 2 as $\Phi(t) = 0$ after all deadlines are passed.

**Definition of the potential function.** Consider any time $t$. For any $t'' \ge t' \ge t$, let $w_a(t', t'')$ be the total remaining size for jobs in the admitted list $Q$ of PS($c$) with deadlines in $(t', t'']$. Recall that PS($c$) processes $Q$ by OA, which defines a sequence of times $t_0, t_1, t_2, \ldots$, where $t_0 = t$ and for $i = 1, 2, \ldots$, $t_i$ is the latest time after $t_{i-1}$ such that $\rho_i = \frac{w_a(t_{i-1}, t_i)}{t_i - t_{i-1}}$ is maximized. We call $I_i = (t_{i-1}, t_i]$ as the $i$-th critical interval. On the other hand, consider the schedule of

Opt, and let $w_o(t', t'')$ be the total remaining size for jobs admitted by Opt by time $t$ with deadlines in $(t', t'']$. The potential function $\Phi(t)$ is defined as

$$\Phi(t) = \alpha \sum_{i \geq 1} \rho_i^{\alpha-1}(w_a(t_{i-1}, t_i) - \alpha w_o(t_{i-1}, t_i)) \tag{3}$$

It is easy to see that $\Phi(t)$ satisfies the boundary condition. We prove that it satisfies the arrival and running conditions as follows. Unlike the previous potential analysis [7, 10], the arrival condition is non-trivial as PS($c$) and Opt may have different decision on admitting a new job.

**Arrival condition.** When a job $j$ arrives at time $t$, there are four cases depending on whether PS($c$) and Opt admit $j$. We first consider the two easier cases where PS($c$) discards $j$. Since PS($c$) discards $j$, all critical intervals $I_i$'s, their densities $\rho_i$'s and $w_a(t_{i-1}, t_i)$'s do not change. Furthermore, $w_o(t_{i-1}, t_i)$ may only increases. Hence, $\Delta\Phi(t) \leq 0$. On the other hand, $\Delta D_o(t) \geq 0$ depending on whether $j$ is discarded by Opt, so the arrival condition holds.

The following discussion considers the case where PS($c$) admits $j$. For simplicity, we first assume that $p(j)$ is small so that admitting $j$ only affect the density of the critical interval that contains $d(j)$ while all other critical intervals are unaffected. Let $I_k$ be the only interval affected and let $\rho$ and $\rho'$ be the density of $I_k$ just before and after $j$ is admitted, respectively. Let $w_a(k)$ and $w_o(k)$ denote the total remaining size for jobs in PS($c$) and Opt, respectively, with deadlines in $I_k$ just before $j$ is admitted. Let $|I_k|$ denote $t_k - t_{k-1}$. Then, $\rho = \frac{w_a(k)}{|I_k|}$, and $\rho' = \frac{w_a(k)+p(j)}{|I_k|}$. We first bound $\Delta\Phi(t)$.

**Lemma 4.** *Let $\Delta\Phi(t)$ be the change in $\Phi(t)$ if $j$ is admitted by PS($c$) and discarded by Opt. Then $\Delta\Phi(t) \leq \alpha^2 c^{\alpha-1} v(j)$.*

*Proof.* Note that $w_o(k)$ remains unchanged as Opt discards $j$. By definition,

$$\begin{aligned}
\Delta\Phi &= \alpha(\rho')^{\alpha-1}(w_a(k) + p(j) - \alpha w_o(k)) - \alpha\rho^{\alpha-1}(w_a(k) - \alpha w_o(k)) \\
&\leq \alpha(\rho')^{\alpha-1}(w_a(k) + p(j)) - \alpha\rho^{\alpha-1} w_a(k) \\
&= \frac{\alpha}{|I_k|^{\alpha-1}} \left((w_a(k) + p(j))^\alpha - w_a(k)^\alpha\right)
\end{aligned}$$

Note that for any convex function $f(z)$ and any real numbers $y > x$, we have $f(y) - f(x) \leq f'(y)(y - x)$, where $f'$ denotes the derivative of $f$. Putting $f(z) = z^\alpha$ where $\alpha > 1$ and consider $y = w_a(k) + p(j)$ and $x = w_a(k)$, we have that

$$\frac{\alpha}{|I_k|^{\alpha-1}} \left((w_a(k) + p(j))^\alpha - w_a(k)^\alpha\right) \leq \frac{\alpha}{|I_k|^{\alpha-1}} \alpha (w_A(k) + p(j))^{\alpha-1} p(j) = \alpha^2 (\rho')^{\alpha-1} p(j)$$

Since $j$ is admitted by PS($c$), we have $\rho' \leq c \cdot \tilde{s}(j)$ by definition. It follows that $\Delta\Phi \leq \alpha^2 c^{\alpha-1} (\tilde{s}(j))^{\alpha-1} p(j) = \alpha^2 c^{\alpha-1} u(j) p(j) = \alpha^2 c^{\alpha-1} v(j)$ □

Therefore, if Opt discards $j$, $\Delta D_o = v(j)$, so $\Delta\Phi(t) \leq \max\{\alpha^\alpha, \alpha^2 c^{\alpha-1}\} \Delta D_o$. Finally, if Opt admit $j$, $\Delta D_o = 0$. The analysis on $\Delta\Phi(t)$ is similar to that in [7]. Note that both $w_a(k)$ and $w_o(k)$ is increased by $p(j)$. Hence,

$$\begin{aligned}\Delta\Phi(t) &= \alpha(\rho')^{\alpha-1}\Big(w_a(k)+p(j)-\alpha(w_o(k)+p(j))\Big)-\alpha\rho^{\alpha-1}\Big(w_a(k)-\alpha w_o(k)\Big)\\ &= \frac{\alpha}{|I_k|^{\alpha-1}}\Big[((w_a(k)+p(j))^{\alpha-1}\Big(w_a(k)+p(j)-\alpha(w_o(k)+p(j))\Big)\\ &\quad -w_a(k)^{\alpha-1}\Big(w_a(k)-\alpha w_o(k)\Big)\Big]\end{aligned}$$

The last term is at most zero by setting $q = w_a(k)$, $r = w_o(k)$, $\delta = p(j)$ to Lemma 5. Hence, $\Delta\Phi(t) \le 0 = \max\{\alpha^\alpha, \alpha^2 c^{\alpha-1}\}\Delta D_o$.

**Lemma 5.** *( [7]) Let $q, r, \delta \ge 0$ and $\alpha \ge 1$, then $(q+\delta)^{\alpha-1}(q+\delta-\alpha(r+\delta)) - q^{\alpha-1}(q-\alpha r) \le 0$.*

So far, we assume that $p(j)$ is small and only one critical interval is affected. If $p(j)$ is large, we follow the technique of [7, 10]. We split $j$ into two jobs $j_1$ and $j_2$ so that their release times, deadlines and value densities are the same as $j$, and $p(j_1)$ is the smallest size such that some critical intervals merge or a critical interval splits. $p(j_2) = p(j) - p(j_1)$. Note that $\Phi(t)$ does not change due to merging or splitting of critical intervals. The above argument can show that the arrival condition holds after $p(j_1)$ is admitted. Furthermore, we can repeat the division recursively on $j_2$ to conclude that the arrival condition holds.

**Running condition.** Analysis for the running condition is similar to [7]. Consider any time $t$ without job arrival. Let $s_a(t)$ and $s_o(t)$ be the speed of PS($c$) and Opt, respectively. Then $E_a(t)$ and $E_o(t)$ are increasing at the rates of $(s_a(t))^\alpha$ and $(s_o(t))^\alpha$ while $D_o(t)$ remains constant. Note that to prove the running condition, it is sufficient to prove that $(s_a(t))^\alpha + \frac{d}{dt}\Phi(t) - \alpha^\alpha(s_o(t))^\alpha \le 0$. In the following, we omit the parameter $t$ for simplicity. E.g., we write $s_a$ to mean $s_a(t)$. PS($c$) processes jobs by OA, which processes jobs by EDF. So at time $t$, PS($c$) is processing a job with deadline in $I_1$. $s_a = \rho_1$, so $w_a(t_0, t_1)$ is decreasing at a rate of $s_a$. Suppose Opt is processing a job with deadline in $I_k$, where $k \ge 1$. Then $w_o(t_{k-1}, t_k)$ is decreasing at a rate of $s_o$. Therefore $\frac{d}{dt}\Phi = \alpha\rho_1^{\alpha-1}(-s_a) + \alpha^2\rho_k^{\alpha-1}s_o \le \alpha\rho_1^{\alpha-1}(-s_a) + \alpha^2\rho_1^{\alpha-1}s_o = -\alpha s_a^\alpha + \alpha^2 s_a^{\alpha-1}s_o$, where the inequality comes from $\rho_k \le \rho_1$. Finally, $s_a^\alpha + \frac{d}{dt}\Phi - \alpha^\alpha s_o^\alpha \le (1-\alpha)s_a^\alpha + \alpha^2 s_a^{\alpha-1}s_o - \alpha^\alpha s_o^\alpha$. The last expression can be shown to be non-positive by differentiation.

## 4 Bounded Speed Model

We first define the penalty ratio of a job sequence.

**Definition 2.** *Consider scheduling in the bounded speed model with maximum speed $T$. The* penalty ratio *of a job, denoted $\Gamma(j)$, equals $\tilde{s}(j)/T$. The* penalty ratio *of a sequence $J$ of jobs, denoted $\Gamma(J)$ or simply $\Gamma$ if $J$ is clear in context, equals the maximum penalty ratio of all jobs in $J$, i.e., $\Gamma = \max_{j\in J}\Gamma(j)$.*

### 4.1 Lower Bound

**Theorem 2.** *For the bounded speed model, any algorithm has competitive ratio at least $\min\{\Gamma^{\alpha-2+1/\alpha}, \frac{1}{2}\Gamma^{\alpha-1}\}$, where $\Gamma$ is the penalty ratio of the job sequence.*

*Proof.* Let Alg be any algorithm. The theorem is obviously true if $\Gamma \le 1$. In the following, let $\Gamma > 1$ be the targeted penalty ratio. Let $x > 1$ be a variable

to be set later. At time 0, release a job $j_1$ with $d(j_1) = x$, $p(j_1) = T$ and $v(j_1) = T^\alpha \Gamma^{\alpha-1}$. Note that $\tilde{s}(j_1) = (v(j_1)/p(j_1))^{1/(\alpha-1)} = T\Gamma$ and $\Gamma(j_1) = \tilde{s}(j_1)/T = \Gamma$. At time 1, one of the following two cases occurs.

- If Alg has completed $j_1$ by time 1, Alg must run at speed $T$ during $[0, 1]$. The total energy usage is $T^\alpha$. Opt can run at speed $\frac{T}{x}$ during $[0, x]$ to finish $j_1$, with total energy usage $(\frac{T}{x})^\alpha x$. So the competitive ratio is $\frac{T^\alpha}{(T/x)^\alpha(x)} = x^{\alpha-1}$
- If Alg has not completed $j_1$ at time 1, another job $j_2$ is released at time 1 with $d(j_2) = x$, $p(j_2) = T(x-1)$, and $v(j_2) = T^\alpha \Gamma^{\alpha-1}(x-1)$. Note that $\tilde{s}(j_2) = T\Gamma$ and $\Gamma(j_2) = \Gamma$. Opt can complete both $j_1$ and $j_2$ by running at speed $T$ throughout $[0, x]$, with total energy usage $T^\alpha x$. Alg cannot complete both $j_1$ and $j_2$ by their deadlines. If Alg discards $j_1$, the competitive ratio is at least $\frac{v(J_1)}{T^\alpha x} = \frac{\Gamma^{\alpha-1}}{x}$; if Alg discards $j_2$, it is at least $\frac{v(j_2)}{T^\alpha x} = \Gamma^{\alpha-1} - \frac{\Gamma^{\alpha-1}}{x}$.

Note that $\Gamma(j_1) = \Gamma(j_2) = \Gamma$, so the penalty ratio of the input sequence is $\Gamma$. The competitive ratio is at least $k = \min\{x^{\alpha-1}, \frac{\Gamma^{\alpha-1}}{x}, \Gamma^{\alpha-1} - \frac{\Gamma^{\alpha-1}}{x}\}$. If $\Gamma \geq 2^{\frac{\alpha}{\alpha-1}}$, we set $x = \Gamma^{\frac{\alpha-1}{\alpha}}$, then $x \geq 2$, $\Gamma^{\alpha-1} = x^\alpha \geq 2x^{\alpha-1}$ and $k \geq x^{\alpha-1} = \Gamma^{\alpha-2+1/\alpha}$. If $\Gamma < 2^{\frac{\alpha}{\alpha-1}}$, we set $x = 2$ and $k \geq \frac{1}{2}\Gamma^{\alpha-1}$. □

Note that $\frac{1}{2}\Gamma^{\alpha-1} = \Omega(\Gamma^{\alpha-2+1/\alpha})$. When $T$ is large, the $e^{\alpha-1}/\alpha$ lower bound from the unbounded speed model holds. Hence, for bounded speed model, any algorithm is $\Omega(\max\{e^{\alpha-1}/\alpha, \Gamma^{\alpha-2+1/\alpha}\})$ -competitive.

### 4.2 Algorithm BPS

We propose an algorithm BPS($c$) (Bounded Profitable Speed with parameter $c$). We show that it is $O(\alpha^\alpha + 2\Gamma^{\alpha-1}(\alpha+1)^{\alpha-1})$-competitive. BPS($c$) maintains a list $Q$ of admitted jobs, which is empty initially and maintained as follows.

**Algorithm BPS(c)**
- **Job execution.** At any time, BPS($c$) uses OA to schedule the jobs in $Q$.
- **Job admission.** When a job $j$ arrives at $r(j)$, let $S$ be the set of jobs remaining in $Q$ just before $j$ arrives. BPS($c$) calculates the OA schedule for $S \cup \{j\}$. Let $s(j)$ be the speed of $j$ in this OA schedule. BPS($c$) admits $j$ into $Q$ if $s(j) \leq \min\{c \cdot \tilde{s}(j), T\}$; and $j$ is discarded immediately otherwise.
- **Job completion.** When a job is completed, remove it from $Q$.

In our analysis, $c = 1$ gives the best competitive ratio for BPS($c$), hence, to ease our discussion, we will fix $c = 1$ and call the resulting algorithm BPS. Note that if $\Gamma \leq 1$, then $\min\{\tilde{s}(j), T\} = \tilde{s}(j)$, and then BPS and PS(1) will admit the same set of jobs and consequently have the identical schedule. By putting $c = 1$ and $b = \alpha + 1$ into Theorem 1, PS(1) is $O(\alpha^\alpha)$-competitive in the unbounded speed model, which implies that BPS is $O(\alpha^\alpha)$-competitive in the bounded speed model for $\Gamma \leq 1$. Hence, in the following, we assume $\Gamma > 1$.

**Theorem 3.** *BPS is* $\left((1 + \Gamma^{\alpha-1}\frac{b^{\alpha-1}}{(b-1)^\alpha})\max\{\alpha^\alpha, \alpha^2\} + \Gamma^{\alpha-1}b^{\alpha-1}\right)$*-competitive in the bounded speed model for any* $b > 1$*, where* $\Gamma > 1$ *is the penalty ratio.*

Putting $b = \alpha + 1$ for $\alpha \geq 2$, and $b = \alpha^{2/\alpha} + 1$ for $1 < \alpha < 2$, we conclude that BPS is $\left(\alpha^2 + 2\Gamma^{\alpha-1}(\alpha^{2/\alpha}+1)^{\alpha-1}\right)$-competitive for $1 < \alpha < 2$, and $\left(\alpha^\alpha + 2\Gamma^{\alpha-1}(\alpha+1)^{\alpha-1}\right)$ -competitive for $\alpha \geq 2$. To prove Theorem 3, consider any job sequence $J'$. Let Opt$'$ be the optimal offline algorithm in the bounded speed model. Let $E'_a$ and $E'_o$ be the energy usage of BPS and Opt$'$, respectively. Let $D'_a$ and $D'_o$ be the value discarded by BPS and Opt$'$, respectively. Theorem 3 follows from the following two inequalities. The proofs are left to the full paper.

- $D'_a \leq \Gamma^{\alpha-1}\left(\frac{b^{\alpha-1}}{(b-1)^\alpha}E'_a + b^{\alpha-1}E'_o\right) + D'_o$, for any $b > 1$.
- $E'_a \leq \max\{\alpha^\alpha, \alpha^2\}(E'_o + D'_o)$.

## References


1. Albers, S.: Energy-efficient algorithms. ACM Communications 53(5), 86–96 (2010)
2. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms 3(4) (2007)
3. Andrew, L., Wierman, A., Tang, A.: Optimal speed scaling under arbitrary power functions. ACM SIGMETRICS Performance Evaluation Review 37(2), 39–41 (2009)
4. Bansal, N., Chan, H.L., Lam, T.W., Lee, L.K.: Scheduling for bounded speed processors. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 409–420. Springer, Heidelberg (2008)
5. Bansal, N., Chan, H.-L., Pruhs, K.: Speed scaling with an arbitrary power function. In: SODA, pp. 693–701 (2009)
6. Bansal, N., Chan, H.-L., Pruhs, K., Katz, D.: Improved bounds for speed scaling in devices obeying the cube-root rule. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 144–155. Springer, Heidelberg (2009)
7. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. JACM 54(1) (2007)
8. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 805–813 (2007)
9. Belady, C.: In the data center, power and cooling costs more than the it equipment it supports. Electronics Cooling Magazine 13(1), 24–27 (2007), `http://electronics-cooling.com/articles/2007/feb/a3/`
10. Chan, H.L., Chan, W.T., Lam, T.W., Lee, L.K., Mak, K.S., Wong, P.W.H.: Energy efficient online deadline scheduling. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 795–804 (2007)
11. Koren, G., Shasha, D.: $D^{over}$: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. SIAM J. Comput. 24(2), 318–339 (1995)
12. Lam, T.W., Lee, L.K., To, I.K.K., Wong, P.W.H.: Speed scaling functions for flow time scheduling based on active job count. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 647–659. Springer, Heidelberg (2008)
13. Markoff, J., Lohr, S.: Intel's huge bet turns iffy. New York Times (September 29, 2002)
14. Pruhs, K., Stein, C.: How to schedule when you have to buy your energy. To appear in RANDOM-APPROX (2010)
15. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Foundations of Computer Science (FOCS), pp. 374–382 (1995)

# New Models and Algorithms for Throughput Maximization in Broadcast Scheduling

## (Extended Abstract)

Chandra Chekuri[1,⋆], Avigdor Gal[2], Sungjin Im[1,⋆⋆], Samir Khuller[3,⋆⋆⋆], Jian Li[3], Richard McCutchen[3,†], Benjamin Moseley[1,‡], and Louiqa Raschid[4]

[1] Dept. of Computer Science, Univ. of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801
[2] Faculty of Industrial Engineering & Management, Technion – Israel Institute of Technology, Technion City, Haifa 32000, Israel
[3] Dept. of Computer Science and UMIACS, Univ. of Maryland, College Park, MD 20742
[4] Robert H. Smith School of Business and UMIACS, Univ. of Maryland, College Park, MD 20742

**Abstract.** In this paper we consider some basic scheduling questions motivated by query processing that involve accessing resources (such as sensors) to gather data. Clients issue requests for data from resources and the data may be dynamic or changing which imposes temporal constraints on the delivery of the data. A proxy server has to compute a probing schedule for the resources since it can probe a limited number of resources at each time step. Due to overlapping client requests, multiple queries can be answered by probing the resource at a certain time. This leads to problems related to some well-studied broadcast scheduling problems. However, the specific requirements of the applications motivate some generalizations and variants of previously studied metrics for broadcast scheduling. We consider both online and offline versions of these problems and provide new algorithms and results.

## 1 Introduction

There is an explosion in the amount of data being produced, collected, disseminated and consumed. One important source of this data is from sensors that are being widely deployed to monitor and collect a variety of information, for example, weather and traffic. Another important source of data comes from individuals and entities publishing content on the web. Two aspects of the above type of data are the following. First, a consumer/client is typically interested only in some small subset of the available data that is relevant to her. Second, the data has temporal relevance and a client is also typically interested in data that is within some time interval of interest to her; for example

⋆ Partially supported by NSF grants CCF-0728782 and CNS-0721899.
⋆⋆ Supported mainly by a Samsung Fellowship, and partially by NSF grants CCF-0728782 and CNS-0721899.
⋆⋆⋆ Supported by NSF Awards CCF-0728839 and CCF-0937865, and a Google Research Award.
† Research supported by REU Award supplement to CCF-0728839.
‡ Partially supported by NSF grant CNS-0721899.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 71–82, 2011.

traffic on a particular road during her commute window. These trends have necessitated a significant increase in the sophistication of data delivery capabilities to keep up with quantity of the data, and the need for client customization [27]. There is a large effort in several areas of computer science to address these issues. Typically, software known as middleware handles the interface between the clients and data sources. In this paper we consider certain scheduling problems that arise in processing the queries.

Middleware primarily consists of proxy servers that collect client queries and access data sources (such as sensors) to answer queries [11,15,5,25,12,28]. In this work we consider a basic and central question that arises when the queries are time sensitive (they also may be periodic) such as "Give me the reading of sensor A at 15 mins after the hour, every hour". The main challenge is to schedule probes to the data sources (e.g., sensors), to obtain the data at the desired time for the clients. Due to processing limitations, the proxy server is limited to probing only a small number of sensors at each time step (we assume for simplicity that it probes one sensor at each time step). However, by probing a sensor at a particular time, multiple overlapping queries requesting data from this sensor, can be answered.

More formally, there are clients that issue queries for data from a resource at a specific time, by specifying an interval of time when the resource should be queried. A central server collects all the queries and needs to design a schedule to probe the resources to answer client queries. When a resource is probed, several client queries can be answered. Typically, the queries are simple and so the computational requirements are minimal; hence we focus on the design of the probing schedule. For example, by identifying overlapping queries to the same resource we may be able to significantly reduce the number of times we query the sensors, since we can "piggyback" all the queries [26,27,28]. This overlapping nature of query processing, is very similar to the manner in which broadcast scheduling problems are approached [2,20,21,17,6]. However, the sensor probing application gives rise to new and interesting variants of broadcast scheduling problems. In this work we focus on a collection of online and offline problems motivated by the above application.

In the broadcast scheduling literature, three objectives have been the focus of study: (i) minimizing average response time[1] [20,1,13,14,16,6,18] (and many others), (ii) minimizing maximum response time [2,6,8], and (iii) maximizing throughput [21,4,10,29]. By response time of a request, we mean the time from the arrival of the request to when it is satisfied. The first two metrics apply to settings in which all requests are to be satisfied. The third metric is relevant in situations where requests may not be satisfied beyond a certain time; in particular, the following model has been studied: Each request has a release time and deadline and it can only be satisfied within its time window and the goal is to maximize the number/weight of satisfied requests. We next explain why these metrics are not directly suitable for our purposes.

In sensor probing, the requests are time sensitive which calls for a more nuanced view of "satisfying" a request. For example, if a client requests the temperature reading, or traffic conditions at 5:30pm, then we may satisfy this query by reporting the value at 5:33pm, this would have a latency of 3 minutes. Suppose we report the value at 5:40pm with a latency of 10 minutes; the data may still be useful to the client but perhaps less

[1] Response time is also commonly referred to as flow time.

than reporting the value at 5:33pm. Finally, the data may be irrelevant if the latency is say more than 20 minutes. This example demonstrates the two objectives of interest in a schedule. We are interested in "completeness", the number of client requests that can be satisfied before their deadline. We are also interested in the "latency" of those requests that we do satisfy. As can be seen, previous metrics do not capture the combination of these metrics; minimizing average response time ignores deadlines and maximizing throughput with deadlines ignores the latency of satisfied requests.

In this paper we take two approaches to finding schedules that address both completeness and latency. In the first approach, we associate an arbitrary *time-dependent profit* function with each query. The profit function can take into account the impact of the latency on the value to the client. The goal then would be to find a schedule that maximizes the total profit of the requests. This model captures the previously studied maximum throughput metric, but allows more control over the quality of the schedule for queries. We consider both offline and online settings and obtain several new results. In the second approach, we directly address the tradeoff between completeness and latency. In addition to satisfying as many requests as possible we hope to also satisfy them close to when the request arrived (the arrival time could be the ideal time when the sensor should be probed). We formalize this in the following way: among all schedules that achieve a desired level of completeness, choose the one that minimizes latency of satisfied requests.

Finally, we consider another variant of the maximum throughput problem that is relevant to our application domain. In some cases, it is perfectly reasonable to report the value "before" the arrival time of the request. In the same example above, the proxy server may have the value of a sensor measuring temperature that has been probed at 5:28pm while a request for the same sensor arrives at 5:30pm. The server can use the reading at 5:28pm to answer the query. We model this aspect in two ways: by relaxing the time window of interest to the client both forwards and backwards in time, and by considering unimodal profit functions.

All the problems we consider are NP-hard in the offline setting via simple reductions from known results on broadcast scheduling [6]. We, therefore, focus on the design of efficient approximation algorithms. We also consider online variants and use the standard competitive analysis framework; for some variants we analyse the algorithms in the resource augmentation framework [19] wherein the algorithm is given extra speed over the adversary. We give below a formal description of the problems considered in the paper, followed by our results.

**Problem Definitions:** For convenience we shall use the standard broadcast scheduling notation of referring to pages instead of referring to sensors. We are given a set of pages $\mathcal{P} = \{p_1, \ldots, p_n\}$. We assume that time is slotted, $\mathcal{T} = \{1, 2, \ldots, T\}$ where $T$ is an upper bound on the schedule length. Suppose a client sends a request for page $p$, which arrives at time $a$ and is associated with deadline $d$. If the server broadcasts $p$ at some time slot $t$ such that $a \leq t \leq d$, we say the request is *satisfied*. We assume the server can broadcast at most one page in a single time slot. We use $J_{p,i}$ to denote the $i$th request for page $p$, which has the arrival time $a_{p,i} \in \mathbb{Z}^+$ and the deadline $d_{p,i} \in \mathbb{Z}^+$. Sometimes, we will consider a generalized request which may be associated with more than one interval. We use $\mathcal{T}_{p,i}$ to denote the set of time slots during which $J_{p,i}$ can be satisfied.

For example, if $J_{p,i}$ has only one interval, then $\mathcal{T}_{p,i} = \{a_{p,i}, a_{p,i}+1, \ldots, d_{p,i}\}$. In this paper, we study the following objective functions.

1. Maximizing throughput (MAX-THP): The objective is to maximize the total number of satisfied requests. In the weighted version of MAX-THP, each request $J_{p,i}$ has a weight $w_{p,i}$. In this case, the objective is to maximize the total weight of all satisfied requests.
2. Maximizing total profit (MAX-PFT): This is a significant generalization of MAX-THP. In a MAX-PFT instance, each request $J_{p,i}$ is associated with an arbitrary non-negative profit function $g_{p,i} : \mathcal{T} \rightarrow \mathbb{Z}^+$. The interpretation is that the request $J_{p,i}$ obtains a value/profit of $g_{p,i}(t)$ if it is satisfied by the broadcast of $p$ at time $t$. However, $p$ may be broadcast multiple times during a schedule. In that case the request $J_{p,i}$ obtains a profit $\max_{t \in \mathcal{T}_p^A} g_{p,i}(t)$ where $\mathcal{T}_p^A$ is the set of time slots in which $p$ was broadcast by a given schedule. The objective is to find a schedule $A$ such that the total profit is maximized.
3. Completeness-Latency tradeoff: We are given a completeness threshold $C \in (0, 1]$. The goal is to find a schedule that completes $C$ fraction of the requests before their deadline and subject to that constraint, minimizes the latency of the completed requests.

**Outline of Results:** We obtain several results for the problems described above. We give a high-level description of these results below.

**Maximizing Throughput and Profit:** Recall that MAX-PFT is a significant generalization of MAX-THP. There is a $3/4$-approximation for the MAX-THP problem [17] via a natural LP relaxation. We adapt the ideas in [17] to obtain a $3/4$-approximation for the special case of MAX-PFT when the profit functions for each query are unimodal (see Section 3), which is of particular interest to our setting. Second, for the general MAX-PFT problem we obtain a $(1 - 1/e)$-approximation, again via the natural LP relaxation. In addition, we show that the MAX-PFT problem can be cast as a special case of submodular function maximization subject to a matroid constraint. This allows us to not only obtain a different $(1 - 1/e)$-approximation but also several generalizations and additional properties via results in [3,9]. The connection also allows us to easily show that the greedy algorithm gives a $1/2$ approximation for MAX-PFT in the *online* setting, generalizing prior work that showed this for MAX-THP [21].

We also consider how the approximation ratios and competitive ratios for MAX-THP and MAX-PFT can be improved via resource augmentation and other relaxations. We show that there is a 2-speed 1-approximation for MAX-THP. Previously, such a result was known only if all requests could be scheduled in a fractional solution [6]. In the online-setting we show that the simple greedy algorithm with $s$-speed achieves a $s/(s+1)$ competitive ratio for MAX-PFT. In a different direction we consider relaxing the time window in MAX-THP and prove the following result. If there is a fractional schedule that satisfies all the client requests (obtained by solving the LP relaxation to the IP), then there is an integral schedule with the following property: each request $J_{p,i}$ is satisfied in a window $[a_{p,i} - L, d_{p,i} + L]$ where $L = d_{p,i} - a_{p,i}$ is the window length of $J_{p,i}$. In other words, by either left shifting the window or right shifting the window by its length, we can always satisfy the request.

**Completion-Latency Tradeoff:** We show that there is an interesting tradeoff that can be obtained between latency and completeness when each request has an associated deadline. Given a fractional LP solution (obtained by relaxing the IP) for minimizing the total latency subject to a certain completeness level, we show that we can use randomized rounding to obtain a schedule with the following properties: the expected completeness of the schedule is $\frac{3}{4}C$, where $C$ is the completeness of the fractional schedule and the expected latency of the scheduled requests is $D(C)$ where $D(C)$ is the minimum fractional latency with completeness requirement $C$. The details of this result are deferred to a full version of the paper.

We also prove another result in broadcast scheduling that is of interest. This concerns the problem of minimizing the maximum response time. The first-in-first-out (FIFO) algorithm is 2-competitive in the online setting [2,6,8] and this is also the best known off-line approximation known. Moreover, it is known that in the online setting no deterministic algorithm is $(2-\epsilon)$-competitive for any $\epsilon > 0$ [2,6]. Here, we show that the same lower bound holds even for *randomized* online algorithms in the oblivious adversary model. The details of this result are omitted due to space constraints.

## 2 Preliminaries

Several of our results rely on the dependent randomized rounding framework of [17]. We first describe the LP relaxation for MAX-THP that is used as the basis for the rounding process.

### 2.1 An LP Relaxation for MAX-THP

We consider a natural integer programming (IP) formulation for MAX-THP. We use the indicator variable $Y_p^{(t)}$. $Y_p^{(t)} = 1$ if page $p$ is broadcast in time-slot $t$ and $Y_p^{(t)} = 0$ otherwise. In addition we define variables $X_{p,i}$ for the request $J_{p,i}$. This variable is 1 if and only if $J_{p,i}$ is satisfied.

$$\text{maximize} \quad \sum_{p,i} w_{p,i} X_{p,i} \tag{1}$$

$$\text{subject to} \quad \forall p,t, \sum_{t \in \mathcal{T}_{p,i}} Y_p^{(t)} \geq X_{p,i} \quad \text{(If } p \text{ is not broadcast in } \mathcal{T}_{p,i}, J_{p,i} \text{ cannot be satisfied)}$$

$$\forall t, \sum_{p} Y_p^{(t)} \leq 1 \quad \text{(One page broadcast at one time-slot)}$$

$$\forall p,t, \; X_{p,i} \in \{0,1\}, Y_p^{(t)} \in \{0,1\}$$

By letting the domain of $X_{p,i}$ and $Y_p^{(t)}$ be $[0,1]$, we obtain the linear programming (LP) relaxation for the problem.

### 2.2 Dependent Rounding Scheme of [17]

We briefly describe the dependent randomized rounding method of [17]. Suppose we are given a bipartite graph $(A, B, E)$ with bipartition $(A, B)$. We are also given a value $x_{i,j} \in$

$[0,1]$ for each edge $(i,j) \in E$. The scheme in [17] provides a randomized polynomial-time algorithm that rounds each $x_{i,j}$ to a random variable $X_{i,j} \in \{0,1\}$, effectively keeping or dropping the edge, in such a way that the following properties hold.

**(P1): Marginal distribution.** For every edge $(i,j)$, $\Pr[X_{i,j}=1] = x_{i,j}$.

**(P2): Degree-preservation.** Consider any vertex $i \in A \cup B$. Define its fractional degree $d_i$ to be $\sum_{j:(i,j)\in E} x_{i,j}$, and integral degree $D_i$ to be the random variable $\sum_{j:(i,j)\in E} X_{i,j}$. Then, $D_i \in \{\lfloor d_i \rfloor, \lceil d_i \rceil\}$. Note in particular that if $d_i$ is an integer, then $D_i = d_i$ with probability 1.

**(P3): Negative correlation.** For any vertex $i$ and any subset $S$ of the neighbors of $i$:

$$\forall b \in \{0,1\},\ \Pr[\bigwedge_{j\in S}(X_{i,j}=b)] \le \prod_{j\in S}\Pr[X_{i,j}=b]. \tag{2}$$

We refer the reader to [17] for more details.

## 3 Throughput and Profit Maximization

This section is devoted to offline and online algorithms for MAX-THP and MAX-PFT.

### 3.1 Offline Algorithms

**Maximizing the Total Profit.** In this section, we consider the profit maximization (MAX-PFT) problem. Recall that in a MAX-PFT instance, each request $J_{p,i}$ is associated with a profit function $g_{p,i}(t) \ge 0$ that is an arbitrary non-negative function of the time it is satisfied. If a request for page $p$ is satisfied multiple times by a schedule $A$, the profit we can get for $p$ is the maximum one, i.e., $\max_{t\in\mathcal{T}_p^A} g_{p,i}(t)$. The objective is to find a schedule $A$ such that the total profit is maximized. Note that MAX-THP is just a special case of MAX-PFT where the profit function $g_{p,i}(t)$ is 1 for $a_{p,i} \le t \le d_{p,i}$.

First, we show how to reduce MAX-PFT to MAX-THP with weighted requests where each request may have multiple intervals. We use a simple slicing trick described as follows. Consider a single request $J_{p,i}$, and let $v_1 < v_2 < \ldots < v_r$ be the distinct nonnegative values taken on by its profit function $g_{p,i}$. Let $v_0 = 0$. We create $r$ new requests for the throughput maximization instance, say $J_{p,i,j}, 1 \le j \le r$, which all require page $p$. $J_{p,i,j}$ has weight $v_j - v_{j-1}$ and intervals consisting of time slots $\{t \mid g_{p,i}(t) \ge v_{j-1}\}$. See Figure 1. It is not hard to show the following lemma.

**Lemma 1.** *The total (weighted) throughput of a schedule $A$ for the constructed* MAX-THP *instance equals its total profit when interpreted as a schedule for the original* MAX-PFT *instance and vise versa.*

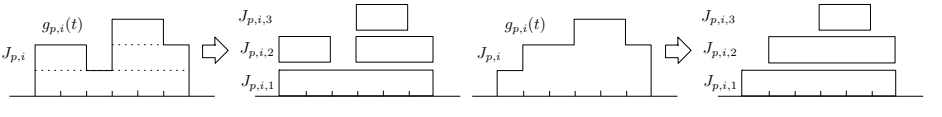



**Fig. 1.** Illustrations of the slicing trick. The left hand side is a request with a general profit function and the right hand side is one with a unimodal profit function.

If each profit function is *unimodal*, meaning that it is non-decreasing up to a point and non-increasing after that point, we observe that the slicing trick should create requests each having only one request interval since the time slots $\{t \mid g_{p,i}(t) > v_{r-1}\}$ are consecutive (see the right hand side of Figure 1). Therefore, we can apply any approximation algorithm that works for the weighted throughput maximization problem with one interval for each request and obtain the same approximation ratio for profit maximization with unimodal profit functions. The best known approximation ratio for weighted throughput maximization is $3/4$ due to Gandhi et al. [17].

**Theorem 1.** *For arbitrary non-negative unimodal profit functions, there is a* $3/4$*-approximation for* MAX-PFT.

However, if the profit function is not unimodal, the resulting MAX-THP instance may contain requests that have multiple request intervals. Next, we show that a simple independent rounding scheme that gives a $(1-1/e)$-approximation for MAX-THP with each request associated with one or more intervals, which implies a $(1-1/e)$-approximation for MAX-PFT.

Let $x_{p,i}, y_p^{(t)}$ be the optimal fractional solution of LP (1). Consider the following simple independent rounding scheme: Consider each time slot $t$ independently and choose exactly one page to broadcast. Page $p$ is chosen with probability $y_p^{(t)}$. Note that this is feasible since $\sum_p y_p^{(t)} \leq 1$. We can easily lower bound the probability that a request is satisfied by the schedule produced by the independent rounding.

**Lemma 2.** *Using independent rounding, the probability that a request* $J_{p,i}$ *is satisfied is at least* $(1 - 1/e)x_{p,i}$.

The expected total number of requests captured is thus

$$\sum_{p,i} \Pr[J_{p,i} \text{ is satisfied}] \geq (1 - \frac{1}{e}) \sum_{p,i} w_{p,i} x_{p,i} \geq (1 - \frac{1}{e}) OPT.$$

We thus conclude:

**Theorem 2.** *For any non-negative profit functions, there is a* $(1 - 1/e)$*-approximation for* MAX-PFT.

MAX-PFT **via Submodular set function maximization:** An alternative algorithm achieving the same ratio can also be obtained by casting MAX-PFT as a special case of the problem of maximizing a monotone submodular set function subject to a matroid constraint. For recent progress on constrained submodular set function maximization, see e.g. [3] and reference therein.

First we give the definition of matroid. Let $N$ be a finite set and $\mathcal{I}$ be a family of subsets of $N$. The pair $(N, \mathcal{I})$ is called matroid if $\mathcal{I}$ satisfies the following properties. (1) $\mathcal{I}$ is non-empty, (2) downward closed: if $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$, and (3) independent: if $A, B \in \mathcal{I}$ and $|A| < |B|$, then $A \cup \{x\} \in \mathcal{I}$ for some $x \in B \setminus A$. One special matroid is a partition matroid. In a *partition* matroid, $N$ is partitioned into $N_1$, $N_2$, ... ,$N_\ell$ with associated integers $k_1, k_2, ..., k_\ell$, and $A \in \mathcal{I}$ if and only if $\forall i \, |A \cap N_i| \leq k_i$. Next we give the definition of monotone submodular set function $f : 2^N \rightarrow \mathcal{R}^+$.

The function $f$ is called submodular when $f(A+x) - f(A) \leq f(B+x) - f(B)$ whenever $B \subseteq A$ and $x \in N$. By monotonicity, we mean that if $B \subseteq A$ then $f(B) \leq f(A)$, and $f(\emptyset) = 0$. The problem of maximizing the submodular function $f$ under the matroid constraint $(N, \mathcal{I})$ can be formulated as finding $A = \arg\max_{A' \in \mathcal{I}} f(A')$.

We interpret MAX-PFT as a special case of the above general problem in the following way. Let $N = \mathcal{P} \times \mathcal{T}$, where $\mathcal{P}$ is the set of pages and $\mathcal{T}$ is the set of time slots. Let $N_t = \mathcal{P} \times \{t\}$. Let $A \in \mathcal{I}$ iff $\forall t \, |A \cap N_t| \leq 1$. Notice that $(N, \mathcal{I})$ is a partition matroid. The function $f$ is defined as follows: $f = \sum_{p,i} \max_{t:(p,t) \in A} g_{p,i}(t)$. It is not hard to see that $f$ is a monotone submodular set function. It is known that maximizing a monotone submodular function can be approximated with factor $1 - \frac{1}{e}$ under a matroid constraint [3]. Therefore, we can obtain a $1 - \frac{1}{e}$-approximation for MAX-PFT.

The advantage of the alternative algorithm above is the following. Once the connection to submodular functions and matroid constraints is seen, one can readily obtain similar results for more general settings. For example, it is possible that a client request can be satisfied by sending any one of several similar pages. In this case, as long as, one is able to define an appropriate submodular profit function, one again obtains a $(1-1/e)$-approximation. Moreover, one can also impose additional constraints as long as they satisfy a matroid constraint; multiple matroid constraints can also be handled with some additional loss in the approximation. Finally, one can also obtain concentration bounds in some cases [9] and these can be useful in handling additional constraints. We defer a detailed description of these extensions to a later version of the paper.

**A 2-Speed 1-Approximation for Throughput Maximization.** In this section, our goal is to show a randomized 2-speed 1-approximation for throughput maximization. Here the objective is to satisfy as many requests as possible by their deadlines. Recall that in [6] gave an algorithm to convert a fractional solution that satisfies all requests to a 2-speed integer solution with the same property. To obtain a true 1-approximation, we need to also handle the case where the fractional solution does not satisfy all requests. Our analysis relies on the result of [6]. For completeness, we begin by showing that if there is a feasible fractional solution to LP (1) that satisfies all requests, there is a 2-speed integral scheduling that can also satisfy all requests. Then we show how to extend this to obtain the 2-speed 1-approximation by using dependent rounding.

Let $x_{p,i}, y_p^{(t)}$ be a fractional solution to the LP where all requests are satisfied by their deadlines. We first construct a bipartite graph $G = (U, V, E)$ as follows. One partite set $U$ contains a vertex $u_t$ representing each time slot $t$. Let $z_p^{(t)} = \sum_{t'=1}^{t} y_p^{(t)}$ be the cumulative amount of page $p$ transmitted through time $t$ in the fractional solution. Let $I_{p,t} = [z_p^{(t-1)}, z_p^{(t)})$, which represents the fractional occurrences of page $p$ transmitted during time slot $t$.

The other partite set $V$ contains, for each page $p$ and for $i = 1$ to $\lceil 2z_p^{(T)} \rceil$, a vertex $v_{p,i}$ representing the $i$th fractional transmission of half of page $p$. Let $I(v_{p,i}) = [(i-1)/2, i/2)$ and let $W_{p,i}$ be the *window* of consecutive time slots $t$ such that $I_{p,t}$ overlaps $I(v_{p,i})$. For each $t \in W_{p,i}$, we add an edge $(u_t, v_{p,i})$. See Figure 2 for an illustration of the construction.

Now we augment the bipartite graph to get a network flow instance. We add a source **s**, edges $(\mathbf{s}, u_t)$ with capacity 1 for $\forall t$, a sink **t** and edges $(v_{p,i}, \mathbf{t})$ with capacity $1/2$ for

$\forall p, i$. First we can observe that there is **s**-**t** flow $f$ with flow value $\sum_p z_p^{(T)}$. In fact, just by letting $f(u_t, v_{p,i})$ be the measure (i.e., length) of $I_{p,t} \cap I(v_{p,i})$ and setting $f(\mathbf{s}, u_t) \forall t$ and $f(v_{p,i}, \mathbf{t}) \forall i$ accordingly, $f$ is such a flow. For each page $p$, we delete the last vertex $v_{p,\lceil 2z_p^{(T)} \rceil}$ if $f(v_{p,\lceil 2z_p^{(T)} \rceil}, \mathbf{t}) < 1/2$. After this, we can see that $f$ saturates all edges $(v_{p,i}, \mathbf{t}) \forall p, i$.

Next, we double the capacities of all edges and find a maximum **s**-**t** integral flow $f'$. This is possible since all capacities are now integral. The obtained integral flow can be interpreted as a 2-speed schedule: If there is a unit of flow going from $u_t$ to $v_{p,i}$, the server will broadcast $p$ at time $t$. Since the capacity of $(\mathbf{s}, u_t)$ is 2, at most 2 pages are broadcast in one time slot. Note that all edges $(v_{p,i}, \mathbf{t})$ are saturated. This in turn means that for each window $W_{p,i}$, the server broadcasts $p$ at least once in some time slot $t \in W_{p,i}$. For each request $J_{p,i}$, we know that $\sum_{t \in \mathcal{T}_{p,i}} y_p^{(t)} \geq 1$. Therefore, some window $W_{p,j}$ is fully contained in $\mathcal{T}_{p,i} = \{a_{p,i}, \ldots, d_{p,i}\}$ due to the construction of the windows. Hence, all requests are satisfied by the 2-speed schedule.

Now, we generalize the above idea to get a true 2-speed 1-approximation, that is a schedule such that the server broadcasts at most 2 pages in one time slot and satisfies at least the number of requests that can be satisfied by the optimum 1-speed schedule. The idea is very simple, instead of scaling the capacities, we just take the bipartite graph $G$ and the flow $f$, and do dependent rounding on $G$ with values $2 \times f$. We notice that all $f$ values defined on the edges of $G$ are at most $1/2$. Therefore, $2 \times f$ are valid probabilities. Consider a request $J_{p,i}$ which is not completely satisfied, i.e., $\sum_{t \in \mathcal{T}_{p,i}} y_p^{(t)} < 1$. In this case, $x_{p,i} = \sum_{t \in \mathcal{T}_{p,i}} y_p^{(t)}$. It is easy to see that $\mathcal{T}_{p,i}$ is fully contained in two windows $W_{p,j}, W_{p,j+1}$ for some $j$. Let $y = \sum_{t \in \mathcal{T}_{p,i}} f(u_t, v_{p,j})$. By (**P2**) of the dependent rounding scheme, we know at most 1 edge incident on a window can be chosen. Therefore, for fixed $p, j$, the events that $(u_t, v_{p,j})$ is chosen are disjoint. Then, by (**P1**),

$$\Pr[(u_t, v_{p,j}) \text{ is chosen for some } t \in \mathcal{T}_{p,i}] = 2 \sum_{t \in \mathcal{T}_{p,i}} y_p^{(t)} = 2y.$$

Similarly, we can show that $\Pr((u_t, v_{p,j+1}) \text{ is chosen for some } t \in \mathcal{T}_{p,i}) = 2(x - y)$. Therefore, $\Pr(J_{p,i} \text{ is satisfied}) \geq \max(2y, 2(x_{p,i} - y)) \geq x_{p,i}$. If $J_{p,i}$ is completely satisfied, we can use the previous argument, that is $\mathcal{T}_{p,i}$ fully contains some window $W_{p,j}$ and some edge incident to $v_{p,i}$ must be chosen. Again, we have $\Pr(J_{p,i} \text{ is satisfied}) = 1 = x_{p,i}$. Since we have shown that each request is satisfied with a probability no smaller than the probability that the request is satisfied in the fractional optimal solution, we obtain the following theorem.

**Theorem 3.** *There is a polynomial time* 2*-speed* 1*-approximation for* MAX-THP.

**Throughput Maximization with a Relaxed Time Window.** In this section, we assume that each request is fractionally fully satisfied by the optimal solution of LP(1), i.e., $x_{p,i} = 1 \ \forall p, i$. Suppose a request $J_{p,i}$ arrives at time $a_{p,i}$ with deadline $d_{p,i}$ (associated with the window $[a_{p,i}, d_{p,i}]$), then we construct an integral schedule such that this request is satisfied within the window $[a_{p,i} - l_{p,i}, d_{p,i} + l_{p,i}]$ where $l_{p,i} = d_{p,i} - a_{p,i} + 1$ is the length of the window $\mathcal{T}_{p,i}$. By left shifting the window or right shifting the window by its length, we can satisfy the request. A shifting, or expanding of the window is necessary and we refer to this as a relaxed schedule since it satisfies all the requests in

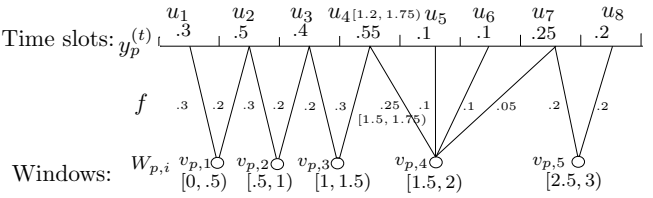



**Fig. 2.** The construction of the bipartite graph $G(U, V, E)$. E.g. $I_{p,4} = [1.2, 1.75)$, $I(v_{p,4}) = [1.5, 2)$ and $I((u_4, v_{p,4})) = I_{p,4} \cap I(v_{p,4}) = [1.5, 1.75]$.

a relaxed manner, and the client request is satisfied at a time approximately close to the desired window of time. For a given instance, we consider a fractional solution which, by assumption, satisfies all requests before their deadlines.

Starting from the instance $I$ that has a fractional solution in which every request is satisfied, we will create an instance $\mathcal{I}$ which is a subset of the requests such that finding an integral solution for $\mathcal{I}$ will also immediately lend a *relaxed integral solution* to the instance $\mathcal{I}$. We focus on a single page $p$. Order all the client requests for page $p$ in order of non-decreasing window length. Initially $\mathcal{I}$ is the empty set of requests. We try to insert each request (in non-decreasing window length order) into set $\mathcal{I}$, and as long as it does not overlap with a request already inserted into $\mathcal{I}$, we insert it. This will give us a collection of non-overlapping requests for page $p$. We do this filtering for every possible page. This gives us a fractional solution in which all requests for the same page are non-overlapping and completely satisfied. Using flow based methods[2] it is easy to convert this to an integral solution that satisfies all the requests. Client requests in $\mathcal{I}$ are clearly satisfied (integrally) within their intervals. Each client request $J_{p,i}$ that was not chosen in $\mathcal{I}$ overlapped with a chosen request with a smaller window size. Thus it is also satisfied in the integral solution within time $[a_{p,i} - l_{p,i}, d_{p,i} + l_{p,i}]$,i.e., satisfied within the relaxed deadlines. We thus conclude:

**Theorem 4.** *Suppose there is a fractional schedule that satisfies all requests. We can convert the fractional solution to an integral one in polynomial time such that each request $J_{p,i}$ can be satisfied in the relaxed window $[a_{p,i} - l_{p,i}, d_{p,i} + l_{p,i}]$ where $l_{p,i} = d_{p,i} - a_{p,i} + 1$ is the length of the window $\mathcal{T}_{p,i}$.*

### 3.2 Online Algorithms

In this section we revisit the problem MAX-PFT, but now in the *online* setting, in which a request is not known to the server until it arrives. As previously discussed in Section 3, maximizing the total profit can be interpreted as maximizing a monotone submodular function subject on a matroid. It is known that a simple greedy algorithm gives 2-approximation for such a problem [24]. Further, the greedy algorithm can be interpreted as an online algorithm in this setting. Thus we can easily obtain a 2-competitive algorithm for MAX-PFT. For the more restricted setting MAX-THP, [21] gave a

[2] This involves the same technique as used for converting a fractional matching in a bipartite graph to an integral matching [16].

2-approximation. Here we show that the greedy algorithm's performance improves in the resource augmentation model when the algorithm is given a speed larger than 1. There is no natural way to interpret resource augmentation in the general framework of submodular set function maximization subject to a matroid constraint. We therefore resort to a direct analysis.

We will be considering a resource augmentation analysis [19]. In this analysis, the online algorithm is given $s$-speed and compared to a 1 speed optimal solution. For some objective function, we say that an algorithm is $s$-speed $c$-competitive if the algorithm's objective is within a $c$ factor of the optimal solution's objective. We describe our greedy algorithm *Maximum Additional Profits First* (for short, MAPF) which is given an integer speed $s \geq 1$. As implied in its name, the algorithm MAPF broadcasts $s$ pages which give the maximum additional profits by broadcasting.

**Algorithm.** MAPF
– At any time $t$, broadcast $s$ pages which give the maximum additional profits.

For this algorithm we show the following theorems. Proofs are deferred to a full version of this paper.

**Theorem 5.** MAPF *is* $s$*-speed* $(1 + 1/s)$*-competitive online algorithm for* MAX-PFT *and* MAPF *is* $s$*-speed* $(1 + 1/s)$*-competitive algorithm for* MAX-THP.

**Theorem 6.** *For any* $\epsilon > 0$ *and speed* $s \geq 1$, MAPF *is not* $s$*-speed* $(1 + 1/s - \epsilon)$*-competitive for* MAX-THP.

For $s = 1$, for any $\epsilon > 0$, there is a lower bound of $(2 - \epsilon)$ on the competitive ratio of any online algorithm, even if it is randomized [21].

## References

1. Bansal, N., Coppersmith, D., Sviridenko, M.: Improved approximation algorithms for broadcast scheduling. In: SODA, pp. 344–353 (2006)
2. Bartal, Y., Muthukrishnan, S.: Minimizing maximum response time inscheduling broadcasts. In: SODA, pp. 558–559 (2000)
3. Calinescu, G., Chekuri, C., Pal, M., Vondrak, J.: Maximizing a monotone submodular set function subject to a matroid constraint. SIAM J. on Computing (to appear)
4. Chan, W., Lam, T., Ting, H., Wong, P.: New Results on On-Demand Broadcasting with Deadline via Job Scheduling with Cancellation. In: Chwa, K.-Y., Munro, J.I.J. (eds.) COCOON 2004. LNCS, vol. 3106, pp. 210–218. Springer, Heidelberg (2004)
5. Carney, D., Lee, S., Zdonik, S.: Scalable Application-Aware Data Freshening. In: ICDE, pp. 481–492 (2003)
6. Chang, J., Erlebach, T., Gailis, R., Khuller, S.: Broadcast Scheduling: Algorithms and Complexity. In: SODA, pp. 473–482 (2008)
7. Charikar, M., Khuller, S.: A robust maximum completion time measure for scheduling. In: SODA, pp. 324–333 (2006)
8. Chekuri, C., Im, S., Moseley, B.: Minimizing Maximum Response Time and Delay Factor in Broadcast Scheduling. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 444–455. Springer, Heidelberg (2009)

9. Chekuri, C., Vondrak, J., Zenklusen, R.: Dependent Randomized Rounding via Exchange Properties of Combinatorial Structures. In: FOCS (2010)
10. Chrobak, M., Dürr, C., Jawor, W., Kowalik, L., Kurowski, M.: A Note on Scheduling Equal-Length Jobs to Maximize Throughput. J. of Scheduling 9(1), 71–73 (2006)
11. Deolasee, P., Katkar, A., Panchbudhe, P., Ramamritham, K., Shenoy, P.: Adaptive Push-Pull: Deisseminating Dynamic Web Data. In: WWW (2001)
12. Eckstein, J., Gal, A., Reiner, S.: Monitoring an Information Source under a Politeness Constraint. INFORMS Journal on Computing 20(1), 3–20 (2007)
13. Edmonds, J., Pruhs, K.: Multicast pull scheduling: when fairness is fine. In: SODA, pp. 421–430 (2002)
14. Edmonds, J., Pruhs, K.: A maiden analysis of longest wait first. In: SODA, pp. 811–820 (2004)
15. Gal, A., Eckstein, J.: Managing Periodically Updated Data in Relational Databases: A Stochastic Modeling Approach. JACM 48(6), 1141–1183 (2001)
16. Gandhi, R., Khuller, S., Kim, Y., Wan, Y.C.: Algorithms for minimizing response time in broadcast scheduling. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 415–424. Springer, Heidelberg (2002)
17. Gandhi, R., Khuller, S., Parthasarathy, S., Srinivasan, A.: Dependent rounding and its applications to approximation algorithms. JACM 53(3), 324–360 (2006); Preliminary version in FOCS (2002)
18. Im, S., Moseley, B.: An Online Scalable Algorithm for Average Flow Time in Broadcast Scheduling. In: SODA (2010)
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. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 290–301. Springer, Heidelberg (2000)
21. Kim, J., Chwa, K.: Scheduling broadcasts with deadlines. Theor. Comput. Sci. 325, 479–488 (2004)
22. Motwani, R., Raghavan, P.: Randomized Algorithms. ACM Comput. Surveys 28(1), 33–37 (1996)
23. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
24. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions - I. Mathematical Programming 14(1), 265–294 (1978)
25. Pandey, S., Dhamdhere, K., Olston, C.: WIC: A General-Purpose Algorithm for Monitoring Web Information Sources. In: VLDB, pp. 360–371 (2004)
26. Roitman, H., Gal, A., Raschid, L.: Satisfying Complex Data Needs using Pull-Based Online Monitoring of Volatile Data Sources. In: ICDE (2008)
27. Roitman, H.: Profile Based Online Data Delivery - Model and Algorithms. Ph.D. Thesis (2008)
28. Roitman, H., Gal, A., Raschid, L.: On Trade-offs in Event Delivery Systems. In: The 4th ACM International Conference on Distributed Event-Based Systems, DEBS (2010)
29. Zheng, F., Fung, S., Chan, W., Chin, F., Poon, C., Wong, P.: Improved On-Line Broadcast Scheduling with Deadlines. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 320–329. Springer, Heidelberg (2006)

# Densest $k$-Subgraph Approximation on Intersection Graphs

Danny Z. Chen[1,⋆], Rudolf Fleischer[2,⋆⋆], and Jian Li[3]

[1] University of Notre Dame, Indiana, USA
dchen@cse.nd.edu
[2] Fudan University, SCS and IIPL, Shanghai, China
rudolf@fudan.edu.cn
[3] University of Maryland, College Park, MD, USA
lijian@cs.umd.edu

**Abstract.** We study approximation solutions for the densest $k$-subgraph problem (DS-$k$) on several classes of intersection graphs. We adopt the concept of $\sigma$-quasi elimination orders, introduced by Akcoglu et al. [1], generalizing the perfect elimination orders for chordal graphs, and develop a simple $O(\sigma)$-approximation technique for graphs admitting such a vertex order. This concept allows us to derive constant factor approximation algorithms for DS-$k$ on many intersection graph classes, such as chordal graphs, circular-arc graphs, claw-free graphs, line graphs of $\ell$-hypergraphs, disk graphs, and the intersection graphs of fat geometric objects. We also present a PTAS for DS-$k$ on unit disk graphs using the shifting technique.

## 1 Introduction

The *(connected) densest $k$-subgraph problem* (DS-$k$) is defined as follows: Given an undirected graph $G = (V, E)$ with $n$ nodes and $m$ edges and a positive integer $k$, find an induced (connected) subgraph with $k$ vertices in $G$ maximizing the number of edges. Reduction from the maximum clique problem shows that this problem is NP-hard. The weighted version of DS-$k$ in which each edge has a positive weight and the goal is to maximize the sum of edge weights in the induced subgraph is called the *heaviest $k$-subgraph problem* (HS-$k$).

Considerable work has been done on finding good quality approximation algorithms for DS-$k$. The first non-trivial approximation algorithm by Kortsarz and Peleg achieved an approximation ratio of $O(n^{0.3885})$ [23]. Arora et al. [3] used random sampling techniques to obtain a polynomial time approximation scheme

⋆ The research of this author was supported in part by the US National Science Foundation under Grants CCF-0515203 and CCF-0916606. This work was partially done while the author was visiting the Shanghai Key Laboratory of Intelligent Information Processing at Fudan University, China.

⋆⋆ The research of this author was supported by the NSFC (no. 60973026), the Shanghai Leading Academic Discipline Project (no. B114), and the Shanghai Committee of Science and Technology (no. 09DZ2272800).

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 83–93, 2011.

(PTAS) for dense graphs with $k = \Omega(|V|)$ and $|E| = \Omega(|V|^2)$. Asahiro et al. [4] showed that the greedy method achieves an approximation ratio of $O(\frac{n}{k})$.

Feige et al. proposed an $\frac{n}{k}$-approximation algorithm based on semidefinite programming [15] and an $n^\delta$-approximation algorithm for some $\delta < \frac{1}{3}$ [14]. Recently, Bhaskara et al. [7] proposed an $O(n^{1/4+\epsilon})$ approximation algorithm that runs in time $n^{O(1/\epsilon)}$. In [11], Demaine et al. gave a 2-approximation algorithm for $H$-minor-free graphs, for any fixed $H$. It is unlikely that there exists a PTAS for general graphs [21].

For some special graph classes and special values of $k$, better algorithms are known [19,32,34]. Maffioli proposed an $O(nk^2)$ time algorithm for connected HS-$k$ on trees [28]. This algorithm can easily be generalized to solve the unconnected case. Corneil and Pearl gave a polynomial time algorithm for DS-$k$ on co-graphs, a subclass of perfect graphs [10].

Keil and Brecht developed polynomial time algorithms for HS-$k$ on graphs with bounded treewidth based on dynamic programming [20]. Liazi et al. [25] presented a polynomial time algorithm for DS-$k$/HS-$k$ on chains (i.e., graphs with maximum degree 2), and a subclass of proper interval graphs. They also obtained a PTAS for chordal graphs if the maximal clique intersection graph is a star, and polynomial time algorithms if the maximal clique intersection graph is a tree of bounded degree [26]. Recently, they showed that a simple greedy algorithm achieves an approximation factor of 3 for DS-$k$ on chordal graphs [27].

Finding dense subgraphs with upper or lower bounds on the size of the subgraphs has also been studied by several researchers [2,22].

*Our Results.* In this paper, we focus on DS-$k$ on several intersection graph classes: chordal graphs, circular-arc graphs, line graphs, disk graphs, and unit disk graphs. The closely related maximum clique problem is polynomial time solvable on these graph classes, except on disk graphs. Note that interval graphs are chordal graphs, and chordal graphs are perfect graphs. Although the maximum clique problem is polynomial time solvable on perfect graphs [18], DS-$k$ is NP-hard on perfect graphs, since it is NP-hard on bipartite graphs [30] and chordal graphs [10]. Connected DS-$k$ is NP-hard on planar graphs [20]. The complexity status of unconnected DS-$k$ on planar graphs, interval graphs, and proper interval graphs has been a long-standing open problem [10].

Since the complexity status of these problems is unknown, it is worthwhile to consider efficient approximation algorithms for them. We adopt the notion of $\sigma$-quasi elimination orders, for $\sigma \geq 1$, proposed by Akcoglu et al. [1], generalizing the perfect elimination orders for chordal graphs. It turns out that many intersection graph classes mentioned above have $O(1)$-quasi elimination orders [35]. This type of vertex order allows us to derive new approximation algorithms for DS-$k$. Our main result is an $O(\sigma)$-approximation algorithm for DS-$k$ if the graph has a polynomial time computable $\sigma$-quasi elimination order. This immediately implies constant factor approximation ratios for many intersection graph classes. These classes include chordal graphs (with $\sigma = 1$), circular-arc graphs (with $\sigma = 2$), claw-free graphs (with $\sigma = 2$), line graphs of $\ell$-hypergraphs (with $\sigma = \ell$), disk graphs (with $\sigma = 5$), unit disk graphs (with $\sigma = 3$), and the intersection

graphs of $\lambda$-fat objects in $d$-dimensional space (with $\sigma = (3\lambda)^d$). We also propose a PTAS for DS-$k$ on unit disk graphs based on the shifting technique [6] combined with a result by Arora et al. [3], if a disk representation is given. This improves the recent 1.5-approximation for DS-$k$ on proper interval graphs [5]. Note that the class of proper interval graphs is equivalent to the class of unit interval graphs [31] which is a subset of the class of unit disk graphs.

## 2 Preliminaries

For a graph $G = (V, E)$, we denote its vertex set by $V(G) = V$ and its edge set by $E(G) = E$. Let $n = |V|$ and $m = |E|$. We denote the degree of a vertex $v$ in $G$ by $deg_G(v)$. For any $v \in V$ and subsets $S, W \subseteq V$, let $d(v, W)$ be the number of edges $(v, w)$ with $w \in W$, and $d(S, W) = \sum_{u \in S} d(u, W)$. Let $G[S]$ denote the subgraph of $G$ induced by $S \subset V$. Let $\alpha(G)$ be the *independence number* of $G$, i.e., the size of a maximum independent set in $G$. The classic Turán bound states that

$$\alpha(G) \geq \sum_{v \in V} \frac{1}{deg(v) + 1} \geq \frac{n}{\bar{d} + 1}$$

where $\bar{d}$ is the average degree of the nodes in the graph [33]. For convenience, we rephrase the bound in the following lemma.

**Lemma 1.** *For any graph $G$, $m \geq \dfrac{n^2 - n\alpha(G)}{2\alpha(G)}$.* □

## 3 Elimination Orders and Intersection Graphs

If $\mathcal{L} = (v_1, v_2, \ldots, v_n)$ is an ordering of the vertices in $V$, we define $Pred_{\mathcal{L}}(v_i) = \{v_i\} \cup \{v_j \mid j < i \text{ and } (v_j, v_i) \in E\}$, the *predecessors* of $v_i$, and $Succ_{\mathcal{L}}(v_i) = \{v_j \mid j > i \text{ and } (v_i, v_j) \in E\}$, the *successors* of $v_i$. In a *perfect elimination order*, every set $Pred_{\mathcal{L}}(v_i)$ forms a clique (note that sometimes in the literature it is required that every set $Succ_{\mathcal{L}}(v_i)$ forms a clique, instead, which just reverses the order). We can generalize this definition by allowing some slack. Let $\sigma$ be a positive integer.

**Definition 2.** *A* $\sigma$-quasi elimination order ($\sigma$-QEO) *of $G$ is an ordering $\mathcal{L} = (v_1, v_2, \ldots, v_n)$ of the vertices in $V$ such that $\alpha(G[Pred_{\mathcal{L}}(v_i)]) \leq \sigma$ for $i = 2, \ldots, n$.*

A perfect elimination order is just a 1-QEO. This notion was introduced by Akcoglu et al. [1] who proposed a $\sigma$-approximation for the weighted maximum independent set problem. Recently, Ye and Borodin explored many properties of QEOs and initiated a more comprehensive study on their algorithmic aspects [35]. In particular, they considered the maximum $\sigma$-colorable subgraphs problem, the minimum vertex covering problem and the minimum vertex coloring problem and obtained improved approximation algorithms on graphs with $O(1)$-QEO. Lemma 1 implies that $G[Pred_{\mathcal{L}}(v_i)]$ has at least $\frac{1}{2\sigma} \cdot \binom{|Pred_{\mathcal{L}}(v_i)|}{2}$ edges, for every

$v_i$ in $\mathcal{L}$, if $|Pred_{\mathcal{L}}(v_i)| \geq 2\sigma - 1$. Note that any induced subgraph of $G$ has a $\sigma$-QEO if $G$ has one. In this paper, we study the following graph classes.

*Chordal graphs.* $G$ is a *chordal graph* if it does not contain an induced cycle of length $k$, for any $k \geq 4$. Chordal graphs are exactly the intersection graphs of subtrees in a tree. A graph is chordal if and only if it has a perfect elimination order [17].

*Circular-arc graphs.* A *circular-arc graph* is the intersection graph of arcs of a circle. Circular-arc graphs are not always chordal, for example any chordless cycle of length greater than four is a circular-arc graph. It is easy to see that any circular-arc graph has a 2-QEO.

*Line graphs.* A graph $L$ is the *line graph* of the (hyper-)graph $G$ if $L$ is the intersection graph of the (hyper-)edges of $G$.

*Claw-free graphs.* A graph $G$ is *claw-free* if it excludes $K_{1,3}$ as an induced subgraph. Claw-free graphs generalize line graphs, which initially motivated the study of claw-free graphs. They have many nice properties, for example, claw-free graphs always have a perfect matching and we can find a maximum independent set in polynomial time. However, it is NP-hard to compute a largest clique in a claw-free graph. For a survey on more results on claw-free graph, see [24], for example. Conveniently, any ordering of the vertices of a claw-free graph is a 2-QEO.

*(Unit) Disk graphs.* $G$ is a (unit) disk graph if it is the intersection graph of a set of closed (unit) disks in the plane. The *disk representation* specifies the centers and radii of the disks. If the disks are not given, the recognition problem of (unit) disk graphs is NP-hard [9]. Disk graphs are a two-dimensional generalization of interval graphs. However, in general, they are neither planar nor perfect. Some NP-hard problems become tractable on unit disk graphs (e.g., the maximum clique problem [8]), and some problems admit significantly better approximation algorithms (e.g., there is a PTAS for the maximum independent set problem on unit disk graphs [29] and on arbitrary disk graphs [13]). Ye and Borodin showed that any (unit) disk graph has a (3-QEO) 5-QEO [35].

*Fat intersection graphs.* Practical instances of geometric problems often deal with objects of "reasonable" shape. One way to formalize this is the notion of fat objects. There are several different definitions of fat objects in computational geometry literature (e.g., see [12]). In this paper, we say a $d$-dimensional convex object $K$ is $\lambda$-*fat*, for some parameter $\lambda \geq 1$ (the *fatness*), if the ratio between the radii of $B_K^+$ and $B_K^-$ is at most $\lambda$, where $B_K^+$ is a smallest sphere containing $K$ and $B_K^-$ is a largest sphere contained in $K$. Examples of objects of bounded fatness are spheres (fatness 1), cubes (fatness $\sqrt{d}$), and ellipsoids with bounded aspect ratio.

A *fat intersection graph* is the intersection graph of a set of fat objects. For example, disk graphs are fat intersection graphs.

**Lemma 3.** *Every fat intersection graph of $\lambda$-fat convex objects in $d$-dimensional space has an $O((3\lambda)^d)$-QEO.*

*Proof.* We sort the vertices of the graph in non-increasing order of the largest disk contained in each corresponding fat object. Then, for each vertex $v_i$, $\alpha(G[Pred_{\mathcal{L}}(v_i)]) = O((3\lambda)^d)$. Since similar ideas have been used before in the literature on algorithms for fat objects (e.g., see [12]), we omit the details of the proof. □

## 4 Approximating DS-$k$ on Graphs with $\sigma$-QEO

In this section, we present a constant factor approximation technique for DS-$k$ on chordal graphs and fat intersection graphs. We focus on presenting the general framework and do not emphasize on fine-tuning the parameters for the smallest possible approximation factor. We use the maximum density subgraph problem, which is polynomial time solvable, as a key subroutine.

### 4.1 The Maximum Density Subgraph Problem (MDSP)

The *maximum density subgraph problem (MDSP)* is defined as follows: Given a graph $G = (V, E, w)$ with non-negative vertex weights $w : V \to \mathbb{R}^+ \cup \{0\}$, find an induced subgraph $H = (W, F)$ maximizing the density

$$\rho(H) = \frac{\sum_{v \in W} w(v) + |F|}{|W|}.$$

This problem can be solved optimally in $O(nm \log(\frac{n^2}{m}))$ time by a reduction to the parametric maximum flow algorithm [16] which produces an induced subgraph $H = (W, F)$ maximizing $\frac{\sum_{e \in F} w(e)}{\sum_{v \in W} w(v)}$, where $w$ is a weight function on the vertices and edges (we set the weights of all original vertices and edges to 1; then we create a sibling with weight zero for each vertex in $V$, connected to the original vertex by an edge of weight $w(v)$).

Note that $w(v) + deg_H(v) \geq \rho$, for each vertex $v \in W$, for any optimal MDSP solution $H = (W, F)$ with maximum density $\rho$. This is because we could delete from $H$ all vertices violating this inequality to obtain an induced subgraph of higher density.

### 4.2 A Constant Factor Approximation Framework

In this subsection, we show how to compute an $O(\sigma)$-approximation for DS-$k$ on any graph $G = (V, E)$ for which we can efficiently compute a $\sigma$-quasi elimination order.

At a high level, our framework works as follows. If we solve MDSP on $G$ with $w(v) = 0$ for all $v \in V$ and obtain a subgraph $H$ of $k'$ vertices, then $H$ is also an optimal DS-$k'$ solution. If $H$ is smaller than the sought DS-$k$ solution (i.e., $k' < k$), then we repeat the MDSP algorithm on the remaining vertices of $G$ and

combine the solution with $H$ (Phase 1). If $H$ is larger (i.e., $k' > k$), then we select some vertices in $H$ to satisfy the cardinality constraint without losing too much density (Phase 2).

Let $G^\star = (V^\star, E^\star)$ be an optimal DS-$k$ solution on $G = (V, E)$ with density $\rho^\star = \frac{|E^\star|}{|V^\star|}$. Without loss of generality assume $\rho^\star \geq 8\sigma$; otherwise, we can trivially get an $O(\sigma)$-approximation.

**Phase 1: Growing $U_t$.** Let $V_0 = V$, $E_0 = E$, and $w_0(v) = 0$ for all $v \in V_0$. Starting with $i = 0$, let $G_{i+1}$ be obtained from $G_i$ by removing the vertices and adjacent edges of an optimal MDSP solution $H_i = (W_i, F_i)$ of $G_i = (V_i, E_i, w_i)$ of density $\rho_i$, where $w_i(v) = d(v, U_{i-1})$ for $v \in V_i$. Let $U_i = \cup_{j=0}^{i} W_j$ be the set of all removed nodes, and $n_i = |U_i|$. We stop at the first time $t$ such that $n_t \geq \frac{k}{2}$. If $n_t \leq k$, then we return $U_t$ plus some arbitrary $k - n_t$ nodes from $V_{t+1}$ as our DS-$k$ approximation.

**Lemma 4.** *If $n_t \leq k$, then $U_t$ is a 4-approximation for DS-$k$ on $G$.*

*Proof.* If $G[U_t \cap V^\star]$ has at least $\frac{|E^\star|}{2}$ edges, then $U_t$ is a 2-approximation for DS-$k$ on $G$. If not, then let $I_i = U_i \cap V^\star$ and $R_i = V^\star \setminus I_i$, for all $i$. Since $|E(I_t)| = |E(U_t \cap V^\star)| < \frac{|E^\star|}{2}$, we have for $i \leq t$

$$\begin{aligned}
\rho_i &= \frac{|F_i| + d(U_{i-1}, W_i)}{|W_i|} \geq \frac{|E(R_{i-1})| + d(U_{i-1}, R_{i-1})}{|R_{i-1}|} \\
&\geq \frac{|E(R_{i-1})| + d(I_{i-1}, R_{i-1})}{|R_{i-1}|} \geq \frac{|E(R_{i-1})| + d(I_{i-1}, R_{i-1})}{k} \\
&= \frac{|E^\star| - |E(I_{i-1})|}{k} \geq \frac{|E^\star| - |E(I_t)|}{k} \geq \frac{|E^\star|}{2k} = \frac{\rho^\star}{2} \,.
\end{aligned}$$

Hence,

$$|E(U_t)| \geq \sum_{i \leq t} (\rho_i \cdot |U_i|) \geq \min_{i \leq t}\{\rho_i\} \cdot \sum_{i \leq t} |U_i| \geq \min_{i \leq t}\{\rho_i\} \cdot k/2 \geq \frac{|E^\star|}{4} \,. \quad \square$$

**Phase 2: Shrinking $U_t$.** If $n_t > k$, then we must delete some vertices from $U_t$ without decreasing the density too much. We first compute a $\sigma$-quasi elimination order $\mathcal{L} = \{v_1, \ldots, v_{n_t}\}$ for $U_t$. If some vertex in $\mathcal{L}$ has a large predecessor set in this order, then we are done, as shown by Lemma 5.

**Lemma 5.** *If there is a vertex $v \in U_t$ with $|Pred_{\mathcal{L}}(v)| \geq \frac{k}{2}$, then we can efficiently find a subgraph of $\frac{k}{2}$ vertices in $Pred_{\mathcal{L}}(v)$ that is an $O(\sigma)$-approximation for DS-$k$ on $G$.*

*Proof.* Let $\mathcal{A} = Pred_{\mathcal{L}}(v)$. Since $|\mathcal{A}| \geq \frac{k}{2} \geq \rho^\star \geq 2\sigma - 1$, the $\sigma$-quasi elimination order property implies that $G[\mathcal{A}]$ has at least $\frac{1}{2\sigma} \cdot \binom{|\mathcal{A}|}{2}$ edges by Lemma 1. We

randomly and uniformly choose a subset $\mathcal{B}$ of the $\frac{k}{2}$ vertices in $\mathcal{A}$. Then, $G[\mathcal{B}]$ has an expected number of $\Theta(\frac{1}{\sigma}) \cdot \Theta(k^2)$ edges:

$$\sum_{e \in E(\mathcal{A})} \frac{k}{2|\mathcal{A}|} \cdot \frac{k}{2|\mathcal{A}|} \geq \frac{k^2}{16\sigma} \cdot (1 - \frac{1}{|\mathcal{A}|}) \geq \frac{1}{\sigma} \cdot (\frac{k^2}{16} - \frac{k}{8}) .$$

It is straightforward to derandomize this algorithm using the conditional probability technique. We omit the details. □

If no vertex $v$ in $\mathcal{L}$ has a predecessor set of size at least $\frac{k}{2}$, then we must work a bit harder to find a dense subgraph.

**Lemma 6.** *If there is no vertex $v \in U_t$ with $|Pred_{\mathcal{L}}(v)| \geq \frac{k}{2}$, then we can efficiently find a subset of $U_t$ of size at most $k$ that is an $O(\sigma)$-approximation for DS-$k$ on $G$.*

*Proof.* From the remark at the end of Subsection 4.1, we see that for any vertex $v \in U_t$, either (1) $|Succ_{\mathcal{L}}(v)| > \frac{\rho_t}{2}$, or (2) $|Pred_{\mathcal{L}}(v)| \geq \frac{\rho_t}{2}$. We now process the vertices of $U_t$ in the reverse order of $\mathcal{L}$, i.e., beginning at $v_{n_t}$. If a vertex satisfies condition (1) above, then we take it. If it satisfies condition (2), then we take it together with a certain subgraph of high-degree vertices of its predecessor set (see Lemma 7 below).

We stop if we have collected at least $\frac{k}{2}$ vertices. In every step, we either add a single vertex $v$ or a subset of its predecessors to the solution. Since no vertex has a predecessor set of size at least $\frac{k}{2}$, we select at most $k$ vertices in total, i.e., we obtain a feasible solution $SOL$ for DS-$k$.

In $G[SOL]$, each vertex $v$ has a degree either at least $\frac{\rho_t}{2}$ if it was selected by condition (1), or $\frac{|Pred_{\mathcal{L}}(v)|-1}{4\sigma} \geq \frac{\rho_t - 2}{8\sigma}$ if it was selected by condition (2). Thus,

$$|SOL| = \frac{1}{2} \sum_{v \in SOL} deg(v) \geq \frac{\rho_t - 2}{8\sigma} \cdot k \geq \frac{\rho^\star - 4}{8\sigma} \cdot k = O(\frac{1}{\sigma}) \cdot |E^\star| .$$

□

**Lemma 7.** *If $|Pred_{\mathcal{L}}(v)| \geq \frac{\rho_t}{2}$ for some vertex $v$, then we can efficiently identify a non-empty subset $\mathcal{H}$ of $Pred_{\mathcal{L}}(v)$ such that every vertex in $G[\mathcal{H}]$ has a degree at least $\frac{|Pred_{\mathcal{L}}(v)|-1}{4\sigma}$.*

*Proof.* We repeatedly delete a vertex of degree less than $\frac{|Pred_{\mathcal{L}}(v)|-1}{4\sigma}$. Since $|Pred_{\mathcal{L}}(v)| \geq \frac{\rho_t}{2} \geq \frac{\rho^\star}{4} \geq 2\sigma$, $G[Pred_{\mathcal{L}}(v)]$ contains at least $\frac{1}{2\sigma} \cdot \binom{|Pred_{\mathcal{L}}(v)|}{2}$ edges, and thus we cannot delete all vertices (and their edges) of $Pred_{\mathcal{L}}(v)$. □

**Theorem 8.** *If $G$ has a polynomial time computable $\sigma$-QEO, then we can efficiently compute an $O(\sigma)$-approximation for DS-$k$ on $G$.* □

It is known that a $\sigma$-QEO can be constructed in $O(\sigma^2 n^{\sigma+2})$ time if there is one [35]. In particular, we can find an $O(1)$-QEO in polynomial time. Combined with Theorem 8, we obtain the claimed results on intersection graphs.

**Corollary 9.** *There is an $O(1)$-approximation algorithm for DS-$k$ on the following intersection graph classes, even if the intersection models are not given as input: chordal graphs, circular-arc graphs, claw-free graphs, line graphs of $\ell$-hypergraphs (with $\ell = O(1)$), disk graphs (with $\sigma = 5$), unit disk graphs (with $\sigma = 3$), and the intersection graphs of $\lambda$-fat objects in $d$-dimensional space (with $\lambda = O(1)$ and $d = O(1)$).* □

## 5 A PTAS for DS-$k$ on Unit Disk Graphs

A PTAS for DS-$k$ on unit disk graphs can be obtained by a standard shifting technique [6], combined with a result by Arora et al. [3]. This technique was also used to develop a PTAS for the maximum independent set problem on unit disk graphs [29]. We give a brief sketch of our algorithm. The following lemma indicates how to combine the optimal solutions for HS on independent subgraphs into a global optimal solution.

**Lemma 10.** *Let $G$ be a graph with connected components $G_1, \ldots, G_p$. If we can efficiently solve HS-$\ell$ on all $G_i$, for any $\ell$, then we can efficiently solve HS-$\ell$ on $G$, for any $\ell$.*

*Proof.* Let $OPT(G, \ell)$ denote an optimal solution of HS-$\ell$ on $G$. Then, $OPT(\cup_{i=1}^{j} G_i, \ell)$ can be computed by the following dynamic program, for any $j$ and $\ell$:

$$OPT(\cup_{i=1}^{j} G_i, \ell) = \max_x \{OPT(\cup_{i=1}^{j-1} G_i, x) + OPT(G_j, \ell - x)\} .$$

□

We may assume that the given disks have diameter one and the disk centers do not have integral coordinates. Let $h$ be a constant to be fixed later. For all $0 \le i, j \le k-1$, we define $\mathcal{D}_{i,j}$ to be the set of disks obtained by removing all disks intersecting a vertical line $x = i + ha$ for some integer $a$ or a horizontal line $y = j + hb$ for some integer $b$.

Let $OPT(G, k)$ be an optimal DS-$k$ solution for $G$. We can show that $\sum_{i=0}^{h-1} \sum_{j=0}^{h-1} |OPT(G, k) \cap \mathcal{D}_{i,j}| \ge (h-2)^2 \cdot |OPT(G, k)|$. Therefore, there exist $i, j$ such that $|OPT(\mathcal{D}_{i,j}, k)| \ge |OPT(G, k) \cap \mathcal{D}_{i,j}| \ge (1 - \frac{2}{h})^2 \cdot |OPT(G, k)|$. By choosing $h = 2/\epsilon$, we see that $\max_{i,j} |OPT(\mathcal{D}_{i,j}, k)|$ is a $(1-\epsilon)$-approximation. Now, we have reduced DS-$k$ on $G$ to computing $OPT(\mathcal{D}_{i,j}, k)$. In the following, we will give a PTAS for computing $OPT(\mathcal{D}_{i,j}, k)$. This gives us a PTAS for DS-$k$ on $G$.

$\mathcal{D}_{i,j}$ may consist of several connected components, each of which is contained in an $h \times h$ square. Let $C$ be one of the components with $n_c$ vertices. An $h \times h$ square can be covered by $(h+1)^2 + h^2 = 2h^2 + 2h + 1$ unit disks. Thus, $C$ can be covered by no more than $2h^2 + 2h + 1$ disjoint cliques, since a set of disks whose centers lie in a common unit circle induces a clique. Therefore, one of these cliques contains no less than $\frac{n_c}{2h^2+2h+1}$ vertices. If $k \le \frac{n_c}{2h^2+2h+1}$, then $OPT(C, k)$ is a clique. If $k > \frac{n_c}{2h^2+2h+1}$, the size of a maximum independent set

in $C$ is no more than $2h^2 + 2h + 1$. By Lemma 1, $C$ contains $\Theta(\frac{n_c^2}{h^2})$ edges. Since $h$ is a constant, we can use the algorithm in [3] to obtain a PTAS for problem instances with $\Theta(n_c^2)$ edges and satisfying $k = \Theta(n_c)$. Now, by Lemma 10, we have a PTAS for computing $OPT(\mathcal{D}_{i,j}, k)$.

We note that similar ideas can be used to obtain a PTAS for unit square intersection graphs. Erlebach et al. [13] used a new subdivision of the plane and the shifting strategy to obtain a PTAS for the maximum independent set problem and the vertex cover problem for disk graphs. However, it is not clear whether their methods can be applied to obtaining a PTAS for DS-$k$ on disk graphs.

## 6 Conclusions

In this paper, we studied approximation algorithms for the densest $k$-subgraph (DS-$k$) problem on several classes of intersection graphs. One of our main contributions is a simple $O(\sigma)$-approximation framework for graphs admitting $\sigma$-QEOs, which leads to improved approximation DS-$k$ algorithms for these graph classes.

One future research direction is to find more algorithmic applications for $\sigma$-QEO. It is worthwhile noting that after the MDSP preprocessing phase, our algorithm is essentially based on local decisions guided by the vertex ordering. This is similar to the approximation algorithms for various graph problems developed in [1,35]. Therefore, we conjecture that there might be a deeper reason to explain this, or even a unified characterization of the problem structures that allows us to apply certain local decision-based approximation algorithms on graphs with QEOs.

Note that all graph classes we considered have $\sigma$-quasi elimination orders with some constant $\sigma \geq 1$. Thus, another direction of research is to identify other graph classes with a $\sigma$-QEO such that $\sigma$ is $o(n^{1/4})$ (this ensures an approximation better than $O(n^{1/4})$ for DS-$k$ [7]).

## References

1. Akcoglu, K., Aspnes, J., DasGupata, B., Kao, M.-Y.: Opportunity cost algorithm for combinatorial auctions. In: Applied Optimization 74: Computational Methods in Decision-Making, Economics and Finance, pp. 455–479 (2002)
2. Andersen, R., Chellapilla, K.: Finding dense subgraphs with size bounds. In: Avrachenkov, K., Donato, D., Litvak, N. (eds.) WAW 2009. LNCS, vol. 5427, pp. 25–37. Springer, Heidelberg (2009)
3. Arora, S., Karger, D., Karpinski, M.: Polynomial time approximation schemes for dense instances of NP-hard problems. In: Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC 1995), pp. 284–293 (1995)
4. Asahiro, Y., Iwama, K., Tamaki, H., Tokuyama, T.: Greedily finding a dense subgraph. In: Karlsson, R., Lingas, A. (eds.) SWAT 1996. LNCS, vol. 1097, pp. 136–148. Springer, Heidelberg (1996)
5. Backer, J., Keil, J.M.: Constant factor approximation algorithms for the densest $k$-subgraph problem on proper interval graphs and bipartite permutation graphs. Information Processing Letters 110(16), 635–638 (2010)

6. Baker, B.: Approximation algorithms for NP-complete problems on planar graphs. Journal of the ACM 41(1), 153–180 (1994)
7. Bhaskara, A., Charikar, M., Chlamtac, E., Feige, U., Vijayaraghavan, A.: Detecting high log-densities — An $O(n^{1/4})$ approximation for densest $k$-subgraph. In: Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC 2010), pp. 201–210 (2010)
8. Breu, H.: Algorithmic aspects of constrained unit disk graphs. PhD thesis, University of British Columbia (1996)
9. Breu, H., Kirkpartrick, D.G.: Unit disc graph recognition is NP-hard. Computational Geometry: Theory and Applications 9, 3–24 (1998)
10. Corneil, D.G., Perl, Y.: Cluster and domination in perfect graphs. Discrete Applied Mathematics 9, 27–39 (1984)
11. Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.: Algorithmic graph minor theory: Decomposition, approximation, and coloring. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), pp. 637–646 (2005)
12. Efrat, A., Katz, M.J., Nielsenc, F., Sharir, M.: Dynamic data structures for fat objects and their applications. Computational Geometry: Theory and Applications 15, 215–227 (2000)
13. Erlebach, T., Jansen, K., Seidel, E.: Polynomial-time approximation scheme for geometric graphs. In: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001), pp. 671–679 (2001)
14. Feige, U., Kortsarz, G., Peleg, D.: The dense $k$-subgraph problem. Algorithmica 29, 410–421 (2001)
15. Feige, U., Selter, M.: On the densest $k$-subgraph problem. Technical report, Department of Applied Mathematics and Computer Science, the Weizmann Institute, Rehovot (September 1997)
16. Gallo, G., Grigoriadis, M.D., Tarjan, R.E.: A fast parametric maximum flow algorithm and applications. SIAM Journal on Computing 18(1), 30–50 (1989)
17. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
18. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Heidelberg (1988)
19. Han, Q., Ye, Y., Zhang, J.: Approximation of dense-$k$-subgraph (2000) (manuscript)
20. Keil, J.M., Brecht, T.B.: The complexity of clustering in planar graphs. The Journal of Combinatorial Mathematics and Combinatorial Computing 9, 155–159 (1991)
21. Khot, S.: Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004), pp. 136–145 (2004)
22. Khuller, S., Saha, B.: On finding dense subgraphs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 597–608. Springer, Heidelberg (2009)
23. Kortsarz, G., Peleg, D.: On choosing a dense subgraph. In: Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1993), pp. 692–701 (1993)
24. Faudree, R., Flandrin, E., Ryjáek, Z.: Claw-free graphs — A survey. Discrete Mathematics 164(1-3), 87–147 (1997)
25. Liazi, M., Milis, I., Zissimopoulos, V.: Polynomial variants of the densest/heaviest $k$-subgraph problem. In: Proceedings of the 20th British Combinatorial Conference, Durham (2005)

26. Liazi, M., Milis, I., Zissimopoulos, V.: The densest $k$-subgraph problem on clique graphs. Journal of Combinatorial Optimization 14(4), 465–474 (2007)
27. Liazi, M., Milis, I., Zissimopoulos, V.: A constant approximation algorithm for the densest $k$-subgraph problem on chordal graphs. Information Processing Letters 108(1), 29–32 (2008)
28. Maffioli, F.: Finding a best subtree of a tree. Technical Report 91.041, Politecnico di Milano, Dipartimento di Elettronica, Italy (1991)
29. Matsui, T.: Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In: Akiyama, J., Kano, M., Urabe, M. (eds.) JCDCG 1998. LNCS, vol. 1763, pp. 194–200. Springer, Heidelberg (2000)
30. Peeters, R.: The maximum edge biclique problem is NP-complete. Discrete Applied Mathematics 131(33), 651–654 (2003)
31. Roberts, F.S.: Indifference graphs. In: Harary, F. (ed.) Proof Techniques in Graph Theory. Academic Press, New York (1969)
32. Srivastav, A., Wolf, K.: Finding dense subgraphs with semidefinite programming. In: Jansen, K., Rolim, J.D.P. (eds.) APPROX 1998. LNCS, vol. 1444, pp. 181–191. Springer, Heidelberg (1998)
33. Turán, P.: On the theory of graphs. Colloquium Mathematicum 3, 19–30 (1954)
34. Ye, Y., Zhang, J.: Approximation of dense $\frac{n}{2}$-subgraph problem and the complement of min-bisection. Journal of Global Optimization 25(1), 55–73 (2003)
35. Ye, Y., Borodin, A.: Sequential elimination graphs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 774–785. Springer, Heidelberg (2009)

# The Train Delivery Problem - Vehicle Routing Meets Bin Packing

Aparna Das*, Claire Mathieu*, and Shay Mozes**

Brown University, Providence RI 02918, USA
{aparna,claire,shay}@cs.brown.edu

**Abstract.** We consider the *train delivery* problem which is a generalization of the bin packing problem and is equivalent to a one dimensional version of the vehicle routing problem with unsplittable demands. The problem is also equivalent to the problem of minimizing the makespan on a single batch machine with non-identical job sizes.

The train delivery problem is strongly NP-hard and does not admit an approximation ratio better than 3/2. We design the first approximation schemes for the general problem. We give an *asymptotic* polynomial time approximation scheme, under a notion of asymptotic that makes sense even though scaling can cause the cost of the optimal solution of any instance to be arbitrarily large. Alternatively, we give a polynomial time approximation scheme for the case where $W$, an input parameter that corresponds to the bin size or the vehicle capacity, is polynomial in the number of items or demands.

## 1 Introduction

We consider the *train delivery* problem, which is a generalization of bin packing. The problem can be equivalently viewed as a one dimensional vehicle routing problem (VRP) with unsplittable demands, or as the scheduling problem of minimizing the makespan on a single batch machine with non-identical job sizes.

In the train delivery problem we are given a positive integer capacity $W$ and a set $S$ of $n$ items, each with a positive position $p_i$ and a positive integer weight $w_i$. We seek a partition of $S$ into subsets $\{S_j\}$ (train tours) that minimizes

$$\sum_j \max_{i \in S_j} p_i \quad \text{subject to} \quad \forall j \ \sum_{i \in S_j} w_i \leq W.$$

We describe some applications of the different formulations of the train delivery problem. In the scheduling setting, integrated circuits are tested by subjecting them to thermal stress for an extended period of time (burn-in). Each circuit has a prespecified minimum burn-in time ($p_i$) and a number of boards needed ($w_i$). Since circuits may stay in the oven for a period longer than their burn-in time, it is possible to place different products as a batch in the oven

* Supported by NSF grant CCF-0728816.
** Supported by NSF grant CCF-0635089.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 94–105, 2011.

simultaneously as long as the capacity of the oven (the number of boards in the oven) is not exceeded. The processing time of each batch is the longest minimum exposure time among all the products in the batch. Once processing is begun on a batch, no product can be removed from the oven until the processing of the batch is complete. We wish to find a partition of the circuits into batches so that the total processing time of all batches (the makespan) is minimized.

In the VRP setting with unsplittable demands, containers are to be transported from a harbor to $n$ customers located at positions $p_i$ along a railway. The number of containers destined to customer $i$ is $w_i$, and the maximum number of containers that the freight train can carry is $W$. All containers for a particular customer must be placed on the same train so that they are delivered at the same time. We wish to find a set of train trips to deliver all containers so as to minimize the total length of all trips.

In the bin packing setting, various temperature sensitive products are shipped by sea from southeast Asia. Each product has a weight (in metric tons) and a maximal temperature at which it may be stored (there is no minimum temperature limit). Each ship can carry at most $W$ tons. Since the route is fixed, the cost of operating the ship is determined by the ambient temperature in the cargo area. The lower the temperature, the higher the cost. The shipping company is interested in keeping the cost of operations as low as possible while keeping the temperatures low enough so none of the products on board a ship are damaged. The goal is to find a packing of the products into ships that minimizes the overall cost of operation.

Bin-packing is the special case of the train delivery problem where all the $p_i$'s are equal. It is well known [18] that bin-packing is strongly NP-hard and does not admit a polynomial time approximation algorithm better than 3/2 unless P=NP, hence those negative results also hold for the train delivery problem. There are, however, algorithms that achieve an approximation factor of $1 + \epsilon$ for bin-packing for any $\epsilon > 0$, provided that the cost of an optimal solution is at least $1/\epsilon$ (i.e. at least $1/\epsilon$ bins are necessary). Such algorithms are called asymptotic polynomial time approximation schemes (APTAS).

**Our Results.** We give the first approximation schemes for the general train delivery problem. The problem does not admit an asymptotic approximation scheme in the usual sense as the cost of the solution is determined by the positions $p_i$. Thus any instance can be scaled so that the optimal solution has arbitrarily large cost without changing the solution itself. To define a notion of asymptotic approximation scheme for our problem we restrict the ratio of the optimal solution and the maximal position. In other words, scale the input so that $\max_i p_i = 1$; then we are in the asymptotic regime if the cost of the scaled input is $\Omega(1/\epsilon^6)$.

**Theorem 1.** *Given an instance of the train delivery problem such that* $\max_i p_i = O(\epsilon^6)OPT$*, Algorithm 1 outputs a solution of cost* $(1 + O(\epsilon))OPT$ *in time* $O(n \log(n)) + \log(n)(1/\epsilon)^{O(1/\epsilon)}$.[1]

[1] An alternative version of our algorithm uses the less severe asymptotic assumption, $\max_i p_i = O(\epsilon)\mathrm{OPT}$, but runs in time $n^{(1/\epsilon)^{O(1/\epsilon)}}$.

Alternatively, we give a polynomial time approximation scheme (PTAS) for the case where $W = poly(n)$ (or where $W$ is specified in unary). Note that bin-packing is still NP-hard for such instances. We note that, unless P=NP, we cannot hope to achieve a PTAS when the conditions of Theorem 2 do not hold. A PTAS that also works when $W > poly(n)$ would give us a polynomial time algorithm, rather than a pseudo polynomial time algorithm, for deciding the NP-hard *partition* problem[2].

**Theorem 2.** *Given an instance of the train delivery problem such that* $W = poly(n)$*, Algorithm 5 outputs a solution of cost* $(1+O(\epsilon))OPT$ *in time* $W^{e^{O(1/\epsilon)}} + O(n\log(n)) + \log(n)(1/\epsilon)^{O(1/\epsilon)}$.

**Related work.** Train delivery in its formulation as the problem of minimizing the makespan on a single batch machine with non-identical job sizes has been extensively studied in the past two decades, and the present paper gives the first asymptotic PTAS for the general version of the problem. To the best of our knowledge, Uzsoy [32] was the first to consider the problem. He proved that it is NP-hard and presented a few heuristics that were evaluated empirically. Many others have considered the problem since. Various heuristics are given in [2,13,12,31] to name just a few, while application of meta-heuristics to the problem were studied in [29,25,23]. The work of Zhang et al. [33] proves approximation ratios for some heuristics, with 7/4 being the best, while showing that others may perform arbitrarily bad. Zhang and Cao [34] also gave an APTAS for the symmetric setting where they assumed that $p_i = w_i$ for all $i$.

The techniques we use in this paper draw on those used in the literature for the bin-packing problem and the vehicle routing problem. Both problems are extensively studied in the literature. Rather than a comprehensive survey, we focus on the previous techniques we extend in this paper. Bin-packing is one of the problems originally shown to be strongly NP hard by Garey and Johnson [18]. Fernandez de la Vega and Lueker [16] obtained the first APTAS. Their algorithm handles big and small demands separately and uses the fact that small demands can be ignored initially and added greedily to any near optimal solution of just the big demands. The big demands are rounded and a linear program is used to find a near optimal solution for them in time polynomial in $n$ and exponential in $1/\epsilon$. Subsequently, Karmarkar and Karp [22] gave an asymptotic *fully* polynomial approximation scheme (AFPTAS) using the same framework and *efficiently* solving and rounding the LP relaxation of the problem. Their running time depends polynomially on $1/\epsilon$, rather than exponentially. Many variants of bin-packing have been considered, (see [10] for a survey). In multi-dimensional bin-packing (See [7,24,4,9] and references therein), the bins are multi-dimensional, but the cost of each bin is still a fixed constant. In variable-size bin-packing (See [17,30,14]) bins of several different sizes are available and

[2] Partition: Decide if set of integers $S$ can be partitioned into sets $S_1$, $S_2$ s.t the sum of $S_1$ and $S_2$ are equal.

the cost of each bin is proportional to its size. In bin-packing with "general cost structure"(See [15,27]), the cost of a bin is a non-decreasing concave function of the number of elements packed in the bin.

There are AFPTAS for all of these variants and all of those we are aware of handle big and small items separately and use rounding of the big items. None of these variants, however, captures the problem we consider. In some variants, in contrast to the classical bin-packing problem, the small demands can make an important contribution to the cost of the solution and must be handled more carefully. The results differ substantially on how much consideration is given to small items while computing a solution of the big items. In the case of bin-packing with "general cost structure" the small items are considered fluid and can be split up arbitrarily among bins [14]. This approach was introduced in the bin-covering problem [11].

The VRP is another widely studied problem. The train delivery problem is the 1-dimensional version of VRP with unsplittable demands [20,6,5] where a set of customers, each with its own demand $w_i$ must be served by vehicles which depart from and return to a single depot. Each vehicle may serve at most $W$ total demand and each customer must be served by a single vehicle. The objective is to minimize the total distance travelled by all vehicles[3]. In the 1-dimensional version the depot is located at the origin and the position of customer $i$ is given by $p_i$. We are not aware of prior work that specifically considers the 1-dimensional setting. For the metric case, Haimovich, Rinnooy Kan, and Stougie give a constant factor approximation [20]. Bramel et al. study the problem on the Euclidean plane where customer demands are drawn i.i.d from any distribution [6]. Labbé et al. [26] give a 2-approximation for the problem on a tree. Additional heuristics that extend their approach were given in [28,8].

For the splittable case, where customers may be served by multiple vehicles, Haimovich and Rinnooy Kan gave a PTAS for the Euclidean plane when $W = O(\log \log n)$ [19]. Their approximation scheme partitions the customers into disjoint instances ($far$ and $close$) based on the distance from the depot. The $far$ instance is small enough so that it can be solved exactly by brute force, but sufficiently large, so that the error incurred by solving the instances independently is controlled. The $close$ instance is "close" enough to the depot such that for small values of $W$ a constant factor algorithm (that they also present) finds a near optimal solution for $close$. Recently, Adamaszek, Czumaj, and Lingas extended this to $W \leq 2^{\log^{\delta} n}$ (where $\delta$ a function of $\epsilon$) [1]. They use a shifting technique [3,21] to partition the instance into disjoint regions, and solve the problem in each region independently.

**Preliminaries.** For the remainder of the paper we use the language of the vehicle routing problem: We refer to tours (rather than sets), customers (rather than items), locations (rather than positions) and demands (rather than weights). We assume the depot is at the origin and that the instance is preprocessed:

[3] The VRP objective is 2 times the objective of the train delivery problem.

**Definition 1.** *(Preprocessed) An instance is preprocessed if:*
- *No demand is has location* $p_i \leq \epsilon \cdot p_{\max}/n$*, where* $p_{\max} = \max_i p_i$*.*
- *All customers are located to the right of the depot.*

If there are demands located closer than $\epsilon \cdot p_{\max}/n$, serve them each with a separate tour. The overall cost for this at most $\epsilon p_{\max} \leq \epsilon \mathrm{OPT}$ as any solution must have cost at least $p_{\max}$. If there are customers on the right and left of the depot, we can solve each side separately, as they are analogous to one another, and return the union of the two solutions.

## 2 Algorithm for Theorem 1

We summarize the main steps of Algorithm 1 and provide details in the following subsections. Its correctness and running time follow from the lemmas below. See the full version of the paper (available on the authors' websites) for all proofs.

**Algorithm 1.** Asymptotic PTAS for train delivery

**Input:** A preprocessed input with demands $(p_i, w_i)_{1 \leq i \leq n}$ and train capacity $W$
**Precondition:** $\max_i p_i \leq \epsilon^6 \mathrm{OPT}$

1: Round the input using Algorithm 2.
2: **for** $1 \leq i \leq 1/\epsilon$ **do**
3: Let $P_i$ be the $i$-th partition into regions (as in Definition 3).
4: **for** each non-empty region $R$ of $P_i$ **do**
5: Get a *relaxed* solution covering all demands in $R$ treating small demands as fluid using Algorithm 3.
6: Extend the *relaxed* solution to cover small demands feasibly using Algorithm 4.
7: Let $\mathrm{Best}(P_i)$ be the union of the solutions found for each region $R \in P_i$.

**Output:** $\min_i \mathrm{Best}(P_i)$, the minimum cost solution found.

**Rounding.** The number of locations is reduced by rounding each location up to the next integer power of $(1+\epsilon)$. We call a demand $w_i$ big if $w_i \geq \epsilon W$ and small otherwise. The rounding technique from classical bin packing is use to reduce the number of distinct big demand sizes at each location.

**Partitioning into regions.** We partition the demands into disjoint regions based on their distance from the depot (Definition 3) so that each region has only a constant number of locations containing demands. We solve the problem approximately within each region independently. A shifting argument shows that if we do this for a few different partitions, the union of the approximate solutions in at least one of the partitions yields a near optimal solution.

**Solving within a region.** Within each region, we treat big and small demands differently. We relax the unsplittable constraint for small demands and allow them to be split up between multiple tours. The relaxed problem is solved by a linear program that considers the big demands individually and the total small

demand weight at each location in the region. Since each region contains just a constant number of locations and each location contains a constant number of distinct big demand sizes, the total number of distinct big demands in $R$ is constant. This allows us to solve the relaxed problem in constant time.

We construct a feasible solution from the relaxed solution by adding the small demands greedily, and prove that the solution output has cost at most $(1+2\epsilon)$ $\mathrm{OPT}(R) + O((1/\epsilon)^5)p_R$, where $\mathrm{OPT}(R)$ denotes the optimal solution of region $R$ and $p_R$ is the location furthest from the depot in $R$ (Lemma 5). Our definition of regions ensures that $p_R$ decreases geometrically as the regions get closer to the depot. Thus the sum of $p_R$ over all regions is $O(p_{\max})$, where $p_{\max}$ is the location of the furthest demand in the instance.

Our assumption $p_{\max} \leq O(\epsilon^6)\mathrm{OPT}$ ensures that the additive cost incurred by the greedy extension (the $p_R$ terms) is within the desired approximation factor.

### 2.1 Rounding

Algorithm 2 outputs a rounded instance satisfying Lemma 1. Its proof extends the bin packing rounding analysis of [16].

**Algorithm 2.** Rounding Instance

**Input:** train capacity $W$, demands $(p_i, w_i)_{1 \leq i \leq n}$

1: Round each $p_i$ up to the smallest integer power of $(1+\epsilon)$.
2: Partition demands $(w_i)_i$ into *big* $(\geq W\epsilon)$ and *small* $(< W\epsilon)$.
**Rounding big demands:**
3: **for** each location $\ell$ s.t. $n_\ell$, the number of big demands at $\ell$, is at least $1/\epsilon^2$ **do**
4: Go through those big demands in decreasing order to partition them into $\epsilon^{-2}$ groups such that each group (except possibly one) has cardinality exactly $\lfloor n_\ell \epsilon^2 \rfloor$.
5: **for** each group $g$ **do**
6: Round every demand in $g$ up to the maximum demand in $g$.

**Output:** rounded instance $I'$ of the train delivery problem

**Lemma 1.** *Given an instance $I$, Algorithm 2 outputs an instance $I'$ such that:*

- *Each $p_i$ has the form $(1+\epsilon)^k$ for some (non-negative) integer $k$.*
- *Each location has at most $1/\epsilon^2 + 1$ distinct big demands.*
- *Any feasible solution for $I'$ is also feasible for $I$.*
- *$OPT(I') \leq (1 + O(\epsilon))OPT(I)$.*

### 2.2 Partitioning into Regions

We say that a tour has small expanse if it covers points in a bounded region. Lemma 2 shows that a near optimal solution can be obtained using only tours with this simple structure.

**Definition 2.** *A tour that covers only points between locations $p$ and $p'$, $p \leq p'$, has* expanse $p'/p$. *A* small *expanse tour has expanse at most* $1/\epsilon$.

**Lemma 2.** *There exists a small expanse tour solution of cost $\leq (1+2\epsilon)\,OPT$.*

We partition instance $I'$ into regions, where each region has large expanse. Intuitively, as the expanse of the region is large only a few tours of the optimal small expanse solution will cover points in more than one region. Thus we can find a near optimal solution by solving each region independently.

**Definition 3.** *Let $I'$ be a rounded instance of the train delivery problem and $p_{\max} = \max_i p_i$. A* block*, defined by an integer $i$, is the set of demands with locations in $(p_{\max}\epsilon^{i+1}, p_{\max}\epsilon^i]$. A* region *is a group of $\leq 1/\epsilon$ consecutive blocks.*

*For $0 \leq j < 1/\epsilon$, let $P_j$ denote the partition of $I$ into regions, one initial region $(\epsilon^j p_{\max}, p_{\max}]$ and the other regions $(\epsilon^j p_{\max}\epsilon^{(i+1)/\epsilon}, \epsilon^j p_{\max}\epsilon^{i/\epsilon}]$ for $i \geq 0$.*

Note that there are $1/\epsilon$ possible ways to partition $I'$ into regions. We use a *shifting* technique similar to that of Baker [3] and Hochbaum and Maass [21] and an averaging argument to show Lemma 3, which states that at least one partition yields a near optimal solution for $I'$.

**Lemma 3.** *There exists a partition $P_j$ s.t. $\sum_{R \in P_j} OPT(R) \leq (1 + O(\epsilon))OPT$.*

### 2.3 Solving the Relaxed Problem in a Region

**Definition 4.** *(Big demand type) A big demand* type *is a pair $(p, b)$ where $p$ is the location of a big demand and $b$ is one of the at most $1/\epsilon^2$ big demand (rounded) sizes at $p$. $n(d)$ denotes the number of demands of type $d$ in region $R$.*

The configuration of a tour roughly describes which points it will cover: for each occurrence of demand type $(p, b)$ in its multiset the tour covers one of the demands from location $p$ with value $b$.

**Definition 5.** *(Configuration) A* configuration *$f$ of a tour in $R$ consists of $r_f$, the furthest location of the tour, and a multiset $M_f$ of big demand types, each with location at most $r_f$, whose values sum up to at most $W$.*

Let $c_f$ denote the remaining capacity of a tour with configuration $f$ (i.e., $c_f = W - \sum_{(p,b)\in M_f} b$). For any big demand type $d$, let $n_f(d)$ denote the multiplicity of $d$ in $M_f$. Let $S$ be the set of small demands in a region $R$. We relax our problem by removing the unsplittable constraint on only the small demands. Algorithm 3 rounds the solution of the linear program below to obtain, by Lemma 4, a near optimal solution of the relaxed problem in constant time. The linear program has one variable $x_f$ for each possible tour configuration $f$. The objective selects a minimum cost multiset of tour configurations such that two constraints are satisfied: Constraint 1 ensures that all big demand types are covered by the selected tour configurations and constraint 2 ensures that for each location $p$, the small demands further right than $p$ can be covered with the remaining capacities of the tour configurations that extend to the right of $p$.

$$\begin{aligned}
&\min \textstyle\sum_{f\in\mathcal{F}} r_f x_f && s.t. && \\
&\textstyle\sum_{f\in\mathcal{F}} x_f n_f(d) \geq n(d) && \text{for all demand types } d && (1)\\
&\textstyle\sum_{f:r_f\geq p} c_f x_f \geq \sum_{(p_i,w_i)\in S, p_i\geq p} w_i && \text{for all locations } p && (2)\\
&x_f \geq 0 &&&&
\end{aligned}$$

**Algorithm 3.** Solve Relaxed Region

**Input:** $R$ a region of $I'$, $S$ the set of small demands in $R$.

1: Let $\mathcal{D}$ be the set of big demand types (Definition 4) and $\mathcal{F}$ be the set of tour configurations for region $R$ (Definition 5).
2: Let $x^* = (x^*_f)_{f\in\mathcal{F}}$ be a basic solution of the linear program (Eq. 1-2).
3: Let $\bar{x}_f = \lceil x^*_f \rceil$ for each $f \in \mathcal{F}$.
4: Cover the big demand types in $\mathcal{D}$ with tours specified by the $(\bar{x}_f)_{f\in\mathcal{F}}$. (Some tours may only be assigned a partial list of the big demands listed in its configuration.)

**Output:** The resulting set of tours covering $\mathcal{D}$.

**Lemma 4.** *Let $p_R$ denote the maximum location in region $R$, $OPT(R)$ the cost of the optimal solution to the (unrelaxed) problem and $T$ the set of tours output by Algorithm 3. $T$ is a solution to the relaxed problem that covers all demands in $R$ such that the big demands are unsplittable and the small demands are allowed to be split. $Cost(T) \leq OPT(R) + p_R((1/\epsilon)^2 \log(1/\epsilon))(2 + 1/\epsilon^2)$. Algorithm 3 outputs $T$ in time $(1/\epsilon)^{O(1/\epsilon)}$.*

### 2.4 Extending a Relaxed Solution of the Region

Let $(r_t, c_t)_t$ denote the set of tours output by Algorithm 3 where $r_t$ denotes the maximum location and $c_t$ the remaining capacity of tour $t$ after it has covered its big demands. Algorithm 4 takes the list of tours $(r_t, c_t)_t$ and extends it to cover the small demands of $R$ in a feasible way (i.e., without splitting any of them). Lemma 5 shows that the greedy extension is a near optimal.

**Algorithm 4.** Greedy Extension

**Input:** small demands $(p_i, w_i)_i$ and a list $T$ of tours $(r_t, c_t)_t$. $r_t$ is the furthest location of tour $t$ and $c_t$ is its remaining capacity.

1: **for** each small demand $(p_i, w_i)$ by order of decreasing $p_i$ **do**
2: **if** there is a tour $t$ with $r_t \geq p_i$ and $c_t \geq w_i$ **then**
3: cover $(p_i, w_i)$ with $t$ and set $c_t := c_t - w_i$
4: **else**
5: add a new tour $t$ with $c_t = W$ and $r_t = p_i$
6: cover $(p_i, w_i)$ with $t$ and set $c_t := c_t - w_i$

**Output:** the resulting tours.

**Lemma 5.** *The output of Algorithm 4 $G$ has $cost(G) \leq (1 + 2\epsilon)cost(T) + 2p_R$.*

## 3 Algorithm for Theorem 2

We summarize the main steps of Algorithm 5 and provide details in the following subsections. Its correctness and running time follow from the lemmas appearing in the subsections. See the full version of paper for all proofs.

**Algorithm 5.** PTAS for the train delivery problem when $W \leq poly(n)$

**Input:** train capacity $W$, demands $(p_i, w_i)_{1 \leq i \leq n}$
**Precondition:** $W \leq poly(n)$.

1: Partition the instance into *close* and *far* using Algorithm 6
2: Find OPT($far$) using Algorithm 7 and Best(*close*) using Algorithm 1.

**Output:** Best(*close*) $\cup$ OPT($far$), as the solution for the whole instance.

**Partition into *close* and $far$ instances.** We index the demands in decreasing order of location, identify a demand $i_c$ and partition the instance into *close* and $far$, where $far$ contains the demands with indices less than $i_c$ and *close* the demands with indices greater than $i_c$.

**Optimal solution of $far$.** The partition is such that $far$ contains $O(W)$ total demand. Thus an optimal solution of $far$ uses a constant number of tours (Lemma 9). This allows us to enumerate, in polynomial time, all possible such solutions. Using a generalization of the well-known dynamic program for subset sum, we can determine in polynomial time whether a proposed solution is feasible or not, hence an optimal algorithm for $far$.

**Overall solution.** The solution is the union of the solutions for *close* and $far$. We use Algorithm 1 to find a near optimal solution to *close*. It is crucial that, on the one hand, $far$ does not contain too much demand, so it can be solved efficiently. On the other hand, $far$ contains enough demand so that the condition of Theorem 1 holds for *close*, namely that $p_{i_c} = O(\epsilon^6)\text{OPT}$, where $p_{i_c}$ denotes the furthest location in *close*.

### 3.1 Partitioning into *Close* and *Far*

Algorithm 6 partitions the instance. Lemma 6 proves that *close* and $far$ can be solved separately at small additional cost. Its proof critically uses the definition of $i_c$. Lemma 7 also follows by choice of $i_c$.

**Algorithm 6.** Partition Close and Far

**Input:** train capacity $W$, demands $(p_i, w_i)_{1 \leq i \leq n}$ s.t. $p_1 \geq \cdots \geq p_n$

1: Let $i_c$ be the smallest index such that
   - $\sum_{j \leq i_c} w_j \geq W/\epsilon^6$
   - $p_{i_c} \sum_{j \leq i_c} w_j \leq \epsilon \sum_{j=1}^{n} w_j p_j$.

   If no such $i_c$ exist, set $i_c := n$.
2: ***Far:*** Let the $far$ instance consist of the demands indexed by $1, \ldots, i_c$.
3: ***Close:*** Let the *close* consist of the remaining demands $i_c + 1, \ldots n$.

**Output:** instances $far$ and *close*

**Lemma 6.** *Algorithm 6 returns two instances* far *and* close *s.t.* $OPT(close) + OPT(far) \leq (1 + O(\epsilon))OPT$.

**Lemma 7.** $OPT > 2p_{i_c}/\epsilon^6$.

### 3.2 Solving the *Far* Instance

The following combinatorial lemma is an extension of Haimovich and Rinnooy Kan's [19] analysis to the case of unsplittable demands. It bounds the total demand in $far$ by $O(W)$ by bounding the number of demands that violate the requirement $p_{i_c} \sum_{j \le i_c} w_j \le \epsilon \sum_{j=1}^{n} w_j p_j$.

**Lemma 8.** *Let $i_c$ be as in Algorithm 6. Then $\sum_{j \le i_c} w_j = \exp(O(1/\epsilon))W$.*

Lemma 9 implies that OPT($far$) uses a constant, $c_{far}$, number of tours. Then we can solve $far$ using dynamic programming.

**Lemma 9.** *OPT uses at most $\lceil 2D/W \rceil$ tours, where $D$ the total demand.*

**Definition 6.** *(Far Configuration) Let $c_{far}$ denote the maximum number of tours OPT($far$) may use. A* configuration *for $far$ consists of:*
- *Locations $r_1 \ge r_2 \ldots \ge r_{c_{far}}$ s.t. $r_i$ is the maximum location of tour $i$.*
- *For every $i \in [1, c_{far}]$, a list $S^i$ of $i$ numbers $S^i = \{s_1^i, \ldots s_i^i\}$.*
  *The* cost *of the configuration is $\sum_{j \le c_{far}} r_j$.*

For $i = 1, 2, \ldots c_{far} - 1$, define an interval $I_i = (r_{i+1}, r_i]$. Let $I_{c_{far}}$ be the interval $[p_{i_c}, r_{c_{far}}]$. The demands in $I_i$ can only be covered by the $i$ tours whose maximum location is at least $r_i$. The list $S^i$ specifies the total demand from interval $I_i$ that is assigned to each of these tours. $S^i$ does not directly describe how to partition the demands among the $i$ tours. Finding a set of partitions that is consistent with a configuration, or finding out that no such set of partitions exists, is done in $W^{e^{O(1/\epsilon)}}$ time (i.e., polynomial in $n$ assuming $W \le poly(n)$)) using a trivial extension of the dynamic program for the subset sum problem (omitted). Algorithm 7 solves $far$ by iterating all configurations, checking feasibility, and returning the minimum cost feasible configuration.

**Algorithm 7.** Solving the $far$ instance

**Input**: $far$ demands $(p_i, w_i)_i$ with $\sum_i w_i = We^{O(1/\epsilon)}$

```
for each configuration f of far as given in definition 6 do
    for each tour j ≤ c_far do
        if Σ_{i≤c_far} s_j^i > W then
            Mark f infeasible. {capacity of tour is exceeded}
    for each interval i ≤ c_far, with total demand dem(I_i) do
        if Extended DP of subset sum cannot partition dem(I_i) into s_1^i, ..., s_i^i then
            Mark f infeasible. {demands cannot be partitioned}
```

**Output:** Solution realized by the minimum cost configuration not marked infeasible.

**Lemma 10.** *Given an instance of $far$ with demand $We^{O(1/\epsilon)}$ Algorithm 7 finds the optimal solution of $far$ in time $W^{e^{O(1/\epsilon)}}$, which is polynomial in $n$ and $W$.*

## References


1. Adamaszek, A., Czumaj, A., Lingas, A.: PTAS for k-tour cover problem on the plane for moderately large values of k. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 994–1003. Springer, Heidelberg (2009)
2. Azizoglu, M., Webster, S.: Scheduling a batch processing machine with non-identical job sizes. Intern. J. Prod. Res. 38, 2173–2184 (2000)
3. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. ACM 41(1), 153–180 (1994)
4. Bansal, N., Correa, J.R., Kenyon, C., Sviridenko, M.: Bin packing in multiple dimensions: inapproximability results and approximation schemes. Math. Oper. Res. 31(1), 31–49 (2006)
5. Bertsimas, D.J., Simchi-Levi, D.: A new generation of vehicle routing research: robust algorithms, addressing uncertainty. Oper. Res. 44(2), 286–304 (1996)
6. Bramel, J., Coffman Jr., E.G., Schor, P.W., Simchi-Levi, D.: Probabilistic analysis of the capacitated vehicle routing problem with unsplit demands. Oper. Res. 40, 1095–1106 (1992)
7. Caprara, A.: Packing 2-dimensional bins in harmony. In: FOCS, Washington, DC, USA, pp. 490–499. IEEE, Los Alamitos (2002)
8. Chandran, B., Raghavan, S.: The vehicle routing problem: latest advances and new challenges, vol. 43, pp. 239–261. Springer, US (2008)
9. Chekuri, C., Khanna, S.: On multi-dimensional packing problems. In: SODA, pp. 185–194. SIAM, Philadelphia (1999)
10. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: a survey, pp. 46–93. PWS Pub. Co., Boston (1997)
11. Csirik, J., Johnson, D.S., Kenyon, C.: Better approximation algorithms for bin covering. In: SODA, PA, USA, pp. 557–566. SIAM, Philadelphia (2001)
12. Dupont, L., Dhaenens-Flipo, C.: Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. Comp. and Op. Res. 29(7), 807–819 (2002)
13. Dupont, L., Ghazvini, F.J.: Minimizing makespan on a single batch processing machine with non-identical job sizes. Eur. J. Auto. Sys. 32, 431–440 (1998)
14. Epstein, L., Levin, A.: An APTAS for generalized cost variable-sized bin packing. SIAM J. Comp. 38, 411–428 (2008)
15. Epstein, L., Levin, A.: Bin packing with general cost structures. Mathematical Programming, 1–37 (2010)
16. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1 + \epsilon$ in linear time. Combinatorica 1(4), 349–355 (1981)
17. Friesen, D.K., Langston, M.A.: Variable sized bin packing. SIAM J. Comput. 15(1), 222–230 (1986)
18. Garey, M.R., Johnson, D.S.: "strong" NP-completeness results: motivation, examples, and implications. J. ACM 25(3), 499–508 (1978)
19. Haimovich, M., Rinnooy Kan, A.H.G.: Bounds and heuristics for capacitated routing problems. Math. of Op. Res. 10(4), 527–542 (1985)
20. Haimovich, M., Rinnooy Kan, A.H.G., Stougie, L.: Analysis of heuristics for vehicle routing problems. Vehicle Routing: Methods and Studies. Manag. Sci. Systems 16, 47–61 (1988)
21. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and vlsi. J. ACM 32(1), 130–136 (1985)

22. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: FOCS, pp. 312–320 (1982)
23. Kashan, A.H., Karimi, B., Jolai, F.: Effective hybrid genetic algorithm for minimizing makespan on a single-batch-processing machine with non-identical job sizes. Intern. J. Prod. Res. 44, 2337–2360 (2006)
24. Kenyon, C., Rémila, E.: A near-optimal solution to a two-dimensional cutting stock problem. Math. Oper. Res. 25(4), 645–656 (2000)
25. Koh, S.-G., Koo, P.-H., Kim, D.-C., Hur, W.-S.: Scheduling a single batch processing machine with arbitrary job sizes and incompatible job families. Int. J. Prod. Econ. 98, 81–96 (2005)
26. Labbé, M., Laporte, G., Mercure, H.: Capacitated vehicle routing on trees. Op. Res. 39(4), 616–622 (1991)
27. Li, C.L., Chen, Z.L.: Bin-packing problem with concave costs of bin utilization. Nav. Res. Log. 53(4), 298–308 (2006)
28. Mbaraga, P., Langevin, A., Laporte, G.: Two exact algorithms for the vehicle routing problem on trees. Nav. Res. Log. 46, 75–89 (1999)
29. Melouk, S., Damodaran, P., Chang, P.-Y.: Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. Inter. J. of Prod. Econ., 141–147 (2004)
30. Murgolo, F.D.: An efficient approximation scheme for variable-sized bin packing. SIAM J. Comput. 16(1), 149–161 (1987)
31. Parsa, N.R., Karimi, B., Kashan, A.H.: A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. Com. Op. Res. 37(10), 1720–1730 (2010)
32. Uzsoy, R.: Scheduling a single batch processing machine with non-identical job sizes. Int. J. of Prod. Res. 32, 1615–1635 (1994)
33. Zhang, G., Cai, X., Lee, C.-Y., Wong, C.K.: Minimizing makespan on a single batch processing machine with nonidentical job sizes. Nav. Res. Log. 48, 226–240 (2001)
34. Zhang, Y., Cao, Z.: An asymptotic PTAS for batch scheduling with nonidentical job sizes to minimize makespan. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) COCOA. LNCS, vol. 4616, pp. 44–51. Springer, Heidelberg (2007)

# An FPTAS for Flows over Time with Aggregate Arc Capacities*

Daniel Dressler and Martin Skutella

Institute for Mathematics, TU Berlin, Str. des 17. Juni 136, 10623 Berlin, Germany
{dressler,skutella}@math.tu-berlin.de

**Abstract.** We study flows over time in networks with transit times on the arcs. Transit times describe how long it takes to traverse an arc. A flow over time specifies for each arc a time-dependent flow rate that must always be bounded by the arc's capacity. Only recently, Melkonian introduced an alternative model where so-called *bridge capacities* bound the total amount of flow traveling along an arc, at any point of time.

The contribution of this paper is twofold. Firstly, we introduce a common generalization of both the classical flow over time model and Melkonian's model. Secondly, we present a non-trivial extension of an FPTAS by Fleischer and Skutella to our new flow model. Prior to this, no approximation algorithm was known for Melkonian's model.

**Keywords:** network flow, dynamic flow, arc capacity, approximation algorithm.

## 1 Introduction

Network flows are a key concept in combinatorial optimization and their modeling abilities are fundamental in logistics. Already Ford and Fulkerson introduced *flows over time* (or *dynamic flows*) that model flow moving through a network *over time* [2]. (Classical flows without this temporal dimension are often called static flows.) In a flow over time model, each arc has a transit time and flow entering the tail of an arc leaves its head only after the transit time has passed. Flows over time are given by a function for each arc that specifies the rate at which flow enters the arc at each point in time. Flow originates at sources and leaves the network at sinks. Traditionally, capacities limit the rate at which flow enters an arc. In a street network, for example, these capacities can model the number of lanes of a road.

Melkonian [7] introduced an alternative model with *bridge capacities*. At any point in time, bridge capacities limit the amount of flow that has entered but not yet left an arc. This can be used to model a load limit on a bridge that supports fewer cars and trucks than the number of lanes might suggest.

In this paper we generalize bridge capacities such that they bound the total flow entering an arc within a sliding time window. The fixed length of the window

* Supported by the DFG research center MATHEON in Berlin and the Federal Ministry for Education and Research (BMBF) under grant 03SKPAI6 "Advest".

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 106–117, 2011.

is independent from the transit time of an arc. In particular, this allows mixing bridge capacities (time window as long as transit time) with flow rate capacities (very short/infinitesimal window), and arcs that may be used only by a certain amount of flow in total (infinite window) in the same network. If necessary, one can even combine several of such capacity constraints on a single arc.

We refer to this general type of capacity constraints as *aggregate capacities*. In this general setting, we study the Dynamic Transshipment Problem which asks whether there is a flow over time that balances the supplies of the sources with the demands of the sinks. Besides the aggregate arc capacities, the flow is restricted by a given time horizon.

*Related Work.* Already classical flows over time tend to be algorithmically harder than static flows, beyond merely containing the static problem as a special case. For traditional flow rate capacities, the Dynamic Transshipment Problem for a network with a single source and a single sink can be solved efficiently by computing a minimum cost static circulation [2]. Hoppe and Tardos [3] give a polynomial-time algorithm for the Dynamic Transshipment Problem with multiple sources and sinks. As soon as arc costs come into play, even the case with a single source and a single sink restricted to series-parallel graphs becomes NP-complete, as shown by Klinz and Woeginger [5].

Fleischer and Skutella [1] present a simple 2-approximation algorithm for a general class of problems including the Quickest Transshipment Problem, which is the optimization version of the Dynamic Transshipment Problem, asking for the minimum feasible time horizon. This result can be generalized to the setting with aggregate arc capacities in a straightforward way. Moreover, a fully polynomial time approximation scheme for these problems is given in [1].

For bridge capacities, Melkonian [7] proves that already the Dynamic Transshipment Problem with a single source and sink is weakly NP-complete, but can be decided by solving a linear program of pseudo-polynomial size. Melkonian also suggests a heuristic approach where the capacity on each arc is replaced by traditional capacities equal to the average sustainable flow on the arc. Finally, he mentions networks with mixed capacities as an interesting research direction, which we pursue with our model.

A related capacity model has earlier been proposed by Klinz and Woeginger [4]: They study *dedicated arcs* that are entirely blocked as long as even a small amount of flow is traveling along them. (As for bridge capacities, the duration of the block and the transit time are identical.) All their flows are discrete, meaning that flow travels in whole packets sent once per time step as opposed to continuous flow rates. They also restrict to integral flow functions, which often prohibits the use of linear programming techniques. They derive interesting complexity results for their setting: For instance, even for a fixed time horizon of 3, one of their variants of the Dynamic Transshipment Problem is NP-hard. They also translate a complexity result of Papadimitriou, Serafini, and Yannakakis [8] into the language of flows over time. This implies that the Dynamic Transshipment Problem for integral flows and dedicated arcs with unit capacities is strongly NP-complete.

Köhler and Skutella [6] study flows over time with load-dependent transit times. In their model, the speed at which flow travels along an arc always depends on the current amount of flow (load) on that arc. The model of Melkonian can be considered as a special case by letting the speed on an arc be constant up to its capacity and infinite if the load exceeds the capacity.

*Our Contribution.* In Section 2, we introduce the flow model with aggregate arc capacities which is a direct generalization of Melkonian's bridge capacities. For this new model, we discuss important properties of flows over time, namely integrality and whether storage at vertices can improve the flow value, and show that forbidding storage or forcing integrality qualitatively restricts the set of solutions. We argue that our model also generalizes traditional flow rate capacities.

Our main contribution, a fully polynomial time approximation scheme for the Dynamic Transshipment Problem, is presented in Section 3. If there is a feasible flow for the original instance, for all $\varepsilon > 0$, we can compute a flow over time satisfying the same set of demands, but violating the capacities and the time bound by a factor of at most $1 + \varepsilon$.

We mention, that the presented FPTAS can be generalized in a straightforward way to the setting with multiple commodities and costs. Due to space limitations, however, we restrict our presentation to the Dynamic Transshipment Problem in this extended abstract and omit some details and proofs.

## 2 Aggregate Arc Capacities

We consider finite directed graphs $G = (V, A)$, possibly with loops and parallel arcs. For $a \in A$, let $\mathrm{tail}(a)$ denote the start vertex and $\mathrm{head}(a)$ the end vertex of $a$. For $v \in V$, we use $A_{\mathrm{in}}(v) := \{a \in A : \mathrm{head}(a) = v\}$ and $A_{\mathrm{out}}(v) := \{a \in A : \mathrm{tail}(a) = v\}$ for the set of arcs entering and leaving $v$, respectively.

Each arc $a \in A$ has a transit time $\tau_a \in \mathbb{R}_{\geq 0}$, a capacity $u_a \in \mathbb{R}_{\geq 0}$ and a length of the sliding window $\ell_a \in \mathbb{R}_{\geq 0}$. Some vertices belong to the sources $S^+ \subset V$, some to the sinks $S^- \subset V$, and we assume $S^+ \cap S^- = \emptyset$. The elements of the set $S^+ \cup S^-$ of sources and sinks are called *terminals*. We assume w. l. o. g. that sources have no incoming arcs and sinks no outgoing arcs. This can be achieved by adding a new vertex for each source $s^+$ with a single outgoing arc that points towards $s^+$ and making it the new source (and similarly for sinks). The tuple $\mathcal{N} := (V, A, S^+, S^-, \tau, u, \ell)$ forms a *flow network*.

We can now define a *flow over time* on the network $\mathcal{N}$. For this, consider a function $f : A \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, where each $f(a, \cdot)$ is Lebesgue-measurable. For notational convenience, we extend the domain of the flow functions by setting $f(a, t) = 0$ for all $t < 0$. The value $f(a, t)$ denotes the rate at which flow enters arc $a$ at time $t$. The transit times prescribe that the same flow rate must leave the arc at time $t + \tau_a$. The *balance* of a vertex $v$ is the net flow rate entering $v$ at time $t$

$$\mathrm{bal}_f(v, t) := \sum_{a \in A_{\mathrm{in}}(v)} f(a, t - \tau_a) - \sum_{a \in A_{\mathrm{out}}(v)} f(a, t)$$

and the *excess* is

$$\mathrm{ex}_f(v,t) := \int_0^t \mathrm{bal}_f(v,\theta)\,\mathrm{d}\theta \ ,$$

that is, the net amount of flow available at $v$ at time $t$. *Flow conservation* generally demands that the amount of flow traveling through the network should not change, except at the terminals. There are two competing models for flow conservation in flows over time. *Strict flow conservation* demands that all flow entering a non-terminal vertex immediately leaves it again:

$$\mathrm{bal}_f(v,t) = 0 \qquad \forall t \in \mathbb{R}_{\geq 0}\ \forall v \in V \setminus (S^+ \cup S^-) \ .$$

In contrast, *weak flow conservation* allows for storage at non-terminal vertices. That is, an unlimited amount of flow may remain at a vertex arbitrarily long, which we can express as follows:

$$\mathrm{ex}_f(v,t) \geq 0 \qquad \forall t \in \mathbb{R}_{\geq 0}\ \forall v \in V \setminus (S^+ \cup S^-) \ .$$

Since we assume that sources have no incoming arcs, their excess will always be non-positive. Similarly, the excess of a sink will always be non-negative. We do not restrict these vertices any further. If at least weak flow conservation is satisfied, we call the function $f$ a *flow over time* on $\mathcal{N}$, or simply a flow.

A flow has *time horizon* $T \in \mathbb{R}_{\geq 0}$ if the network is "empty" for all $t > T$. In particular, all flow functions must be 0 after time $T$, but this is a too weak requirement. Instead we demand that no flow may be on an arc after time $T$, i. e., $f(a,t) = 0$ for all $t > T - \tau_a$, $a \in A$. Additionally, no flow may be stored at a non-terminal vertex after time $T$, i. e., $\mathrm{ex}_f(v,T) = 0$ for all $v \in V \setminus (S^+ \cup S^-)$.

For a flow with finite time horizon $T$, one can look at how much flow is being sent from the sources to the sinks. This is conveniently given by the excess of the terminals. Thus, a flow satisfies *demands* $d \in \mathbb{R}^V$ if $d_v = \mathrm{ex}_f(v,T)$ for all $v \in V$. In particular, the demands of sources have to be non-positive, the demands of sinks non-negative, and other vertices must have demand 0.

So far, there is no upper limit on $f$. In our setting, the novelty is that *aggregate arc capacities* restrict how much flow can be sent. The total amount of flow entering arc $a$ must be bounded by $u_a$ in every time window of length $\ell_a$. That is, we require

$$\int_t^{t+\ell_a} f(a,\theta)\,\mathrm{d}\theta \leq u_a \qquad \forall t \in \mathbb{R}_{\geq 0}\ \forall a \in A \ .$$

A flow obeying these capacity constraints is called *feasible*. Note that this condition is trivially satisfied if $\ell_a = 0$, and these arcs effectively have no capacity. Traditional flow rate capacities, however, are of the form $f(a,t) \leq u_a$, and this cannot be expressed precisely with aggregate arc capacities. This is no serious drawback, as we will soon see that a small enough $\ell_a$ can essentially model traditional capacities.

We now have everything in place to state our main problem precisely.

Dynamic Transshipment Problem
Input: A flow network $\mathcal{N}$, a time horizon $T$, and a demand vector $d$.
Question: Is there a feasible flow over time in $\mathcal{N}$ with time horizon $T$ satisfying demands $d$?

### 2.1 Discretizations

From now on we assume that $\tau$, $u$, $\ell$, as well as $T$, and $d$ are integral. This can be achieved by scaling, provided that the data was rational to begin with.

When dealing with flows over time algorithmically, most tasks require a representation of the actual flow functions. Clearly, every finite representation must be based on assumptions on the flow function. The most common way is to *discretize* the problem into a finite number of time steps. This usually involves a time-expanded network, which handily reduces many problems involving time to well-studied static flow problems. We essentially do the same, but avoid introducing the full machinery of static flows on time-expanded networks, as this plays a minor role here. Instead we will state our results on piecewise constant functions that change only at multiples of some parameter. For details on time-expanded networks we refer to the survey [9].

We quickly fix some notation for rounding: For $a, \Delta \in \mathbb{R}$, $\Delta > 0$, we use $\lceil a \rceil_\Delta$ to denote $a$ rounded up to the next multiple of $\Delta$. Analogously, $\lfloor a \rfloor_\Delta$ rounds down to multiples of $\Delta$. If $\Delta$ is omitted, we assume $\Delta = 1$.

Now, a function $g : \mathbb{R} \to \mathbb{R}$ is called *$\Delta$-constant* if $g$ restricted to $[i\Delta, (i+1)\Delta)$ is constant for all $i \in \mathbb{Z}$. A flow over time is $\Delta$-constant if each $f(a, \cdot)$ is. Thus, we need only $\lceil T/\Delta \rceil$ values for each arc to describe a $\Delta$-constant flow with time horizon $T$.

When dealing with $\Delta$-constant flows on networks with transit times that are multiples of $\Delta$, we can replace integration by summation in all expressions so far. Another crucial property is that, assuming a finite time horizon, we need only finitely many constraints to ensure that the infinitely many flow conservation and capacity conditions are satisfied. We omit the proof due to space constraints.

**Lemma 1.** *Let $\mathcal{N} = (V, A, S^+, S^-, \tau, u, \ell)$ and assume all $\tau_a$ are multiples of some $\Delta > 0$. Let $f : A \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be a $\Delta$-constant function for all first arguments $a \in A$.*

(i) *Function $f$ is a flow on $\mathcal{N}$ satisfying weak (strict) flow conservation if and only if it satisfies weak (strict) flow conservation for all $t = i\Delta$ with $i \in \mathbb{Z}$.*
(ii) *For $a \in A$, the flow $f$ obeys the aggregate arc capacity on $a$ if and only if the capacity constraints are obeyed for $t_1 = i\Delta$ and for $t_2 = i\Delta - \ell_a$ for all $i \in \mathbb{Z}$.*

Next we show that 1-constant flows are precise enough for integral transit times and capacity windows. More generally, $\Delta$-constant flows suffice for data rounded to multiples of $\Delta$. This lemma is well-known for traditional capacities [1] and requires only little more work in our setting.

**Lemma 2.** *Let $\mathcal{N} = (V, A, S^+, S^-, \tau, u, \ell)$ be a flow network and $f$ a feasible flow over time with time horizon $T \in \mathbb{R}_{\geq 0}$. If the parameters $\tau$ and $\ell$ are multiples of $\Delta > 0$, then there is a $\Delta$-constant feasible flow over time $\bar{f}$ with time horizon $\lceil T \rceil_\Delta$ with the same demands as $f$. If strict flow conservation holds for $f$, then it also holds for $\bar{f}$.*

*Proof (Sketch).* It suffices to average $f$ over each interval $[i\Delta, (i+1)\Delta)$, for $i \in \mathbb{Z}$, to obtain the desired $\Delta$-constant flow:

$$\bar{f}(a,t) := \frac{1}{\Delta} \int_{\lfloor t \rfloor_\Delta}^{\lfloor t \rfloor_\Delta + \Delta} f(a,\theta)\,\mathrm{d}\theta \qquad \forall t \in \mathbb{R}_{\geq 0}\ \forall a \in A\ .$$

The required properties of $\bar{f}$ can easily be verified. □

Since we assume that $\tau$, $\ell$, and $T$ are integral ($u$ and $d$ may be rational), these lemmata show that the Dynamic Transshipment Problem can be formulated as a linear program and thus be solved in time polynomial in the size of the network and $T$. Of course, this only yields a pseudo-polynomial time algorithm.

**Corollary 1.** *For integral $\tau$, $\ell$, and $T$, the* Dynamic Transshipment Problem *can be decided in pseudo-poylnomial time.*

The lemmata also imply that arcs with $\ell_a = 1$ can be used to limit the flow rate to at most $u_a$, modeling traditional capacities. While there may be flows that exceed the traditional capacity momentarily, we can always construct the equivalent 1-constant flow $\bar{f}$ that obeys them.

**Corollary 2.** *For integral $\tau$, $\ell$, and $T$, only flows obeying the traditional capacities $f(a,t) \leq u_a$ for all arcs with $\ell_a = 1$ need to be considered.*

### 2.2 Path Decompositions

One common tool from network flow theory, that we will need, are path decompositions and flows along paths. A *path over time* $P = (A^P, h^P)$ consists of a finite sequence of arcs $A^P = a_1, \ldots, a_q \in A$ and starting times $h^P = h_1, \ldots, h_q \in \mathbb{R}_{\geq 0}$. The path begins at $\mathrm{tail}(a_1)$ and ends at $\mathrm{head}(a_q)$. The arcs in between must form a walk, that is, $\mathrm{head}(a_i) = \mathrm{tail}(a_{i+1})$ for $i = 1, \ldots, q-1$. The time $h_i$ states when the path continues on $a_i$. Thus, $h_{i+1} \geq h_i + \tau_{a_i}$ must hold if a flow unit should travel along $P$. This is therefore required of all paths over time.

In general, we want to send a certain time-dependent flow rate along a path: Let $x_P : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be a Lebesgue-measurable function. Then we call $x_P$ the inflow rate into $P$ and it yields a flow $f_P$ satisfying weak flow conservation as follows:

$$f_P(a,t) := \sum_{i \in \{1,\ldots,q\} : a = a_i} x_P(t - h_i)\ .$$

Put another way, the flow rate $x_P(t)$ is sent along $P$ and enters $a_i$ at time $t + h_i$. We want to restrict ourselves to paths over time that do not visit any vertex $v$ more than once. This can easily be achieved by removing the subpath starting at

the first arc with $\text{tail}(a_i) = v$ and ending with $\text{head}(a_j) = v$. The path continues with $a_{j+1}$ at time $h_{j+1}$. The next lemma summarizes that flow traveling along $P$ is indeed a flow over time. Such a flow is called a *path flow*. We omit a more detailed proof.

**Lemma 3.** *Let $P$ and $f_P$ be as described above and assume $\text{tail}(a_1) \in S^+$ and $\text{head}(a_q) \in S^-$. If $x_P(t) = 0$ for all $t \geq T$ for some fixed $T \in \mathbb{R}_{\geq 0}$, then $f_P$ is a flow over time that satisfies weak flow conservation and has time horizon at most $T + h_q + \tau_{a_q}$.*

A *path decomposition* describes a flow by the paths over time the flow units take. More precisely, a flow $f$ has a path decomposition if there is a finite set of paths over time $\mathcal{P}$ and functions $x_P : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, for $P \in \mathcal{P}$, such that $f$ is the sum of the path flows $f_P$, i. e., $f(a, \cdot) = \sum_{P \in \mathcal{P}} f_P(a, \cdot)$ for all $a \in A$.

A major advantage is that path decompositions can be readily manipulated while ensuring flow conservation. However, in general, a flow over time cannot be decomposed into path flows because flow units may travel in cycles. Omitting such cycles and possibly storing flow at an intermediate vertex instead does not have any drawbacks in our setting, and thus we can always obtain a path decomposition of a flow that is just-as-good. In the following, for $a, b \in \mathbb{R}$, let $\chi_{[a,b)} : \mathbb{R} \to \mathbb{R}$ be the characteristic function of time interval $[a, b)$.

**Lemma 4.** *Let $\tau$ be integral and $f$ a 1-constant flow on $\mathcal{N}$ with time horizon $T \in \mathbb{Z}_{\geq 0}$. Then there is a 1-constant flow $f' \leq f$ on $\mathcal{N}$ with the same demand vector as $f$ and the same time horizon $T$ such that $f'$ has a path decomposition with inflow rates of the form $d_P \chi_{[0,1)}(t)$ for some constants $d_P \in \mathbb{R}_{\geq 0}$, for $P \in \mathcal{P}$. Each path $P \in \mathcal{P}$ contains no vertex more than once.*

This follows from slightly rephrased techniques for time-expanded networks. Notice that the demands of the decomposed flow are the sum of the demands of the path flows.

### 2.3 Understanding the Problem

Flows with bridge capacities can always be interpreted as flows over time for larger traditional capacities: For 1-constant flows, the flow rates never exceed $u_a$. But this is a weak bound on the average capacity of an arc over a long period of time, which is essentially $u_a/\ell_a$. The effect of the aggregate arc capacities is clearly visible if the capacity windows are close to the time horizon. For instance, consider a network consisting of a single arc $a$ with length $\tau_a = 0$, capacity $u_a = 1$, and window $\ell_a$. The time horizon is $T > 0$. With traditional capacities, the flow rate is limited by $u_a$, and a flow can send only $T$ flow units in this instance. Bridge capacities, however, allow flow value 1 for any arbitrarily small time horizon $T > 0$ in this network. This is achieved by sending a short impulse with a high flow rate.

*Flow Conservation and Flow Value.* It is useful to think of bridge flows as flows that tend to send impulses as opposed to more uniform flow rates, that

traditional flow capacities necessitate. One might even wonder what reason there could be not to use the full capacity of an arc within a single interval $\Delta$ and then pause until the full capacity is available again $\ell_a$ time units later. We now discuss the small example in Figure 1. In the case of strict flow conservation, it already exhibits somewhat unexpected solutions that follow no such simple rule.

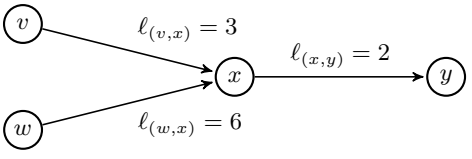



**Fig. 1.** The sources are $v$ and $w$, the sink is $y$. All capacities are 1 and all transit times are 0. The capacity windows are given on the arcs.

First note that arcs $(v, x)$ and $(w, x)$ together have an average capacity of $\frac{1}{3} + \frac{1}{6} = \frac{1}{2}$, which equals the average capacity of $(x, y)$. Thus, continuously sending flow with a rate of $\frac{1}{3}$ on $(v, x)$ and $\frac{1}{6}$ on $(w, x)$, then $\frac{1}{2}$ on $(x, y)$ yields a flow of value $T/2$, and this is optimal for average capacities (but need not be optimal for bridge capacities). This solution also satisfies strict flow conservation.

In contrast, a "pulsed" flow sends flow at a rate of $u_a$ for one time unit, and then waits for at least $\ell_a - 1$ time units. Since $u$ equals 1 for all arcs in our example, pulsed flows are exactly the integral flows.

A flow that satisfies only weak flow conservation can send a flow of value $\lceil \frac{T}{2} \rceil$. For this, one simply pulses flow into $(v, x)$ and $(w, x)$. These pulses start arriving at $x$ at times $\{0, 3, 6, 9, 12, \ldots\}$ and $\{0, 6, 12, \ldots\}$. They can be forwarded to $y$ at times $\{0, 2, 4, 6, 8, 10, 12, \ldots\}$ and this repeats with a periodicity of 12. Due to the capacity on $(x, y)$, this is the optimum flow value for integral time horizon.

Finally, a flow that satisfies strict flow conservation and uses only integral values cannot send 4 units of flow within time $T = 7$. For this, the first flow particles from each pulse would have to arrive at $x$ exactly at times $\{0, 2, 4, 6\}$. At most two of these could be contributed by the more restricted arc $(w, x)$, but the remaining time steps always contain a pair less than 3 apart. Therefore, they cannot all be supplied by $(v, x)$. On the other hand, the flow sending the average capacities achieves a flow value of 3.5, but this is not the optimum value of a fractional flow without storage. A flow of value $\frac{11}{3}$ is the true optimum, but showing this is best left to an LP-solver.

From this we can see that weak flow conservation really increases the set of feasible Dynamic Transshipment Problem instances and we will not consider strict flow conservation in the remainder of this paper. The proof of our main result also depends on weak flow conservation.

With respect to integrality, we want to draw attention to the complexity results from Papadimitriou et al. [8] (also described in [4]): Their problem can be seen as an integral flow on a network with dedicated arcs (that are blocked entirely as long as flow travels on them) with unit capacities. For $u_a = 1$, aggregate arc capacities and dedicated arcs behave the same, just like the pulsed flows in

the example above. We immediately obtain strong NP-hardness for *integral* flows with aggregate arc capacities. However, the Dynamic Transshipment Problem is only weakly NP-hard. Because of this, there must be instances where there is no integral but a fractional solution (assuming P $\neq$ NP).

**Corollary 3.** *The* Dynamic Transshipment Problem *restricted to integral flow functions is strongly NP-hard, even for unit capacities.*

## 3 Approximation Scheme

Our main result states that while it is NP-complete to decide whether an instance of the Dynamic Transshipment Problem is feasible, one can find an approximate solution that exceeds the time horizon and the capacities by a factor of $(1+\varepsilon)$, if the instance is feasible. For infeasible instances we might either prove they are infeasible or find feasible approximate solutions. Our approach is a non-trivial extension of the work of Fleischer and Skutella [1] on standard flows over time.

Throughout this section we consider an instance of the Dynamic Transshipment Problem consisting of $\mathcal{N} = (V, A, S^+, S^-, \tau, u, \ell)$, time horizon $T$, and demands $d$. All parameters are integers, so that we only need to consider 1-constant flows.

The actual algorithm is quite natural. For given $\varepsilon > 0$, as in [1], we choose a suitable discretization $\Delta \in \mathbb{Z}_{>0}$ such that $T/\Delta$ is polynomially bounded in the input size and $\varepsilon^{-1}$. The transit times are rounded up to multiples of $\Delta$, the rounded transit times are denoted by $\tau'_a := \lceil \tau_a \rceil_\Delta$, for $a \in A$. Moreover, the time horizon and capacities are increased slightly, while the lengths $\ell_a$ of the sliding windows remain the same. With Lemma 1, we can formulate the resulting new instance as a polynomial-sized linear program. If the new instance is infeasible, so is the original one. Otherwise, one obtains a $\Delta$-constant flow that approximately satisfies the Dynamic Transshipment Problem instance.

We have to prove two directions in order to show correctness of this algorithm. The easier one is that any solution to the rounded instance is indeed an approximate solution to the original instance. Intuitively, this is true since weak flow conservation is maintained when the flow is interpreted in the original network with shorter transit times.

**Lemma 5.** *If $f$ is a feasible flow on $\mathcal{N}' := (V, A, S^+, S^-, \tau', u, \ell)$ with time horizon $T'$, then $f$ is a feasible flow on $\mathcal{N}$ with the same time horizon, satisfying weak flow conservation and the same demands.*

*Proof (Sketch).* Any condition not involving transit times is identical for both networks. When we decrease the transit times from $\tau'$ to $\tau$, this affects the balance of the vertices in a one-sided way: Flow on an arc may arrive earlier, and then has to wait at the head vertex for an additional $\tau'_a - \tau_a$ time units. □

For the other direction we need to show that the existence of a feasible solution to the original instance implies feasibility of the rounded instance. The first problem here is that we increase transit times and, thus, need to rearrange the flow. Otherwise, flow conservation is violated since flow units will be sent onwards before they arrive at an intermediate vertex.

The main idea is to consider a path decomposition of a given feasible flow, as in [1], before we change transit times. Then we can reassemble the flow paths (now with longer transit times) and are guaranteed weak flow conservation. However, this might violate the capacities by a large factor: If multiple paths for the original transit times enter the same arc one after each other, they can possibly all be delayed to arrive simultaneously according to the rounded transit times. The solution is to make sure that the flow along each path is distributed over a larger time interval than flow units can possibly be delayed by the rounding. This new "smoothed" flow will still be congested, but the collisions are spread out equally in order to keep the violation of capacities bounded.

**Lemma 6.** *Let $f$ be a feasible 1-constant flow for the given* Dynamic Transshipment Problem *instance. Let $0 < \varepsilon < 1$ with $\varepsilon^{-1} \in \mathbb{Z}$ and $\Delta := \lfloor \varepsilon^2 T/|V| \rfloor$. If $\Delta > 0$, then there is a feasible $\Delta$-constant flow $\bar{f}$ on $\mathcal{N}' = (V, A, S^+, S^-, \tau', (1+\varepsilon)u, \ell)$ with time horizon $\bar{T} := (1+2\varepsilon)T$ satisfying the same demands as $f$.*

*Proof.* According to Lemma 4, we can replace $f$ by a flow that has a path decomposition consisting of a set of paths over time $\mathcal{P}$ and flow rates into the paths of the form $x_P(t) = d_P \chi_{[0,1)}(t)$ with $d_P \in \mathbb{R}_{\geq 0}$, for $P \in \mathcal{P}$. Moreover, each path uses every vertex at most once and thus consists of $q \leq |V| - 1$ arcs.

We now define for each path $P = (A^P, h^P) \in \mathcal{P}$ a path $P' = (A^{P'}, h^{P'})$ with the same sequence of arcs $A^{P'} = A^P$ that matches the transit times $\tau'$. To simplify notation, we denote $h_i^{P'}$ simply by $h'_i$, for $i = 1, \ldots, q$. Let $h'_1 := \lceil h_1^P \rceil_\Delta$ and $h'_i := \max\{\lceil h_i^P \rceil_\Delta, h'_{i-1} + \tau'_{a_{i-1}}\}$, for $i = 2, \ldots, q$. This yields a path over time, i. e., the starting times are compatible with the transit times. They are also multiples of $\Delta$ and $h'_i \geq h_i^P$, for all $i$. On the other hand, a simple induction yields $h'_i - h_i^P \leq i\Delta$. Since $q \leq |V| - 1$, we can generalize this to $0 \leq h'_i - h_i^P \leq |V|\Delta$, for all $i$.

Instead of sending flow according to the original function $d_P \chi_{[0,1)}$ into path $P'$, we smooth the flow as follows (in contrast to the approach in [1], we use a different, somewhat simpler smoothing here). Let $z := |V|/\varepsilon$, which is in $\mathbb{Z}$. We distribute the flow over an interval of length $z\Delta \leq \varepsilon T$. This can be accomplished by sending flow according to the function $x'_P(t) := \frac{d_P}{z\Delta} \cdot \chi_{[0,z\Delta)}(t)$ into path $P'$. The corresponding path flow is $f'_{P'}$ and we claim that the flow $\bar{f}$ defined by $\bar{f}(a, \cdot) := \sum_{P': P \in \mathcal{P}} f'_{P'}(a, \cdot)$, for $a \in A$, has the desired properties. It certainly satisfies weak flow conservation. The time horizon of each $f'_{P'}$ is at most $T + z\Delta + |V|\Delta \leq (1 + \varepsilon + \varepsilon^2)T \leq (1 + 2\varepsilon)T$. Each path flow $f'_{P'}$ still satisfies a demand of $d_P$. Since the inflow rates into the paths are $\Delta$-constant, and $\tau'$ was rounded to multiples of $\Delta$, each path flow and $\bar{f}$ are $\Delta$-constant. The important task left is to show that capacities $(1 + \varepsilon)u$ are obeyed.

For $a = a_i \in A^P$ we have $f'_{P'}(a,t) = \frac{d_P}{z\Delta}\chi_{[0,z\Delta)}(t - h'_i)$. We can conveniently relate these smoothed flow rates to $f_P(a,t)$.

$$\begin{aligned} f'_{P'}(a,t) &= \frac{d_P}{z\Delta}\chi_{[0,z\Delta)}(t-h'_i) = \frac{d_P}{z\Delta}\chi_{[h'_i-h_i,z\Delta+h'_i-h_i)}(t-h_i) \\ &\le \frac{d_P}{z\Delta}\chi_{[0,z\Delta+|V|\Delta)}(t-h_i) \\ &= \frac{d_P}{z\Delta}\sum_{\theta=0}^{z\Delta+|V|\Delta-1}\chi_{[\theta,\theta+1)}(t-h_i)\ . \end{aligned}$$

We continue by assuming $t \in \mathbb{Z}$, so we know that $t - h_i \in \mathbb{Z}$. For integral arguments, we can express $\chi_{[\theta,\theta+1)}(t)$ more convoluted as $\int_{t-\theta}^{t-\theta+1}\chi_{[0,1)}(1-\mu)\,\mathrm{d}\mu$ and reassemble the last sum into one integral:

$$\begin{aligned} f'_{P'}(a,t) &\le \frac{d_P}{z\Delta}\int_{t-h_i}^{t-h_i+z\Delta+|V|\Delta}\chi_{[0,1)}(1-\mu)\,\mathrm{d}\mu \\ &= \frac{1}{z\Delta}\int_0^{z\Delta+|V|\Delta} f_P(a,t+1-\mu)\,\mathrm{d}\mu\ . \end{aligned}$$

As promised, by smoothing the flow, the new path flow is still close to the averaged original flow, and importantly, the delayed flow units corresponding to $[z\Delta, z\Delta+|V|\Delta)$ only weigh in at $\frac{1}{z\Delta}$ their original rate.

For $a \in A$, we can now determine the capacity needed by the flow $\bar{f}$ resulting from the path flows. For $t \in \mathbb{Z}$, it holds that

$$\begin{aligned} \int_t^{t+\ell_a}\bar{f}(a,\theta)\,\mathrm{d}\theta &= \int_t^{t+\ell_a}\sum_{P':P\in\mathcal{P}} f'_{P'}(a,\theta)\,\mathrm{d}\theta \\ &\le \frac{1}{z\Delta}\int_t^{t+\ell_a}\int_0^{z\Delta+|V|\Delta}\sum_{P\in\mathcal{P}} f_P(a,\theta+1-\mu)\,\mathrm{d}\mu\,\mathrm{d}\theta \\ &= \frac{1}{z\Delta}\int_0^{z\Delta+|V|\Delta}\int_t^{t+\ell_a} f(a,\theta+1-\mu)\,\mathrm{d}\theta\,\mathrm{d}\mu \\ &\le \frac{1}{z\Delta}\int_0^{z\Delta+|V|\Delta} u_a\,\mathrm{d}\mu = \frac{z\Delta+|V|\Delta}{z\Delta}\,u_a\ . \end{aligned}$$

Since $z = |V|/\varepsilon$, the last term is exactly $(1+\varepsilon)u_a$. We satisfy these capacities for all $t \in \mathbb{Z}$. This covers all required test points $i\Delta$ and $j\Delta - l_a$. Thus, $\bar{f}$ is a feasible $\Delta$-constant flow on $\mathcal{N}'$ with time horizon $(1+2\varepsilon)T$, satisfying the same demands as $f$. □

Finally, our main theorem falls into place.

**Theorem 1.** *Given a feasible instance of the* Dynamic Transshipment Problem *and $\varepsilon > 0$, one can determine, in time polynomial in the input size and $\varepsilon^{-1}$, a feasible flow on $\bar{\mathcal{N}} = (V, A, S^+, S^-, \tau, (1+\varepsilon)u, \ell)$ with time horizon $(1+\varepsilon)T$ satisfying the given demands $d$.*

*Proof.* We can assume $\varepsilon < 1$ and use Lemma 6 for $\varepsilon'$ chosen such that $\frac{1}{4}\varepsilon < \varepsilon' \leq \frac{1}{2}\varepsilon$ and $1/\varepsilon' \in \mathbb{Z}$. If $\Delta = 0$, then $T \leq n/\varepsilon'^2$, and we can solve the exact problem for $\Delta = 1$. Otherwise, we can compute $\bar{f}$ on $(V, A, S^+, S^-, \tau', (1+\varepsilon')u, \ell)$ with time horizon $(1+2\varepsilon')T \leq (1+\varepsilon)T$. Since $T/\Delta \in O(|V|/\varepsilon^2)$, the flow $\bar{f}$ can be obtained efficiently by solving a linear program of polynomial size. According to Lemma 5, $\bar{f}$ is also a feasible flow for transit times $\tau$, and increasing the capacities from $(1+\varepsilon')u$ to $(1+\varepsilon)u$ is no problem, either. □

Note that simple examples exist showing that the slightly stronger approach of Fleischer and Skutella for the classical flow over time model (that does not require a violation of capacities) cannot be generalized to the setting of aggregate capacities.

## 4 Conclusion

We have introduced a generalized model of flows over time and presented a fully polynomial time approximation scheme with resource augmentation for the problem of computing optimal flows for this model. We only mention that the FPTAS can easily be generalized to the setting with multiple commodities, costs on the arcs and a given bound on the total flow cost.

**Acknowledgements.** The authors are indebted to Martin Guenther, Ronald Koch, and José Verschae for many helpful comments and discussions.

## References

1. Fleischer, L., Skutella, M.: Quickest flows over time. SIAM Journal on Computing 36, 1600–1630 (2007)
2. Ford, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. Operations Research 6, 419–433 (1958)
3. Hoppe, B., Tardos, É.: The quickest transshipment problem. Mathematics of Operations Research 25, 36–62 (2000)
4. Klinz, B., Woeginger, G.J.: One, two, three, many, or: complexity aspects of dynamic network flows with dedicated arcs. Operations Research Letters 22, 119–127 (1998)
5. Klinz, B., Woeginger, G.J.: Minimum-cost dynamic flows: The series-parallel case. Networks 43, 153–162 (2004)
6. Köhler, E., Skutella, M.: Flows over time with load-dependent transit times. SIAM Journal on Optimization 15, 1185–1202 (2005)
7. Melkonian, V.: Flows in dynamic networks with aggregate arc capacities. Information Processing Letters 101, 30–35 (2007)
8. Papadimitriou, C.H., Serafini, P., Yannakakis, M.: Computing the throughput of a network with dedicated lines. Discrete Applied Mathematics 42, 271–278 (1993)
9. Skutella, M.: An introduction to network flows over time. In: Cook, W., Lovász, L., Vygen, J. (eds.) Research Trends in Combinatorial Optimization, pp. 451–482. Springer, Heidelberg (2009)

# List Factoring and Relative Worst Order Analysis*

Martin R. Ehmsen[1], Jens S. Kohrt[1,2], and Kim S. Larsen[1]

[1] Department of Mathematics and Computer Science
University of Southern Denmark, Odense, Denmark
{ehmsen,svalle,kslarsen}@imada.sdu.dk
[2] CP[3]-Origins, University of Southern Denmark, Odense, Denmark

**Abstract.** Relative worst order analysis is a supplement or alternative to competitive analysis which has been shown to give results more in accordance with observed behavior of online algorithms for a range of different online problems. The contribution of this paper is twofold. First, it adds the static list accessing problem to the collection of online problems where relative worst order analysis gives better results. Second, and maybe more interesting, it adds the non-trivial supplementary proof technique of list factoring to the theoretical toolbox for relative worst order analysis.

## 1 Introduction

The static list accessing problem [29,4] is a well-known problem in online algorithms. Many deterministic as well as randomized algorithms are known, and these have been investigated theoretically as well as experimentally. See [10] for a discussion of the importance of the problem in relation to dictionary implementation, connections to paging, and applications in compression algorithms. For readers unfamiliar with the standard algorithms for list accessing or relative worst order analysis, we refer to the rigorous definitions in Sections 2 and 3.

The starting point for our work was the discrepancy between the findings obtained using competitive analysis [21,29,23] and the observations made through experimental work. Competitive analysis finds that Move-To-Front is optimal while Frequency-Count and Transpose have lower bounds on their competitive ratio which grow linearly with the length of the list [10]. In contrast, experimental results [9] suggest that Move-To-Front and Frequency-Count are almost equally good and both are far better than Transpose. Results from [7] seem to indicate the same.

To a large extent driven by the paging problem [10] and the difficulties there in theoretically separating various algorithm proposals, many alternative performance measures have been developed to supplement standard competitive analysis. Examples include [31,8,24,25,11,5]; see [16] for a survey. Some of these

* This work was supported in part by the Danish Natural Science Research Council.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 118–129, 2011.

measures are tailored towards a specific online problem, whereas others are more generally applicable; see [13] for a comparative study of these measures on a simple problem.

Of these alternatives to competitive analysis relative worst order analysis [11,12] is the measure that has been applied to the largest variety of online problems. Results that are in accordance with experiments have been derived for a range of fairly different online problems in situations where competitive analysis has given the "wrong" answer. Online problems of this nature include (but are not limited to) the following:

- Classical bin packing: Worst-Fit is better than Next-Fit [11].
- Dual bin packing: First-Fit is better than Worst-Fit [11].
- Paging: LRU is better than Flush-When-Full and look-ahead helps [12].
- Scheduling: minimizing makespan on two related machines, a post-greedy algorithm is better than scheduling all jobs on the fast machine [19].
- Bin coloring [26]: a natural greedy-type algorithm is better than just using one open bin at a time [18].
- Proportional price seat reservation: First-Fit is better than Worst-Fit [14].

We apply relative worst order analysis to the static list accessing problem. We first extend the list factoring technique [9,3] known from competitive analysis to relative worst order analysis. We then apply the technique to the three deterministic online list accessing algorithms Move-To-Front, Time-Stamp, and Frequency-Count. We show that these algorithms are equally good and much better than Transpose when analyzed using relative worst order analysis, thereby obtaining results that are in accordance with the cited experimental work.

Adding static list accessing to the collection of problems above where relative worst order analysis gives better or more nuanced results than competitive analysis is a step in documenting to what extent relative worst order analysis is generally applicable. It is also interesting that relative worst order analysis can be equipped with a powerful supplementary proof techniques such as list factoring. To our knowledge, relative worst order analysis is the first of the alternative performance measures to be equipped with a list factoring lemma.

Some of the deterministic list accessing algorithms are quite old. It is difficult to pin-point the origin of Frequency-Count, since it is intimately related to probability theoretical considerations, and it is not clear when it started being viewed as an algorithm. Move-To-Front and Transpose were formulated in [27]. Time-Stamp [1] is a deterministic algorithm that arose as a special case of a family of randomized algorithms.

In addition to the deterministic algorithms, we also consider the randomized algorithms BIT [28] and Randomized-Move-To-Front[1]. Deterministic and randomized online algorithms are often compared informally, but it is not clear how

[1] In the many papers that discuss Randomized-Move-To-Front, we have not been able to find a reference to the paper with the first definition of the algorithm. However, [7] cites personal communication with J. Westbrook from 1996 regarding properties of the algorithm.

much sense it makes to compare a worst-case guarantee with an average-case performance. We compare the two randomized algorithms to each other and find them incomparable whereas competitive analysis slightly favors the former (often referred to as a "surprising result"), showing that BIT is $\frac{7}{4}$-competitive and Randomized-Move-To-Front is 2-competitive against an oblivious adversary [28,20].

For early related work, we refer the reader to [4,10]. Newer work obtains separations between list accessing algorithms by analyzing these with respect to some measure of locality of reference [2,6,15].

## 2 List Accessing

In the *static list accessing problem* [29,4], we have a fixed collection of items arranged in a linear list, $\mathcal{L} = (a_1, a_2, \ldots, a_\ell)$, of length $\ell$. The request sequence, I, consists of requests of *access* to items in the list, and the accesses must be served in an online manner. The cost of accessing an item depends on its position (index) in the list. In the *full cost model*, accessing an item currently at index $j$ costs $j$. In the *partial cost model*, the final positive access is not counted, so accessing an item currently at index $j$ costs $j - 1$, denoted negative accesses.

After accessing an item, it can be moved to any position further towards the front of the list without any additional cost. Such a move can be seen as a number of transpositions of the accessed item with items preceding it in the list. The transpositions used to perform such a move are denoted *free*. Furthermore, at any time, an algorithm may exchange two adjacent items in the list at a cost of one. Such a transposition is denoted a *paid* transposition. The objective of a list accessing algorithm is to use free and paid transpositions in order to minimize the overall cost of serving the request sequence. Further discussion of the modelling issues can be found in [10].

Many different algorithms have been proposed for the list accessing problem. Some of the most well-known deterministic paging algorithms are the following.

MTF (Move-To-Front): After accessing the requested item, MTF moves the item to the front of the list.

FC (Frequency-Count): After accessing the requested item, FC moves the item forward in the list such that the resulting list is in sorted order with respect to the frequency with which the items have been accessed, i.e., for every item, FC maintains a counter which is incremented on an access to the item and the list is sorted in non-increasing order of the counters. FC only moves the accessed item forward the least number of positions necessary to maintain the sorted order.

TS (Time-Stamp): After accessing item $a_i$, it is inserted in front of the first item $a_j$ (from the front of the list) that precedes $a_i$ in the list and was accessed at most once since the last access to $a_i$. The algorithm does nothing if there is no such item $a_j$ or if $a_i$ is accessed for the first time.

TRANS (Transpose): After accessing the requested item, it is transposed with the item in front of it in the list. If the item is already at the front of the list, it stays there.

In addition to the above deterministic algorithms, we also consider the following well-known randomized algorithms.

BIT**:** For each item in the list, BIT [28] maintains a bit. Before processing a request sequence, BIT initializes the bits independently and uniformly at random. On a request for an item, BIT first complements the items bit. If the bit is then one, the item is moved to the front of the list. Otherwise, BIT does not move the item.

RMTF **(**Randomized-Move-To-Front**):** After each access to a requested item, RMTF moves the item to the front of the list with probability $\frac{1}{2}$.

## 3 Relative Worst Order Analysis

The relative worst order ratio was first introduced in [11] in an effort to combine the desirable properties of the max/max ratio [8] and the random-order ratio [24]. The measure was later refined in [12].

Instead of comparing online algorithms to an optimal offline algorithm (and then comparing their competitive ratios), two online algorithms are compared directly. However, instead of comparing their performance on the exact same request sequence, they are compared on their respective worst permutations of the same sequence.

Formally, if I is a request sequence of length $n$ and $\sigma$ is a permutation on $n$ elements, then $\sigma(\mathrm{I})$ denotes I permuted by $\sigma$. Let A be a list accessing algorithm and let $\mathrm{A}(\mathrm{I})$ denote the cost of running A on I. Define $\mathrm{A_W}(\mathrm{I})$ to be the performance of A on a worst possible permutation of I with respect to A, i.e., $\mathrm{A_W}(\mathrm{I}) = \max_\sigma\{\mathrm{A}(\sigma(\mathrm{I}))\}$.

For any pair of algorithms A and B, we define

$$\mathrm{c_u}(\mathrm{A},\mathrm{B}) = \inf\{c \mid \exists b\colon \forall \mathrm{I}\colon\ \mathrm{A_W}(\mathrm{I}) \le c\,\mathrm{B_W}(\mathrm{I}) + b\} \quad \text{and}$$
$$\mathrm{c_l}(\mathrm{A},\mathrm{B}) = \sup\{c \mid \exists b\colon \forall \mathrm{I}\colon\ \mathrm{A_W}(\mathrm{I}) \ge c\,\mathrm{B_W}(\mathrm{I}) - b\}.$$

Intuitively, $\mathrm{c_l}$ and $\mathrm{c_u}$ can be thought of as tight lower and upper bounds, respectively, on the performance of A relative to B.

If $\mathrm{c_l}(\mathrm{A},\mathrm{B}) \ge 1$ or $\mathrm{c_u}(\mathrm{A},\mathrm{B}) \le 1$, the algorithms are said to be *comparable* and the *relative worst order ratio*, $\mathrm{WR_{A,B}}$, of algorithm A to algorithm B is defined. Otherwise, $\mathrm{WR_{A,B}}$ is undefined.

$$\text{If } \mathrm{c_u}(\mathrm{A},\mathrm{B}) \le 1, \text{ then } \mathrm{WR_{A,B}} = \mathrm{c_l}(\mathrm{A},\mathrm{B}), \text{ and}$$
$$\text{if } \mathrm{c_l}(\mathrm{A},\mathrm{B}) \ge 1, \text{ then } \mathrm{WR_{A,B}} = \mathrm{c_u}(\mathrm{A},\mathrm{B}).$$

When either $\mathrm{c_l}(\mathrm{A},\mathrm{B}) \ge 1$ or $\mathrm{c_u}(\mathrm{A},\mathrm{B}) \le 1$ holds, the relative worst order ratio is a bound on how much better the one algorithm can be. If $\mathrm{WR_{A,B}} < 1$, then A is better than B, and if $\mathrm{WR_{A,B}} > 1$, then B is better than A. If A is better than B according to the relative worst order ratio, A and B are said to be comparable in A's favor. Finally, if the ratio is one, then the algorithms perform identically according to the relative worst order ratio.

In [11,12], it was shown that the relative worst order ratio is a transitive measure, i.e., the relative worst order ratio defines a partial ordering of the algorithms for a given problem.

## 4 List Factoring

The list factoring technique was first introduced by Bentley and McGeoch [9] and later extended and improved in a series of papers [22,30,1,3]. It reduces the analysis of list accessing algorithms to lists of size two. Previously the technique was developed and applied only in the context of competitive analysis, where it can be used to prove upper bounds on the competitive ratio [10]. In this section, we show that list factoring can also be applied in the context of relative worst order analysis to separate online algorithms and prove upper bounds.

In the following, let A denote any online list accessing algorithm that does not use paid transpositions. We are going to consider the partial cost model where accessing the $i$th item in the list costs $i-1$. For any request sequence I, let $\mathrm{A}^\star(\mathrm{I})$ denote the cost A incurs while processing I in the partial cost model.

Consider the list when A is about to process the $i$th request $I_i$ and define

$$\mathrm{A}^\star(a_j, i) = \begin{cases} 1 & \text{if } a_j \text{ is in front of } I_i \text{ in the list} \\ 0 & \text{otherwise (including } a_j = I_i) \end{cases}$$

for all items $a_j$ in the list.

We also define

$$\mathrm{A}^\star_{ab}(\mathrm{I}) = \sum_{i:I_i \in \{a,b\}} (\mathrm{A}^\star(a,i) + \mathrm{A}^\star(b,i))$$

It is an easy observation, also made in [10], that we can then write the cost of A on a sequence I in the partial cost model as

$$\mathrm{A}^\star(\mathrm{I}) = \sum_{\{a,b\} \subseteq \mathcal{L}, a \neq b} \mathrm{A}^\star_{ab}(\mathrm{I})$$

Let $\mathrm{I}_{ab}$ be the projection of I over $a$ and $b$, i.e., the sequence obtained from I by deleting all requests to items different from $a$ or $b$.

An algorithm A is said to have the *pairwise property*, if for all pairs, $a$ and $b$, of two items in $\mathcal{L}$, we have

$$\mathrm{A}^\star_{ab}(\mathrm{I}) = \mathrm{A}^\star(\mathrm{I}_{ab})$$

In competitive analysis, this setup can be used to prove upper bounds on the competitive ratio (in the partial cost model) of algorithms that have the pairwise property. In addition, if the algorithms also are cost independent (the decisions they make are independent of the cost), then the ratio carries over to the full cost model [10].

For the relative worst order ratio, we show that this technique can also be used to separate algorithms and prove upper bounds.

Consider an algorithm A that has the pairwise property. It follows that

$$\mathrm{A}^\star_{\mathrm{W}}(\mathrm{I}_{ab}) = \max_\sigma \mathrm{A}^\star(\sigma(\mathrm{I}_{ab})) = \max_\sigma \mathrm{A}^\star((\sigma(\mathrm{I}))_{ab}) = \max_\sigma \mathrm{A}^\star_{ab}(\sigma(\mathrm{I}))$$

The three equalities follow from the definition of a worst order, simple properties of permutations, and the pairwise property, respectively.

We now say that A has the *worst order projection property*, if and only if for all sequences I, there exist a worst ordering $\sigma_{\mathrm{A}}(\mathrm{I})$ of I with respect to A, such that for all pairs $\{a, b\} \subseteq \mathcal{L}$ $(a \neq b)$, $\sigma_{\mathrm{A}}(\mathrm{I})_{ab}$ is a worst ordering of $\mathrm{I}_{ab}$ with respect to A.

Using the above, we obtain a lemma similar to the Factoring Lemma for competitive analysis [10].

**Lemma 1.** *Let* A *and* B *be two online list accessing algorithms that do not use paid transpositions and that have the pairwise property and the worst order projection property, and let* $\mathcal{L}$ *be a list. If there exists constants* $c$ *and* $b_1$ *such that for every pair* $\{a, b\} \subseteq \mathcal{L}$ $(a \neq b)$*, and for every request sequence* I, $\mathrm{A}^\star_{\mathrm{W}}(\mathrm{I}_{ab}) \leq c\,\mathrm{B}^\star_{\mathrm{W}}(\mathrm{I}_{ab}) + b_1$*, then there exists a constant* $b_2$ *such that for every request sequence* I, $\mathrm{A}^\star_{\mathrm{W}}(\mathrm{I}) \leq c\,\mathrm{B}^\star_{\mathrm{W}}(\mathrm{I}) + b_2$.

*In addition, if* A *and* B *are cost independent and* $c \geq 1$*, then* $\mathrm{A}_{\mathrm{W}}(\mathrm{I}) \leq c\,\mathrm{B}_{\mathrm{W}}(\mathrm{I}) + b_2$.

*Proof.* Consider any algorithm A satisfying the hypothesis. Then

$$\begin{aligned}\mathrm{A}^\star_{\mathrm{W}}(\mathrm{I}) &= \max_\sigma \mathrm{A}^\star(\sigma(\mathrm{I})) = \max_\sigma \sum_{\{a,b\}\subseteq\mathcal{L},a\neq b} \mathrm{A}^\star_{ab}(\sigma(\mathrm{I})) \\ &= \sum_{\{a,b\}\subseteq\mathcal{L},a\neq b} \max_\sigma \mathrm{A}^\star_{ab}(\sigma(\mathrm{I})) = \sum_{\{a,b\}\subseteq\mathcal{L},a\neq b} \mathrm{A}^\star_{\mathrm{W}}(\mathrm{I}_{ab})\end{aligned}$$

Now consider two algorithms A and B satisfying the hypothesis. We get

$$\begin{aligned}\mathrm{A}^\star_{\mathrm{W}}(\mathrm{I}) &= \sum_{\{a,b\}\subseteq\mathcal{L},a\neq b} \mathrm{A}^\star_{\mathrm{W}}(\mathrm{I}_{ab}) \leq \sum_{\{a,b\}\subseteq\mathcal{L},a\neq b} (c\,\mathrm{B}^\star_{\mathrm{W}}(\mathrm{I}_{ab}) + b_1) \\ &= c \sum_{\{a,b\}\subseteq\mathcal{L},a\neq b} \mathrm{B}^\star_{\mathrm{W}}(\mathrm{I}_{ab}) + \sum_{\{a,b\}\subseteq\mathcal{L},a\neq b} b_1 = c\,\mathrm{B}^\star_{\mathrm{W}}(\mathrm{I}) + \binom{\ell}{2} b_1\end{aligned}$$

Hence, we have the result in the partial cost model. Now assume A and B are cost independent and $c \geq 1$. It is clear that for a cost independent algorithm A, the cost in the partial and the full cost model are related as $\mathrm{A}_{\mathrm{W}}(\mathrm{I}) = \mathrm{A}^\star_{\mathrm{W}}(\mathrm{I}) + |\,\mathrm{I}\,|$. Hence, $\mathrm{A}^\star_{\mathrm{W}}(\mathrm{I}) \leq c\,\mathrm{B}^\star_{\mathrm{W}}(\mathrm{I}) + b$ implies that $\mathrm{A}_{\mathrm{W}}(\mathrm{I}) \leq c\,\mathrm{B}_{\mathrm{W}}(\mathrm{I}) + b$ and the result follows.

It follows from the above that we can use list factoring to separate online algorithms, and an upper bound on the relative worst order ratio on lists of size two carries over to lists of any size. However, as it is also the case for competitive analysis, the list factoring technique cannot be used to prove lower bounds.

For *randomized algorithms*, the worst ordering is defined in terms of the algorithm's excepted cost when run on the sequence. In this case, a randomized algorithm is said to have either of the two properties if for all settings of the random choices made by the algorithm (a deterministic execution of the algorithm), the property holds. With this definition, it is clear that the list factoring technique can also be applied to randomized algorithms.

In the following, we repeatedly use the fact that MTF, FC, and TS have the pairwise property and are cost independent [10].

## 5 Worst Orderings

Intuition suggests that one can obtain a worst ordering of any sequence for most online list accessing algorithms by considering the request sequence as a multiset of items and always request the item from the multiset which currently is farthest back in the list.

Formally, for any deterministic online list accessing algorithm A and any request sequence I, we inductively define the *FB ordering* (Farthest Back ordering) of I as follows. Let $S^0$ be the multiset of all items requested in I. Let $S^{i-1}$ be $S^0$ with the first $i-1$ items in the FB ordering removed. The $i$th item in the FB ordering of I with respect to A, $\mathrm{FB_A(I)}_i$, is the item in $S^{i-1}$ which currently is farthest back in the list after A has processed the first $i-1$ requests of $\mathrm{FB_A(I)}$. In addition, we say that A has the *FB property* if for any request sequence the FB ordering of that sequence is a worst ordering with respect to A.

When the algorithm in question is obvious, we drop it from the notation and write FB(I). Note that for any deterministic algorithm and request sequence, the FB ordering of this input sequence is uniquely determined.

Observe that TRANS does not have the FB property as the following example illustrates. Consider the request sequence $\mathrm{I} = \langle a, b, c, c\rangle$ with the initial list $\mathcal{L} = (a, b, c)$. In this case, we have $\mathrm{FB(I)} = \langle c, b, c, a\rangle$ with $\mathrm{TRANS(FB(I))} = 10$. However, on the ordering $\mathrm{I}' = \langle c, c, b, a\rangle$, TRANS incurs a cost of 11. Hence, FB(I) is not a worst ordering for TRANS.

The other deterministic algorithms considered in this chapter do have the FB property.

**Lemma 2.** MTF, TS, *and* FC *all have the* FB *property.*

*Proof.* Omitted due to space restrictions; see the full version of the paper [17].

When applying the list factoring technique, we use the following lemma.

**Lemma 3.** *If a deterministic algorithm has the* FB *property, then it also has the worst order projection property.*

*Proof.* This holds since a projection of an FB ordering is again an FB ordering.

Thus, MTF, FC, and TS all have the worst order projection property.

# 6 Algorithm Comparisons

## 6.1 Deterministic Algorithms

**Theorem 1.** *The algorithms* MTF *and* FC *perform identically according to the relative worst order ratio.*

*Proof.* We apply the list factoring technique introduced in Section 4 since both FC and MTF have the FB property.

Consider any request sequence I and any pair $\{a, b\} \subseteq \mathcal{L}$, $a \neq b$. Assume without loss of generality that the initial list has $a$ in front of $b$, i.e., $\mathcal{L}_{ab} = (a, b)$.

Now, the FB ordering of $\mathrm{I}_{ab}$ for MTF is of the form $\langle (b, a)^m \rangle$ with a possible tail of repeated requests to either $a$ or $b$, whichever is requested the most in I. The FB ordering for FC is of the form $\langle (b, a, a, b)^{\lfloor \frac{m}{2} \rfloor} \rangle$ with a possible tail of repeated requests to either $a$ or $b$, whichever is requested the most in I. Observe that if $m$ is not divisible by two, there is an extra request to either $a$ or $b$. However, such a request only contributes a constant extra cost which we can ignore. It now follows that the cost for FC on its worst permutation (the FB ordering) is the same as the cost for MTF on its worst permutation (the FB ordering), except for a possible additive constant.

**Theorem 2.** *The algorithms* MTF *and* TS *perform identically according to the relative worst order ratio.*

*Proof.* We apply the list factoring technique introduced in Section 4 since both TS and MTF have the FB property.

Consider any request sequence I and any pair $\{a, b\} \subseteq \mathcal{L}$, $a \neq b$. Assume without loss of generality that the initial list projected onto $a$ and $b$ has $a$ at the front, i.e., $\mathcal{L}_{ab} = (a, b)$.

The FB ordering of $\mathrm{I}_{ab}$ for TS is of the form $\langle (b, b, a, a)^{\lfloor \frac{m}{2} \rfloor} \rangle$. The remaining arguments are exactly the same as in the proof of Theorem 1.

Combining the previous two lemmas and using the fact that the relative worst order ratio is a transitive measure, we arrive at the following corollary.

**Corollary 1.** *The algorithms* MTF*,* TS*, and* FC *perform identically according to the relative worst order ratio.*

We now show that TRANS cannot be better than any of MTF, TS, and FC according to the relative worst order ratio.

**Lemma 4.** *There exists a constant $b$ such that for any request sequence* I*,*

$$\mathrm{MTF}_{\mathrm{W}}(\mathrm{I}) \leq \mathrm{TRANS}_{\mathrm{W}}(\mathrm{I}) + b$$

*Proof.* Omitted due to space restrictions; see the full version of the paper [17].

On the other hand, TRANS can be much worse than MTF, FC, and TS under the relative worst order ratio.

**Theorem 3.** $\mathrm{WR}_{\mathrm{TRANS,MTF}} \geq \frac{\ell}{2}$.

*Proof.* Lemma 4 shows that TRANS cannot be better than MTF according to the relative worst order ratio. Assume that the initial list is $\mathcal{L} = (a_1, a_2, \ldots, a_\ell)$ and consider the request sequence $\mathrm{I} = \langle (a_\ell, a_{\ell-1})^m \rangle$.

It is clear that MTF incurs a cost of $2\ell + 4(m-1)$ on its worst permutation of I. On the other hand, TRANS leaves the two items at the end of the list and incurs a cost of $2m\ell$. For $m$ approaching infinity, the ratio approaches $\frac{\ell}{2}$.

### 6.2 Randomized Algorithms

In this section, to make the proofs more readable, we use the partial cost model. Here, as in the rest of this paper, all results hold for the full cost model as well.

**Lemma 5.** *For integers $n \geq 1$ and $m \geq 2$ and a request sequence $\mathrm{I} = \langle (b, a^m)^n \rangle$ with initial list $\mathcal{L} = (a, b)$, the expected cost of* BIT *for a single repetition of $\langle b, a^m \rangle$ is $\frac{7}{4}$, and* I *is its own worst permutation with respect to* BIT.

*Proof.* For each access to $b$, at most the next two accesses to $a$ contribute to the expected cost of BIT. It follows by induction that after each repetition of $\langle b, a^m \rangle$, $a$ is at the front of the list for BIT. Hence, the expected cost of the prefix $\langle b, a \rangle$ of the next repetition is $\frac{3}{2}$, and after that $a$ is at the front of BIT's list with probability $\frac{3}{4}$. Thus, the expected cost of the following access to $a$ is $\frac{1}{4}$, after which $a$ is at the front of BIT's list with probability 1, and the remaining accesses to $a$ in the current repetition do not cost anything. Hence, for any $m$, the total expected cost of a single repetition is $\frac{7}{4}$ for BIT. It is clear that I is its own worst permutation for BIT.

**Lemma 6.** *There exists a request sequence* I *such that the expected cost for* RMTF *on its worst permutation of* I *is strictly less than the expected cost for* BIT *on its worst permutation.*

*Proof.* Consider the request sequence $\mathrm{I} = \langle (b, a, a)^n \rangle$ for some integer $n$ with the initial list $\mathcal{L} = (a, b)$.

For BIT, by Lemma 5, the cost of each repetition of $\langle b, a, a \rangle$ is $\frac{7}{4}$.

For RMTF, first consider any subsequence $\langle b, a^m \rangle$ of a worst ordering of I for some positive integer $m$. Assume that before this subsequence, in RMTF's execution, $a$ is at the front of the list with probability $p$.

After the access to $b$, $a$ is not at the front with probability $1 - \frac{p}{2}$. In this case, the up to $m$ requests to $a$ while it is not at the front can be described by a truncated geometric distribution [10, Lemma 4.1] with an expected number of $2\left(1 - \frac{1}{2^m}\right)$. Hence, the cost of the entire subsequence is

$$c_m(p) = p + \left(1 - \frac{p}{2}\right) 2 \left(1 - \frac{1}{2^m}\right) = 2 - \frac{2-p}{2^m}$$

The probability of $a$ being at the front of the list after the repetition is then

$$1 - \frac{1 - \frac{p}{2}}{2^m} = 1 - \frac{2 - p}{2^{m+1}}$$

Now, consider the input sequence $\langle (b, a^m)^n \rangle$ for $n$ approaching infinity. The probability of $a$ being at front of the list after each repetition of $ba^m$ approaches $p_m$, where

$$p_m = 1 - \frac{2 - p_m}{2^{m+1}} \Rightarrow p_m = 1 - \frac{1}{2^{m+1} - 1}$$

Hence, the cost of a repetition approaches

$$c_m(p_m) = 2 - \frac{2 - p_m}{2^m} = 2 - \frac{1}{2^m} - \frac{1}{2^m(2^{m+1} - 1)} = \frac{2^{m+2} - 4}{2^{m+1} - 1}$$

The results are summarized in Table 1, including values for small $m$.

**Table 1.** The cost and the value of $p$ after the phase for various $m$

| $m$ | $c_m(p)$ | $p$ after phase | $p_m$ | $c_m(p_m)$ |
|---|---|---|---|---|
| 0 | $p$ | $\frac{p}{2}$ | 0 | 0 |
| 1 | $1 + \frac{p}{2}$ | $\frac{1}{2} + \frac{p}{4}$ | $\frac{2}{3}$ | $\frac{4}{3}$ |
| 2 | $\frac{3}{2} + \frac{p}{4}$ | $\frac{3}{4} + \frac{p}{8}$ | $\frac{14}{15}$ | $\frac{12}{7}$ |
| 3 | $\frac{7}{4} + \frac{p}{8}$ | $\frac{7}{8} + \frac{p}{16}$ | $\frac{30}{31}$ | $\frac{28}{15}$ |
| $\geq 4$ | $\leq 2$ | $\leq 1$ | $\leq 1$ | $\leq 2$ |

Returning to I, assume for the moment that the ordering of I is indeed a worst ordering for RMTF. By the above, it is clear that for $p < 1$, the expected cost for RMTF to serve a repetition is strictly less than the expected cost for BIT, and only on the very first repetition is $p = 1$; all following repetitions have $p < 1$. Also, the cost of a repetition approaches $c_2(p_2) = \frac{12}{7}$ for $n$ approaching infinity. This is strictly less than the cost of BIT, $\frac{7}{4}$.

Now, we only need to show that the ordering of I is a worst ordering for RMTF. The proof of this is omitted due to space restrictions; see the full version of the paper [17]. The approach is to match up phases of the form $\langle b, a^m \rangle$.

**Lemma 7.** *There exists a request sequence* I *such that the expected cost for* BIT *on its worst permutation of* I *is strictly less than the expected cost for* RMTF *on its worst permutation.*

*Proof.* Consider the request sequence $\mathrm{I} = \langle (b, a, a, a)^n \rangle$, $n \geq 1$, with the initial list $\mathcal{L} = (a, b)$.

For BIT, by Lemma 5, the cost of each repetition of $\langle b, a, a, a \rangle$ is $\frac{7}{4}$, and I is its own worst permutation.

For RMTF, by Table 1, the cost of each repetition approaches $c_3(p_3) = \frac{28}{15}$ from above. Since this is strictly more than $\frac{7}{4}$, $\mathrm{RMTF}(\mathrm{I}) > \mathrm{BIT}_W(\mathrm{I})$.

An interesting observation is that the sequences used in the previous lemmas are both repetitions of the pattern $\langle b, a^m \rangle$ for different values of $m$. The previous two lemmas imply the following:

**Corollary 2.** BIT *and* RMTF *are incomparable under relative worst order analysis.*

## 7 Open Problems

In order to apply the list factoring technique together with relative worst order analysis, both of the "pairwise properties" and the "worst order projection properties" must hold. We have not been able to show a dependence between these two properties, i.e., does one follow from the other? On the other hand, we have not been able to exhibit an example for which one holds and the other does not. Another interesting question is whether the list factoring technique can be used with performance measures other than competitive analysis and, as demonstrated here, relative worst order analysis.

## Acknowledgments

We would like to thank Joan Boyar for initial discussions on the relationship between MTF and TRANS.

## 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., 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)
3. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Information Processing Letters 56, 135–139 (1995)
4. Albers, S., Westbrook, J.: Self-organizing data structures. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms — The State of the Art. LNCS, vol. 1442, pp. 13–51. Springer, Heidelberg (1998)
5. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: 18th ACM-SIAM Symposium on Discrete Algorithms, pp. 229–237 (2007)
6. 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)
7. Bachrach, R., El-Yaniv, R.: Online list accessing algorithms and their applications: Recent empirical evidence. In: Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 53–62 (1997)
8. Ben-David, S., Borodin, A.: A New Measure for the Study of On-Line Algorithms. Algorithmica 11, 73–91 (1994)
9. Bentley, J.L., McGeoch, C.C.: Amortized analyses of self-organizing sequential search heuristics. Communications of the ACM 28, 404–411 (1985)

10. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
11. Boyar, J., Favrholdt, L.M.: The relative worst order ratio for on-line algorithms. ACM Transactions on Algorithms 3(2), article 22 (2007)
12. Boyar, J., Favrholdt, L.M., Larsen, K.S.: The relative worst-order ratio applied to paging. Journal of Computer and System Sciences 73, 818–843 (2007)
13. Boyar, J., Irani, S., Larsen, K.S.: A Comparison of Performance Measures for Online Algorithms. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 119–130. Springer, Heidelberg (2009)
14. Boyar, J., Medvedev, P.: The relative worst order ratio applied to seat reservation. ACM Transactions on Algorithms 4(4), article 48 (2008)
15. Dorrigiv, R., Ehmsen, M.R., López-Ortiz, A.: Parameterized analysis of paging and list update algorithms. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 104–115. Springer, Heidelberg (2010)
16. Dorrigiv, R., López-Ortiz, A.: A Survey of Performance Measures for On-line Algorithms. SIGACT News 36(3), 67–81 (2005)
17. Ehmsen, M.R., Kohrt, J.S., Larsen, K.S.: List factoring and relative worst order analysis (2010); arXiv:1009.5787
18. Epstein, L., Favrholdt, L.M., Kohrt, J.S.: Comparing online algorithms for bin packing problems. Journal of Scheduling (accepted for publication)
19. Epstein, L., Favrholdt, L.M., Kohrt, J.S.: Separating scheduling algorithms with the relative worst order ratio. Journal of Combinatorial Optimization 12(4), 362–385 (2006)
20. Garefalakis, T.: A new family of randomized algorithms for list accessing. In: Burkard, R.E., Woeginger, G.J. (eds.) ESA 1997. LNCS, vol. 1284, pp. 200–216. Springer, Heidelberg (1997)
21. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17(2), 416–429 (1969)
22. Irani, S.: Two results on the list update problem. Information Processing Letters 38(6), 301–306 (1991)
23. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica 3, 79–119 (1988)
24. Kenyon, C.: Best-fit bin-packing with random order. In: 7th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 359–364 (1996)
25. Koutsoupias, E., Papadimitriou, C.H.: Beyond Competitive Analysis. SIAM Journal on Computing 30(1), 300–317 (2001)
26. Krumke, S.O., de Paepe, W.E., Rambau, J., Stougie, L.: Bincoloring. Theoretical Computer Science 407(1-3), 231–241 (2008)
27. McCabe, J.: On serial files with relocatable records. Operations Research 13(4), 609–618 (1965)
28. Reingold, N., Westbrook, J., Sleator, D.D.: Randomized competitive algorithms for the list update problem. Algorithmica 11, 15–32 (1994)
29. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)
30. Teia, B.: A lower bound for randomized list update algorithms. Information Processing Letters 47, 5–9 (1993)
31. Young, N.: The $k$-server dual and loose competitiveness for paging. Algorithmica 11, 525–541 (1994)

# Approximation Algorithms for Domination Search*

Fedor V. Fomin[1], Petr A. Golovach[2], and Dimitrios M. Thilikos[3]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
fomin@ii.uib.no
[2] School of Engineering and Computing Sciences, Durham University, South Road, Durham DH1 3LE, UK
petr.golovach@durham.ac.uk
[3] Department of Mathematics, University of Athens, Panepistimioupolis, GR-15784 Athens, Greece
sedthilk@math.uoa.gr

**Abstract.** The $r$-domination search game on graphs is a game-theoretical approach to the investigation of several graph and hypergraph parameters including treewidth and hypertree width. The task is to identify the minimum number of cops sufficient to catch the visible and fast robber. In $r$-domination search, the robber is being arrested if he resides inside a ball of radius $r$ around some cop. In this setting, the power of the cops does not depend only on how many they are but also on the local topology of the graph around them. This is the main reason why the approximation complexity of the $r$-domination search game varies considerably, depending on whether $r = 0$ or $r \geq 1$. We prove that this discrepancy is canceled when the game is played in (non-trivial) graph classes that are closed under taking of minors. We give a constant factor approximation algorithm that for every fixed $r$ and graph $H$, computes the minimum number of cops required to capture the robber in the $r$-domination game on graphs excluding $H$ as a minor.

**Keywords:** Domination search, graph minors, approximation algorithms.

## 1 Introduction

Graph searching games are played on graphs (in this paper all graphs are undirected and simple), where a group of searchers (cops) tries to catch a fugitive (robber). In the model known as a node searching, the robber stands on a vertex of the graph and at any moment he can run (arbitrarily fast) to another vertex along a path in the graph. However he is not allowed to run through a vertex occupied by a cop [14,15]. Each cop at any time either stands on a vertex

* The research of the first author is supported by the Norwegian Research Council. The research of the second author is supported by the EPSRC EP/GO43434/1. The research of the third author is supported by the project "Kapodistrias" (ΑΠ 02839/28.07.2008) of the National and Kapodistrian University of Athens.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 130–141, 2011.

or is in a helicopter (that is, is temporarily removed from the game). The aim of the cops is to capture the robber by landing a cop via helicopter on a vertex occupied by the robber and the robber's objective is to avoid capture. There are two variants of the game, which were studied intensively depending on whether cops posses complete information on the current location of the robber (i.e. the robber is visible to cops) [7,22] or when the cops have no such information (i.e. the robber is invisible) [14,17,18,20]. It appeared that the visible case is strongly related to the fundamental graph parameter called treewidth, and that the invisible case is related to the pathwidth of a graph. We refer to [11] for further references on graph searching.

In the domination or $r$-domination versions of graph searching, cops' aim is more modest: instead of capturing the robber their task is that, during the game, at least one of the cops will be at distance $r$ or closer to the robber (the distance is in the standard shortest path metric of the graph). Another interpretation of the $r$-domination game is that cops have more power and can catch the robber not just by occupying his vertex but by just having the robber in its $r$-neighborhood. As in the case of classical search games, there are two versions of the game, one with visible [16] and the other with invisible robber [2,10].

In this paper we study the $r$-domination search game with a visible robber. This game is a natural generalization of the search game introduced by Seymour and Thomas [22], and thus for $r = 0$, $k + 1$ cops can capture the robber on a graph $G$ if and only if the treewidth of $G$ is at most $k$. For $r = 1$, the $r$-domination search is strongly related to the Marshals and Robber game played on hypergraphs. The Marshals and Robber game is a game-theoretic approach to the investigation of the hypertree-width, another intensively studied parameter within the context of several applications [1,12,13]. Kreutzer and Ordyniak have shown in [16] that for any hypergraph $\mathcal{H}$, a graph $G_H$ with the property that the minimum number of marshals required to win on $\mathcal{H}$ equals the 1-domination cop number of $G_H$ can be constructed. That way $r$-domination search game is a powerful model serving as a general game-theoretical model for a number of fundamental parameters.

However, there is a price one has to pay for such a generality—the computational complexity of the game changes drastically with even small changes of $r$, like from 0 to 1. For example, computing the treewidth of a graph, and thus the minimum number of cops for $r = 0$ is fixed parameter tractable [3], while for $r = 1$ the problem becomes $W[2]$-hard [16]. This change is also indicated by the the fact that the corresponding graph parameter is *not* closed under the operation of taking a graph minor, and thus most of the powerful techniques from Graph Minor Theory cannot be applied. Moreover, the search number can be approximated within a factor of $c \cdot \sqrt{\log n}$ for $r = 0$, while already for $r = 1$ it is NP-complete to approximate he search number within a $c \cdot \log n$ factor [16]. The main explanation of this behavior is that for $r \geq 1$ the power of the searchers depends not only on how many they are but also on the *local topology* of their position (i.e. their $r$-neighborhoods). The results of this paper indicate that this drastic change is *canceled* when imposing the sparsity restriction of the

absence of some fixed graph as a minor: then, the approximation complexity of the general problem (when $r \geq 1$) does not deviate from the special case (where $r = 0$).

In this paper we give several approximation algorithms computing the minimum number of cops required to win in the $r$-dominating search games for graphs excluding some fixed graph as a minor. For planar graphs, and more generally, for graphs excluding some fixed apex graph as a minor, we show that for every fixed $r \geq 1$, the $r$-domination cop number of a graph $G$ can be approximated within a constant multiplicative factor by the treewidth of $G$ (and hence by the 0-domination cop number). Since there are constant factor approximation algorithms for the treewidth of such graphs, our results yield approximation algorithms. While techniques from Graph Minor Theory do not seem to be applicable for $r \geq 1$, we use the recent results from [9] on contractions in graphs. This type of arguments cannot be extended further. For example, it is well known that the treewidth of a $(k \times k)$-grid is $k$. If we add just one universal vertex $v$ adjacent to all vertices of the grid, we obtain a graph of treewdith $k+1$. Clearly, this graph belongs in the class of graphs excluding $K_6$ as a minor. However, for $r \geq 1$, one cop placed on $v$ is at distance at most $r$ to every vertex of the graph, and one cop can always win. Thus on graphs excluding some fixed graph $H$ as a minor, the $r$-domination cop number of a graph cannot be approximated by its treewidth. And this marks the borderline where the difference between the cases $r = 0$ and $r \geq 1$ becomes computationally essential.

Our approximation algorithm for computing the $r$-domination cop number of an $H$-minor-free graph $G$ is technical. The main idea is to construct in polynomial time a new $H$-minor-free graph $G'$ such that its treewidth sandwiches up to a constant factor the cop number of $G$. Our approach extends the technique used in [8] for approximating a series of graph or hypergraph parameters including fractional and generalized hypertree-width. This reduces the whole problem to the case where $r = 0$, which is known to have a constant factor approximation for $H$-minor free graphs, due to the results of Feige et al. in [6]. In that sense, our results constitute an extension of the corresponding results of [6], for every $r \geq 1$.

The paper is organized as follows. In Section 2, we give the formal definition of the dominating search game as well as an annotated extension of it. In the same section, we also give the main definitions and some preliminary results that are important for the proofs of the later sections. Our results on apex minor-free graphs are presented in Section 3, while the approximation algorithm for $H$-minor-free graphs and the proofs of the results supporting its correctness are presented in Section 4. Finally, in Section 5 we conclude with some open problems.

## 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 $N_G[v] = N_G(v) \cup \{v\}$. For $U \subseteq V(G)$, we put $N_G[U] = \bigcup_{v \in U} N_G[v]$. The *distance* $\mathrm{dist}_G(u, v)$ between vertices $u$ and $v$ in a connected graph $G$ is the number of edges in a shortest $(u, v)$-path in $G$. For a positive integer $r$, $N_G^{(r)}[v] = \{u \in V(G) \colon \mathrm{dist}_G(u, v) \leq r\}$ and for $U \subseteq V(G)$, $N_G^{(r)}[U] = \bigcup_{v \in U} N_G^{(r)}[v]$. Whenever there is no ambiguity we omit the subscripts. If $U \subseteq V(G)$ (resp. $u \in V(G)$ or $E \subset E(G)$ or $e \in E(G)$) then $G - U$ (resp. $G - u$ or $G - E$ or $G - e$) is the graph obtained from $G$ by the removal of vertices of $U$ (resp. of vertex $u$ or edges of $E$ or of the edge $e$). For graphs $G_1$ and $G_2$, $G_1 \cap G_2$ ($G_1 \cup G_2$ respectively) is the graph with the vertex set $V(G_1) \cap V(G_2)$ and the edge set $E(G_1) \cap E(G_2)$ (the vertex set $V(G_1) \cup V(G_2)$ and the edge set $E(G_1) \cup E(G_2)$ respectively).

*Cops and Robber game.* We consider a generalization of the Helicopter Cops and Robber game introduced by Seymour and Thomas [22]. Let $G$ be a connected undirected graph, and let $r$ be a non-negative integer. The distance $r$ domination search game is played by two players: cop and robber. The cop-player has a team of cops who attempt to capture the robber. The robber stands on a vertex of the graph, and can at any time run at great speed to any other vertex along a path of the graph. However, he is not permitted to run through a vertex at distance at most $r$ from a vertex occupied by a cop. Each cop at any time either stands on a vertex or is in a helicopter (that is, is temporarily removed from the game). The aim of the cop-player is to capture the robber by landing a cop via helicopter on a vertex at distance at most $r$ from the vertex occupied by the robber, and the robber's objective is to avoid capture. The robber can see the movements of helicopters and may run to a new vertex before the helicopter lands. We consider the variant of the game when the robber is visible. For an integer $r$ and a graph $G$, we denote by $\mathbf{dc}_r(G)$ the minimum number of cops sufficient for the cops to win on graph $G$ and call it the *$r$-domination cop number*.

*Black & White Domination Cops and Robber game.* It is convenient for us to consider an annotated variant of the Domination Cop and Robber Game. In this variant the robber can only occupy vertices from a prescribed set and move along edges of the subgraph induced by this set. Let $G$ be a graph, and let $B$ (*black* vertices) and $W$ (*white* vertices) be a partition of the set of vertices $V(G)$. We assume that $B \neq \emptyset$ (the set $W$ can be empty). We call a graph with a given partition $B$ and $W$ the *black and white* graph. By $B(G)$ and $W(G)$ we denote the set of black vertices and the set of white vertices of $G$ respectively.

Let $G$ be a black and white graph, and let $r$ be a non-negative and $k$ be positive integers. We define the *position* of the cops as a set of vertices $U \subset V(G)$, $|U| \leq k$, occupied by the cops (clearly, we can assume that each vertex is occupied by at most one cop). We denote by $\mathcal{U}_k$ the set of all possible position of the cops. The *position of the robber* is a vertex of $B(G)$ occupied by him. The *strategy*

*of cops* is a function $C\colon \mathcal{U}_k \times B(G) \to \mathcal{U}_k$. Calls of this function correspond to moves of cops. If the cops have a position $U$ and the robber has a position $v$, then the cops move to the position $U' = C(U, v)$: cops remain on the vertices of $U \cap U'$, the cops from $U \setminus U'$ are removed from the graph, and then cops are placed on vertices of $U' \setminus U$. Respectively, we define the *strategy of the robber* as a function $R\colon \mathcal{U}_k \times \mathcal{U}_k \times B(G) \to B(G)$ such that if $v' = R(U, U', v)$ then there is a $(v, v')$-path $P$ in $G[B(G)]$ with the property $V(P) \cap N_G^{(r)}[U \cap U'] = \emptyset$. Calls of this function corresponds to moves of the robber. If the cops are moving from a position $U$ to $U'$ and the robber occupies $v$, then he moves from $v$ to $v' = R(U, U', v)$.

The game is defined by the (possibly infinite) sequence of pairs from $\mathcal{U}_k \times B(G)$ $(U_0, v_0), (U_1, v_1), \ldots$, where $U_0 = \emptyset$, $U_i = C(U_{i-1}, v_{i-1})$ and $v_i = R(U_{i-1}, U_i, v_{i-1})$. This sequence is finite if there is $m \geq 1$ such that $v_m \in N_G^{(r)}(U_m)$. In this case we say that the cop-player wins, otherwise it is said that the robber-player wins.

A strategy of cops is called a *winning* strategy, if cop-players wins for any choice of a strategy by the robber-player. The *$r$-domination cop number* $\mathbf{dc}_r(G, B(G))$ is the minimum number of cops $k$ such that they have a winning strategy of cops. For $W(G) = \emptyset$, we let $\mathbf{dc}_r(G) = \mathbf{dc}_r(G, V(G))$. The *winning* strategy for the robber is a strategy such that the robber wins against any strategy of cops. In what follows we usually give informal descriptions of strategies of the cops and the robber by describing their movements. It is easy to make the following two observations.

**Proposition 1.** *For any two non-negative integers $r, r'$, $r \leq r'$, and any black and white graph $G$, $\mathbf{dc}_r(G, B(G)) \geq \mathbf{dc}_{r'}(G, B(G))$.*

**Proposition 2.** *Let $G$ be a black and white graph, $X \subseteq W(G)$ and $N_G^{(r)}[X] \cap B(G) = \emptyset$. Then $\mathbf{dc}_r(G, B(G)) = \mathbf{dc}_r(G - X, B(G))$.*

The complexity of the $r$-Domination Cops and Robbers problem was considered in [16]. This problem asks for given non-negative integer $r$, positive integer $k$ and a given connected graph $G$, whether $\mathbf{dc}_r(G) \leq k$.

**Proposition 3 ([16]).** *For $r \geq 1$, the $r$-Domination Cops and Robbers problem is*

*i)* NP*-hard,*
*ii)* W[2]*-hard when parameterized by $k$, and*
*iii) there is a constant $c$ such that there is no polynomial time algorithm that approximates the $r$-domination cop number for $n$-vertex graphs within a multiplicative factor $c \cdot \log n$, unless* P $\neq$ NP.

*Contractions and minors.* Given an edge $e = \{x, y\}$ of a graph $G$, the graph $G/e$ is obtained from $G$ by contracting the edge $e$, i.e. the endpoints $x$ and $y$ are replaced by a new vertex $v_{xy}$ which is adjacent to the old neighbors of $x$ and $y$ (except $x$ and $y$). We say that $x$ and $y$ are contracted to $v_{xy}$, and we

also sometimes say that $x$ is contracted to $y$ (or $y$ to $x$). For a black and white graph $G$, it is assumed that if $x \in B(G)$ and $y \in B(G)$ then the obtained vertex $v_{xy}$ is black and $v_{xy}$ is white otherwise. A graph $H$ obtained by a sequence of edge-contractions is said to be a *contraction* of $G$.

It can be observed that the $r$-domination cop number is a contraction-closed parameter.

**Proposition 4.** *Let $H$ be a contraction of a connected black and white graph $G$. For any $r \geq 0$, $\mathbf{dc}_r(H, B(H)) \leq \mathbf{dc}_r(G, B(G))$.*

It is said that a graph $H$ is a *minor* of a graph $G$ if $H$ is the contraction of some subgraph of $G$. We say that a graph $G$ is *$H$-minor-free* when it does not contain $H$ as a minor. We also say that a graph class $\mathcal{G}$ is *$H$-minor-free* (or, excludes $H$ as a minor) when all its members are $H$-minor-free. An *apex graph* is a graph that can be made planar by the removal of a single vertex. A graph class $\mathcal{G}$ is *apex-minor-free* if $\mathcal{G}$ excludes a fixed apex graph $H$ as a minor.

*Grids and their triangulations.* Let $k$ and $r$ be positive integers where $k, r \geq 2$. The *$(k \times r)$-grid* is the Cartesian product of two paths of lengths $k-1$ and $r-1$ respectively. A vertex of a $(k \times r)$-grid is a *corner* if it has degree 2. Thus each $(k \times r)$-grid has 4 corners. A vertex of a $(k \times r)$-grid is called *internal* if it has degree 4, otherwise it is called *external*.

A *partial triangulation* of a $(k \times r)$-grid is a planar graph obtained from a $(k \times r)$-grid (we call it the *underlying grid*) by adding edges. Let us note that there are many non-isomorphic partial triangulations of on underlying grid. For each partial triangulation of a $(k \times r)$-grid we use the terms *corner*, *internal* and *external* referring to the corners, the internal and the external vertices of the underlying grid. We define $\Gamma_k$ (see Figure 1) as the following (unique, up

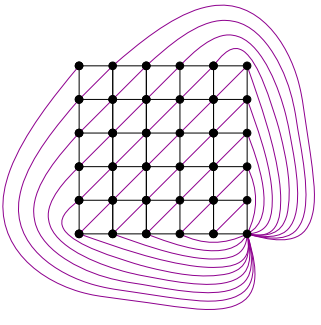


**Fig. 1.** The graph $\Gamma_6$

to isomorphism) triangulation of a plane embedding of the $(k \times k)$-grid. Let $\Gamma$ be a plane embedding of the $(k \times k)$-grid such that all external vertices are on the boundary of the external face. We triangulate internal faces of the $(k \times k)$-grid such that all the internal vertices have degree 6 in the obtained graph and all

non-corner external vertices have degree 4, and then one corner of degree two is joined by edges with all vertices of the external face (we call this corner *loaded*). We need the following claim.

**Lemma 1.** *Let $G$ be a black & white graph where*

*i) $B(G)$ induces $\Gamma_k$ for some $k > 2r+1$,*
*ii) $W(G)$ is an independent set, and*
*iii) for any $v \in W(G)$, $N_G(v)$ induces a clique in $G$.*

*Then for any $r \geq 0$, $\mathbf{dc}_r(G, B(G)) \geq \frac{k}{2r+1} - 1$.*

*Proof.* We prove that if $p < \frac{k}{2r+1} - 1$ then the robber has a winning strategy on $G$ against $p$ cops. Let $B(G) = \{(i,j) | 0 \leq i \leq k-1, 0 \leq j \leq k-1\}$. It is assumed that the vertices are numbered in such a way that $(i,j)$ and $(i',j')$ are adjacent in the underlying grid for $\Gamma_k$ if and only if $i = i'$ and $|j - j'| = 1$ or $|i - i'| = 1$ and $j = j'$. Denote by $X_i$ the set of vertices $\{(i,j) \mid r \leq j \leq k-r-1\}$ for $i \in \{r, \ldots, k-r-1\}$, and let $Y_j = \{(i,j) \mid r \leq i \leq k-r-1\}$ for $j \in \{r, \ldots, k-r-1\}$. Let also $\mathcal{U}_p$ be the set of all subsets of $V(G)$ with at most $p$ elements (i.e. $\mathcal{U}_p$ is the set of all possible positions of $p$ cops). By *i*), each vertex $v \in W(G)$ is adjacent to either one vertex of $B(G)$, or two adjacent vertices in this set, or to three vertices which compose a triangle in $\Gamma_k$. Using this property and *ii*), we conclude that for any vertex $v \in V(G)$, there is a vertex $v' \in B(G)$ such that $N_G^{(r)}[v] \cap B(G) \subseteq N_{\Gamma_k}^{(r)}[v']$. Hence for any $U \in \mathcal{U}_p$, there are $i(U), j(U) \in \{r, \ldots, k-r-1\}$ such that $N_G^{(r)}[U] \cap X_{i(U)} = \emptyset$ and $N_G^{(r)}[U] \cap Y_{j(U)} = \emptyset$. We define the robber's strategy $R$ as follows: for any $U, U' \in \mathcal{U}_p$ and each $(i,j) \in B(G)$, $R(U, U', (i,j)) = (i(U'), j(U'))$. It remains to note that if $i = i(U)$ and $j = j(U)$ then $Z = X_i \cup Y_j \cup X_{i(U')} \cup Y_{j(U)}$ induces a connected subgraph in $\Gamma_k$ and $N_{\Gamma_k}^{(r)}[U \cap U'] \cap Z = \emptyset$. Therefore $R$ is a winning strategy for the robber.

*Treewidth.* A *tree decomposition* of a graph $G$ is a pair $(\mathcal{X}, T)$ where $T$ is a tree with nodes $\{1, \ldots, m\}$ and $\mathcal{X} = \{X_i \mid i \in V(T)\}$ is a collection of subsets of $V(G)$ (called *bags*) such that:

*i)* $\bigcup_{i \in V(T)} X_i = V(G)$,
*ii)* for each edge $\{x,y\} \in E(G)$, $\{x,y\} \subseteq X_i$ for some $i \in V(T)$, and
*iii)* for each $x \in V(G)$ the set $\{i \mid x \in X_i\}$ induces a connected subtree of $T$.

The *width* of a tree decomposition $(\{X_i \mid i \in V(T)\}, T)$ is $\max_{i \in V(T)} \{|X_i| - 1\}$. The *treewidth* of a graph $G$ denoted $\mathbf{tw}(G)$ is the minimum width over all tree decompositions of $G$.

It is well known that Seymour and Thomas [22] established a close connection between treewidth and graph searching.

**Proposition 5 ([22]).** *For any connected graph $G$, $\mathbf{dc}_0(G) = \mathbf{tw}(G) + 1$.*

*Surfaces.* A *surface* $\Sigma$ is a compact 2-manifold without boundary (we always consider connected surfaces). Whenever we refer to a *$\Sigma$-embedded graph* $G$ we

consider a 2-cell embedding of $G$ in $\Sigma$. To simplify notations, we do not distinguish between a vertex of $G$ and the point of $\Sigma$ used in the drawing to represent the vertex or between an edge and the line representing it. We also consider a graph $G$ embedded in $\Sigma$ as the union of the points corresponding to its vertices and edges. That way, a subgraph $H$ of $G$ can be seen as a graph $H$, where $H \subseteq G$. Recall that $\Delta \subseteq \Sigma$ is an open (resp. closed) disc if it is homeomorphic to $\{(x,y) : x^2+y^2 < 1\}$ (resp. $\{(x,y) : x^2+y^2 \leq 1\}$) in $\mathbf{R}^2$. The *Euler genus* of a non-orientable surface $\Sigma$ is equal to the non-orientable genus $\tilde{g}(\Sigma)$ (or the cross-cap number). The *Euler genus* of an orientable surface $\Sigma$ is $2g(\Sigma)$, where $g(\Sigma)$ is the orientable genus of $\Sigma$. We refer to the book of Mohar and Thomassen [19] for more details on graphs embeddings. The *Euler genus* of a graph $G$ (denoted by $\mathbf{eg}(G)$) is the minimum integer $\gamma$ such that $G$ can be embedded on a surface of the Euler genus $\gamma$.

## 3 $r$-Domination Cop Number for Apex-Minor-Free Graphs

We prove here that the $r$-domination cop number of an apex-minor-free graphs can be approximated by its treewidth.

**Theorem 1.** *Let $r$ be a non-negative integer and let $H$ be an apex graph. Then for any connected graph $G$ excluding $H$ as a minor, it holds that $\mathbf{dc}_r(G) - 1 \leq \mathbf{tw}(G) \leq c_{H,r} \cdot \mathbf{dc}_r(G)$ where $c_{H,r}$ is a constant depending only on $H$ and $r$.*

*Proof.* By Proposition 5, it is sufficient to prove this theorem for $r > 0$. The first inequality follows immediately from Proposition 1 and Proposition 5. The proof of the second inequality is based on the following result.

**Proposition 6 ([9]).** *For every apex graph $H$, there is $c_H > 0$ such that every connected $H$-minor-free graph of treewidth at least $c_H \cdot k$, where $k$ is a positive integer, contains $\Gamma_k$ as a contraction.*

Let $k = \lfloor \frac{\mathbf{tw}(G)}{c_H} \rfloor$ and observe that $G$ contains $\Gamma_k$ as a contraction and also $\mathbf{tw}(G) \leq c_H(k+1)$. Assume that $k > 2r+1$. By Proposition 4 and Lemma 1 (for $B(G) = V(G)$ and $W(G) = \emptyset$), $\mathbf{dc}_r(G) \geq \mathbf{dc}_r(\Gamma_k) \geq \frac{k}{2r+1} - 1$. Hence $(\mathbf{dc}_r(G)+1)(2r+1) \geq k$ and therefore $\mathbf{tw}(G) \leq c_H((\mathbf{dc}_r(G)+1)(2r+1)+1)$. Now we can conclude that there is a constant $c_{H,r}$ such that $\mathbf{tw}(G) \leq c_{H,r} \cdot \mathbf{dc}_r(G)$.

## 4 $r$-Domination Cop Number for $H$-Minor-Free Graphs

The following theorem is the main result of this paper.

**Theorem 2.** *Let $r$ be a positive integer and $H$ be a graph. There is a polynomial time algorithm that given a connected graph $G$ excluding $H$ as a minor returns a $c_{H,r}$-factor approximation of $\mathbf{dc}_r(G)$, where $c_{H,r}$ is a constant depending only on $H$ and $r$.*

The remaining part of this section is devoted to the proof of Theorem 2.

### 4.1 Graph Minor Theorem and Preliminary Results

The proof is based the Excluded Minor Theorem from the Graph Minor theory. Before we state it, we need some definitions.

**Definition 1** (Clique-Sums). *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two disjoint graphs, and $k \geq 0$ an integer. For $i = 1, 2$, let $W_i \subseteq V_i$, form a clique of size $h$ and let $G_i'$ be the graph obtained from $G_i$ by removing a set of edges (possibly empty) from the clique $G_i[W_i]$. Let $F : W_1 \to W_2$ be a bijection between $W_1$ and $W_2$. The* $h$-clique-sum *of $G_1$ and $G_2$, denoted by $G_1 \oplus_{h,F} G_2$, or simply $G_1 \oplus G_2$ if there is no confusion, is the graph obtained by taking the union of $G_1'$ and $G_2'$ by identifying $w \in W_1$ with $F(w) \in W_2$, and by removing all the multiple edges. The image of the vertices of $W_1$ and $W_2$ in $G_1 \oplus G_2$ is called the* join *of the sum.*

Note that some edges of $G_1$ and $G_2$ are not edges of $G$, since it is possible that they had edges which were removed by clique-sum operation. Such edges are called *virtual* edges of $G$. We remark that $\oplus$ is not well defined; different choices of $G_i'$ and the bijection $F$ could give different clique-sums. A sequence of $h$-clique-sums, not necessarily unique, which result in a graph $G$, is called a *clique-sum decomposition* of $G$.

**Definition 2** (*$h$-nearly embeddable graphs*). *Let $\Sigma$ be a surface and $h > 0$ be an integer. A graph $G$ is* $h$-nearly embeddable *in $\Sigma$ if there is a set of vertices $Z \subseteq V(G)$ (called* apices*) of size at most $h$, such that graph $G - Z$ is the union of subgraphs $R_0, \ldots, R_h$ with the following properties:*

*i) There is a set of cycles $C_1, \ldots, C_h$ of $R_0$ in $\Sigma$ such that the cycles $C_i$ are the borders of open pairwise disjoint discs $\Delta_i$ in $\Sigma$;*
*ii) $R_0$ has an embedding in $\Sigma$ in such a way that $\Delta_1, \ldots, \Delta_h$ are faces of $R_0$;*
*iii) Graphs $R_1, \ldots, R_h$ (called* vortices*) are pairwise disjoint and for $1 \leq i \leq h$, $V(R_0) \cap V(R_i) \subseteq C_i$;*
*iv) For $1 \leq i \leq h$, let $U_i := \{u_1^i, \ldots, u_{m_i}^i\}$ be the vertices of $V(R_0) \cap V(R_i) \subset C_i$ appearing in an order obtained by clockwise traversing of $C_i$. Then $R_i$ has a path decomposition $\mathcal{B}_i = (B_j^i)_{1 \leq j \leq m_i}$ of width at most $h$, such that $u_j^i \in B_j^i$ for $1 \leq j \leq m_i$.*

The following proposition is known as the Excluded Minor Theorem [21] and is the cornerstone of Robertson and Seymour's Graph Minors theory.

**Proposition 7 ([21]).** *For every non-planar graph $H$, there exists an integer $c_H$, depending only on $H$, such that every graph excluding $H$ as a minor can be obtained by $c_H$-clique-sums from graphs that can be $c_H$-nearly embedded in a surface $\Sigma$ in which $H$ cannot be embedded. Moreover, while applying each of the clique sums, at most three vertices from each summand other than apices and vertices in vortices are identified.*

### 4.2 Approximation of the $r$-Domination Cop Number

Now we are ready to describe our approximation of the $r$-domination cop number for $H$-minor-free graphs. Let $H$ be a graph. We assume that $H$ is not planar (otherwise we can apply Theorem 1). Let $G$ be a graph that does not contain $H$ as a minor. Let $G_1 \oplus \cdots \oplus G_m$ be a $c_H$-clique-sum decomposition of $G$. Denote by $Z_i$ the set of apices of $G_i$. For $i = 1, \ldots, m$, we define $F(G_i)$ as the graph obtained if we consider $G_i - N_G^{(r)}[Z_i]$ and then we remove each virtual edge $\{x, y\}$ of $G_i$ such that all $(x, y)$-paths in $G$ whose internal vertices are not in $V(G_i)$ are intersected by $N_G^{(r)}[Z_i]$.

The proof of Theorem 2 is based on the following theorem.

**Theorem 3.** *Let $r$ be a positive integer, let $H$ be a graph and let $G$ be a connected graph excluding $H$. Let also $k = \max\{\mathbf{tw}(F(G_i)) \mid i = 1, \ldots, r\}$. Then, $\mathbf{dc}_r(G) - c_{H,r} \leq k \leq c_{H,r} \cdot \mathbf{dc}_r(G)$ where $c_{H,r}$ is a constant depending only on $H$ and $r$.*

*Proof.* Due the space restrictions we only sketch here the main ideas of our proof. We start with the first inequality. It is based on the following claim.

**Claim 1.** $\mathbf{dc}_r(G) \leq k + 2c_H + 1$.

*Sketch of the proof of Claim 1.* Let $p = k + 2c_H + 1$. We describe a winning strategy for $p$ cops on $H$.

The clique-sum decomposition $G = G_1 \oplus G_2 \oplus \cdots \oplus G_m$ can be considered as a tree decomposition $(\mathcal{X}, T)$ of $G$ for some tree $T$ with nodes $\{1, 2, \ldots, m\}$ with the bags $X_i = V(G_i)$, i.e. the vertex sets of the summands are the bags of this decomposition. The idea behind the winning strategy for cops is to "chase" the robber in the graph along $m+1$ decompositions: one is induced by the clique-sum decomposition and others are tree decompositions of $F(G_i)$.

Now our aim is to prove the second inequality.

**Claim 2.** *There is a constant $c_{H,r}$ such that $k \leq c_{H,r} \cdot \mathbf{dc}_r(G)$.*

*Sketch of the proof of Claim 2.* Assume that $k = \mathbf{tw}(F(G_i))$ for some $1 \leq i \leq m$, and denote $F = F(G_i)$. Assume that $F$ is connected (otherwise let $F$ be a component of $F(G_i)$ with treewidth $k$). Consider a component of $G - N_G^{(r)}[Z_i]$ which contains vertices of $V(F)$, denote by $B(G)$ the set of its vertices and let $W(G) = V(G) \setminus B(G)$. Clearly, $\mathbf{dc}_r(G) \geq \mathbf{dc}_r(G, B(G))$. By Proposition 2, $\mathbf{dc}_r(G, B(G)) \geq \mathbf{dc}_r(G - Z_i, B(G))$. Also using this proposition we can assume that $G' = G - Z_i$ is connected (otherwise vertices of components of $G - Z_i$ which do not contain $B(G)$ can be removed, since they are at least $(r + 1)$-distant from vertices of $B(G)$). Now we contract all edges $\{x, y\}$ of $G'$ such that either $x, y \in W(G')$ or $x, y \in B(G') \setminus V(F)$. Denote the obtained graph by $\hat{G}$, and let $\hat{F}$ be the subgraph of $\hat{G}$ induced by $B(\hat{G})$. Note that $W(\hat{G})$ is an independent set of $\hat{G}$. By Proposition 4, $\mathbf{dc}_r(G', B(G)) \geq \mathbf{dc}_r(\hat{G}, B(\hat{G}))$. Hence $\mathbf{dc}_r(G) \geq \mathbf{dc}_r(\hat{G}, B(\hat{G}))$.

Recall that all summands in the clique-sum decomposition of $G$ can be $c_H$-nearly embedded in some surface $\Sigma$ in which $H$ cannot be embedded in a such way that while applying each of the clique sums, at most three vertices from each summand other than apices and vertices in vortices are identified. The graph $F$ is a subgraph of $G_i$. Therefore, the $c_H$-nearly embedding of $G_i$ induces $c_H$-nearly embedding of $F$. Using same arguments as in the proof of Proposition 6 (Theorem 1 in [9]), it can be proved that there exists a constant $c > 0$, depending only on $\Sigma$, such that if $\mathbf{tw}(F) \geq c \cdot p$, then $\hat{F}$ can be contracted to $\Gamma_p$. Moreover, if we consider $\hat{G}$ and contract in it the edges that are contracted in $\hat{F}$ in order to construct $\Gamma_p$ then for the obtained black and white graph $Q$, each vertex of $W(Q)$ is adjacent to a clique in $B(Q)$. By Proposition 4, $\mathbf{dc}_r(\hat{G}, B(\hat{G})) \geq \mathbf{dc}_r(Q, B(Q))$.

Now the set $B(Q)$ induces $\Gamma_p$, $W(Q)$ is independent, and for each vertex $v \in W(Q)$, $N_Q(v)$ induces a clique in $Q$. Therefore it is possible to apply Lemma 1 and conclude that $\mathbf{dc}_r(Q, B(Q)) \geq \frac{p}{2r+1} - 1$ and $\mathbf{dc}_r(G) \geq \frac{p}{2r+1} - 1$.

It remains to note that $(2r+1)(\mathbf{dc}_r(G)+1) \geq p \geq \frac{k}{c}-1$ and $c((2r+1)(\mathbf{dc}_r(G)+1)+1) \geq k$. Now we can choose a constant $c_{H,r}$ for which $c_{H,r} \cdot \mathbf{dc}_r(G) \geq k$.

Finally, Demaine et al. [5] proved that a clique-sum decomposition can be obtained in time $O(n^c)$ for some constant $c$ which depends only on $H$ (see also [4]). As far as we constructed summands $G_i$, the construction of graphs $F(G_i)$ can be done in polynomial time. Moreover, since the algorithm of Demaine et al. provides $c_H$-nearly embeddings of these graphs, it is possible to use it to construct a polynomial constant factor approximation algorithm for the computation of $\mathbf{tw}(F(G_i)$ (see also [6]). This concludes the proof of Theorem 2.

## 5 Open Problems

We conclude with the following open problems.

- Can $\mathbf{dc}_r$ be computed in polynomial time on $H$-minor free graphs? This problem is open even for $r = 0$.
- For every $r \geq 1$, the parameterized version of $\mathbf{dc}_r(G) \leq k$ is $\mathsf{W}[2]$-hard. What is the parameterized complexity of the problem on planar graphs?

## References

1. Adler, I., Gottlob, G., Grohe, M.: Hypertree width and related hypergraph invariants. Eur. J. Comb. 28, 2167–2181 (2007)
2. Aggarwal, D., Mehta, S.K., Deogun, J.S.: Domination search on graphs with low dominating-target-number. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 28–37. Springer, Heidelberg (2005)
3. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
4. Dawar, A., Grohe, M., Kreutzer, S.: Locally excluding a minor. In: LICS, pp. 270–279. IEEE Computer Society, Los Alamitos (2007)

5. Demaine, E.D., Hajiaghayi, M.T., Kawarabayashi, K.: Algorithmic graph minor theory: Decomposition, approximation, and coloring. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), pp. 637–646. IEEE Computer Society, Los Alamitos (2005)
6. Feige, U., Hajiaghayi, M., Lee, J.R.: Improved approximation algorithms for minimum weight vertex separators. SIAM J. Comput. 38, 629–657 (2008)
7. Fomin, F.V., Fraigniaud, P., Nisse, N.: Nondeterministic graph searching: from pathwidth to treewidth. Algorithmica 53, 358–373 (2009)
8. Fomin, F.V., Golovach, P.A., Thilikos, D.M.: Approximating acyclicity parameters of sparse hypergraphs. In: Albers, S., Marion, J.-Y. (eds.) STACS, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. LIPIcs, vol. 3, pp. 445–456 (2009)
9. Fomin, F.V., Golovach, P.A., Thilikos, D.M.: Contraction bidimensionality: The accurate picture. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 706–717. Springer, Heidelberg (2009)
10. Fomin, F.V., Kratsch, D., Müller, H.: On the domination search number. Discrete Appl. Math. 127, 565–580 (2003)
11. Fomin, F.V., Thilikos, D.M.: An annotated bibliography on guaranteed graph searching. Theor. Comput. Sci. 399, 236–245 (2008)
12. Gottlob, G., Leone, N., Scarcello, F.: Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. J. Comput. System Sci. 66, 775–808 (2003)
13. Gottlob, G., Miklós, Z., Schwentick, T.: Generalized hypertree decompositions: NP-hardness and tractable variants. J. ACM 56, 1–32 (2009)
14. Kirousis, L.M., Papadimitriou, C.H.: Interval graphs and searching. Discrete Math. 55, 181–184 (1985)
15. Kirousis, L.M., Papadimitriou, C.H.: Searching and pebbling. Theoret. Comput. Sci. 47, 205–218 (1986)
16. Kreutzer, S., Ordyniak, S.: Distance $d$-Domination Games. In: Paul, C., Habib, M. (eds.) WG 2009. LNCS, vol. 5911, pp. 308–319. Springer, Heidelberg (2010)
17. LaPaugh, A.S.: Recontamination does not help to search a graph. J. Assoc. Comput. Mach. 40, 224–245 (1993)
18. Megiddo, N., Hakimi, S.L., Garey, M.R., Johnson, D.S., Papadimitriou, C.H.: The complexity of searching a graph. J. ACM 35, 18–44 (1988)
19. Mohar, B., Thomassen, C.: Graphs on surfaces. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore (2001)
20. Parsons, T.D.: The search number of a connected graph. In: Proceedings of the Ninth Southeastern Conference on Combinatorics, Graph Theory, and Computing. Congress. Numer., Utilitas Math., vol. XXI, pp. 549–554 (1978)
21. Robertson, N., Seymour, P.D.: Graph minors. XVI. Excluding a non-planar graph. J. Combin. Theory Ser. B 89, 43–76 (2003)
22. Seymour, P.D., Thomas, R.: Graph searching and a min-max theorem for treewidth. J. Combin. Theory Ser. B 58, 22–33 (1993)

# Lower Bounds for Smith's Rule in Stochastic Machine Scheduling

Caroline Jagtenberg[1], Uwe Schwiegelshohn[2], and Marc Uetz[3]

[1] Utrecht University, Dept. of Mathematics, P.O. Box 80010,
3508 TA Utrecht, The Netherlands
carolinetjes@gmail.com
[2] TU Dortmund, Robotics Research Institute, 44227 Dortmund, Germany
uwe.schwiegelshohn@udo.edu
[3] University of Twente, Dept. of Applied Mathematics, P.O. Box 217,
7500 AE Enschede, The Netherlands
m.uetz@utwente.nl

**Abstract.** We consider the problem to minimize the weighted sum of completion times in nonpreemptive parallel machine scheduling. In a landmark paper from 1986, Kawaguchi and Kyan [5] showed that scheduling the jobs according to the WSPT rule –also known as Smith's rule– has a performance guarantee of $\frac{1}{2}(1+\sqrt{2}) \approx 1.207$. They also gave an instance to show that this bound is tight. We consider the stochastic variant of this problem in which the processing times are exponentially distributed random variables. We show, somehow counterintuitively, that the performance guarantee of the WSEPT rule, the stochastic analogue of WSPT, is not better than 1.243. This constitutes the first lower bound for WSEPT in this setting, and in particular, it sheds new light on the fundamental differences between deterministic and stochastic scheduling problems.

**Keywords:** stochastic scheduling, WSEPT, exponential distribution.

## 1 Introduction

Minimizing the weighted sum of completion times on $m$ parallel, identical machines is an archetypical problem in the theory of scheduling. In this problem, we are given $n$ jobs which have to be processed non-preemptively on $m$ machines. Each job $j$ comes with a processing time $p_j$ and a weight $w_j$, and when $C_j$ denotes job $j$'s completion time in a given schedule, the goal is to compute a schedule that minimizes the total weighted completion time $\sum_j w_j C_j$. In the classical 3-field notation for scheduling problems [3], the problem is denoted $\mathrm{P}\,|\,|\sum w_j C_j$. For a single machine, a simple exchange argument shows that scheduling the jobs in order of nonincreasing ratios $w_j/p_j$ gives the optimal schedule [11]. Greedily scheduling the jobs in this order is known as WSPT rule or Smith's rule. On parallel identical machines, WSPT is known to be a $\frac{1}{2}(1+\sqrt{2})$–approximation, and this bound is tight [5]. The computational tractability of the problem was

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 142–153, 2011.

finally settled by showing the existence of a PTAS [10], given that the problem is strongly NP-complete if $m$ is part of the input [2].

In this paper, we consider the stochastic variant of the problem. It is assumed that the processing time $p_j$ of a job $j$ is not known in advance. It becomes known upon completion of the job. Only the distribution of the corresponding random variable $P_j$, or at least its expectation $\mathbb{E}[P_j]$, is given beforehand. More specifically, we assume that the processing times of jobs are governed by independent, exponentially distributed random variables. That is to say, each job comes with a parameter $\lambda_j > 0$, and the probability that its processing time exceeds $t$ equals

$$\mathbb{P}[P_j > t] = e^{-\lambda_j t}.$$

We denote that by writing $P_j \sim \exp(\lambda_j)$. Exponentially distributed processing times somehow represent the cream of stochastic scheduling, in particular when juxtaposing stochastic and deterministic scheduling: The exponential distribution is characterized by the memory-less property, that is,

$$\mathbb{P}[P_j > s + t \mid P_j > s] = \mathbb{P}[P_j > t].$$

So for any non-finished job it is irrelevant how much processing it has already received. This is obviously a decisive difference to deterministic scheduling models, and puts stochastic scheduling apart. Next to that, the model with exponentially distributed processing times is attractive because it makes the stochastic model analytically tractable.

In the stochastic setting, the analogue of Smith's rule is greedily scheduling the jobs in order of non-increasing ratios $w_j/\mathbb{E}[P_j]$, also called WSEPT [8]. For a single machine, this is again optimal [9]. For parallel machines, it has been shown that the WSEPT rule achieves a performance bound of $(2 - \frac{1}{m})$ within the class of all stochastic scheduling policies [7]. That is to say, if $\Pi^*$ denotes an optimal stochastic scheduling policy, then

$$\mathbb{E}\left[\sum w_j C_j^{\mathrm{WSEPT}}\right] \leq \left(2 - \frac{1}{m}\right) \mathbb{E}\left[\sum w_j C_j^{\Pi^*}\right].$$

We refer e.g. to [6] for precise definitions on stochastic scheduling policies. In general, scheduling policies can be quite complicated, and it is not even clear if the optimal policy is non-idling in the setting considered here [12]. For the purpose of this paper, it suffices to know that stochastic scheduling policies are non-anticipatory in the sense that they are, at any given time $t$, only allowed to use information that is available at that time $t$. This is also the case for WSEPT, as the distributions $P_j$, and particularly $\mathbb{E}[P_j]$ are available beforehand.

The major purpose of this paper is to establish the first lower bound for the $(2 - \frac{1}{m})$ performance guarantee of [7] for exponentially distributed processing times. In fact, we are not aware of any result in this direction. The only result known to us is an instance showing that WSEPT can miss the optimum by a factor 3/2, but then for arbitrary processing time distributions [13, Ex. 3.5.12]. Our main result is the following.

**Theorem 1.** *When scheduling jobs with exponentially distributed processing times on parallel, identical machines in order to minimize* $\mathbb{E}[\sum w_j C_j]$*, the performance guarantee of WSEPT is not better than* $\alpha$ *with* $\alpha > 1.243$*.*

To obtain our result, we carefully adapt and analyze the worst-case instance of [5]. Note that the originality of this result lies in the fact that $1.243 > \frac{1}{2}(1+\sqrt{2}) \approx 1.207$. Hence, stochastic scheduling with exponentially distributed processing times has worse worst-case instances than deterministic scheduling.

This result may seem counterintuitive: For unit weights where $w_j = 1$, the SPT rule is optimal for minimizing $\sum_j C_j$ in the deterministic setting [8], and also SEPT is optimal for minimizing $\mathbb{E}[\sum_j C_j]$ when processing times are exponentially distributed [1]. For exponentially distributed processing times and weights that are agreeable in the sense that there exists an ordering such that $w_1 \geq \cdots \geq w_n$ and $w_1\lambda_1 \geq \cdots \geq w_n\lambda_n$, scheduling the jobs in this WSEPT order is optimal [4], while the corresponding deterministic problem is already NP-hard, and in particular, WSPT is not optimal. That is to say, there are examples where the stochastic version with exponentially distributed processing times is computationally easier than the deterministic version of the same problem. Our result shows that with arbitrary weights, the situation is again fundamentally different. Next to this qualitatively new insight, our analysis also sheds light on phenomena in stochastic scheduling which are interesting in their own right.

The paper is organized as follows. In Section 2, we briefly review the worst-case instance presented in [5]. We derive several technical lemmas about scheduling jobs with exponentially distributed processing times in Section 3. Section 4 presents the analysis of the stochastified instance of [5], and finally, Section 5 summarizes our conclusions.

## 2 The Kawaguchi and Kyan Instance

We briefly summarize the instance from [5] that achieves the bound $\frac{1}{2}(1+\sqrt{2})$ for deterministic scheduling, as the instance we will consider is a stochastic variant thereof. Let $n$ be the number of jobs and $m$ the number of machines. Denote the processing time of job $j$ by $p_j$ and its weight by $w_j$. The (deterministic) instance is then given by:

$$\begin{aligned} m &= h + \lfloor (1+\sqrt{2})h \rfloor \\ n &= mk + h \\ p_j = w_j &= 1/k \qquad \text{for} \quad 1 \leq j \leq mk \\ p_j = w_j &= 1+\sqrt{2} \qquad \text{for} \quad mk+1 \leq j \leq mk+h\,. \end{aligned}$$

Here, $h$ denotes an integer, and $k$ is an integer that can be divided by $\lfloor (1+\sqrt{2})h \rfloor$. In particular, $k > h$ and $k$ can be chosen arbitrarily large. Notice that $\frac{w_j}{p_j} = 1$ for all $j$. This means that any list schedule is in fact a WSPT schedule. Let us refer to the first $mk$ jobs as short jobs, and the remaining $h$ jobs as long jobs.

Let $S_L$ be the total weighted completion time of a schedule in which all short jobs are processed first, and $S^*$ be the total weighted completion time of a schedule where the long jobs are processed first. Figure 1 depicts these two schedules. The schedule on the left of Figure 1 has value $S^*$. Here the last jobs

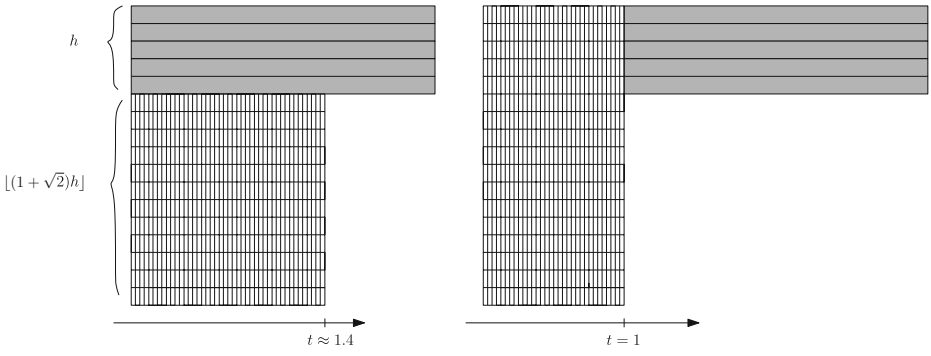



**Fig. 1.** Two different WSPT schedules with values $S^*$ and $S_L$ respectively

of length $1/k$ finish at time $t \approx 1.4$ (for large values of $m$ and $k$). The schedule on the right of Figure 1 has value $S_L$, it finishes the last jobs of length $1/k$ exactly at time 1. In Figure 1 we used $h = 5$ and $k = 32$. It can be verified (see [5]) that $S_L = (1+\sqrt{2})(2+\sqrt{2})h+(m/2)(1+1/k)$ and $S^* = (1+\sqrt{2})^2h+(m/2)(m/\lfloor(1+\sqrt{2})h\rfloor+1/k)$. The ratio $S_L/S^*$ then tends to $(1+\sqrt{2})/2$ as $h \to \infty$ and $k \to \infty$.

## 3 Preliminaries on Jobs with Exponentially Distributed Processing Times

The crucial insight when stochastifying the instance by Kawaguchi and Kyan is the following. The schedule with value $S_L$ is essentially identical to the expected situation in stochastic scheduling. However, the schedule with value $S^*$ has a significantly different realization with exponentially distributed processing times. This is expressed in the following lemmas, where $\lambda$ and $x$ are arbitrary positive parameters. In the following we denote by

$$H_n := \sum_{i=1}^{n} \frac{1}{i}$$

the $n$th harmonic number, with $H_0 := 0$.

The first lemma gives an estimate on expected job completion times for parallel jobs with $P_j \sim \exp(\lambda)$.

**Lemma 1.** *When scheduling in parallel $m$ jobs with i.i.d. exponential processing times $P_j \sim \exp(\lambda)$, the expected number $m(t)$ of machines that are idle at a given time $t$ is bounded as follows,*

$$m(t) \geq \lfloor m\left(1 - e^{-\lambda t}\right) \rfloor .$$

*Proof.* The first completion time is distributed as the minimum of $m$ independent $\exp(\lambda)$ distributions. This is an $\exp(m\lambda)$ distribution, hence it is expected at time $t_1 = \frac{1}{m\lambda}$. After the first job completion, we have $m-1$ jobs remaining. Since the exponential distribution is memoryless, the next completion is expected a time $\frac{1}{(m-1)\lambda}$ later, so $t_2 = \frac{1}{m\lambda} + \frac{1}{(m-1)\lambda}$. By continuing this argument we find that the $i$th job completion is expected at time

$$t_i = \sum_{j=1}^{i} \frac{1}{(m-j+1)\lambda} = \frac{1}{\lambda} \sum_{l=m-i+1}^{m} \frac{1}{l} = \frac{1}{\lambda} \left( H_m - H_{m-i} \right). \tag{1}$$

Note that $m(t_i) = i$, for $i = 1, \ldots, m$. We now use that $H_i \geq \ln(i) + \gamma$ for all $i$, where

$$\gamma := \lim_{i \to \infty} (H_i - \ln i) \approx 0.57721$$

denotes the Euler-Mascheroni constant. Furthermore, $H_i - \ln(i)$ is monotonically decreasing in $i$. Hence we may conclude that

$$t_i \leq \frac{1}{\lambda} \left( \ln(m) + \gamma - \ln(m-i) - \gamma \right) = \frac{1}{\lambda} \ln\left( \frac{m}{m-i} \right). \tag{2}$$

Here, $i$ is the expected number of finished jobs at time $t_i$. Hence, (2) yields

$$m(t_i) = i \geq m(1 - e^{-\lambda t_i}) \tag{3}$$

for $i = 1, 2 \ldots, m$. Together with the fact that $m(t)$ is integer valued, (3) yields

$$m(t) \geq \lfloor m \left( 1 - e^{-t\lambda} \right) \rfloor .$$

for all $t \geq 0$. □

Note that the last job is expected to finish at time $\frac{1}{\lambda}\Theta(\log m)$. Nevertheless, the average expected completion time of the jobs is $1/\lambda$; see also Figure 2 for an illustration.

**Lemma 2.** *Consider $kT$ jobs with i.i.d. processing times $P_j \sim \exp(k)$ and weights $w_j = 1/k$, scheduled on a single machine. Then for all $\varepsilon > 0$ there exists $k$ large enough so that*

$$\mathbb{E}\left[ \textstyle\sum_j w_j C_j \right] \leq \displaystyle\int_0^T t \, dt + \varepsilon .$$

*Proof.* As there is an expected job completion each $1/k$ time steps, one easily calculates that $\mathbb{E}\left[ \sum_j w_j C_j \right] = \frac{1}{2}T^2 + \frac{1}{2k}T$, so for $k \geq \frac{T}{2\varepsilon}$ the claim is true. □

**Lemma 3.** *Let $m(t) \geq 0$ denote the number of available machines at time $t$, and assume $m(t)$ is non-decreasing. When greedily scheduling jobs with i.i.d. processing times $P_j \sim \exp(k)$ and weights $w_j = 1/k$ from time 0 until $T$ on the available machines, for all $\varepsilon > 0$ there exists $k$ large enough so that*

$$\mathbb{E}\left[ \textstyle\sum_j w_j C_j \right] \leq \displaystyle\int_0^T m(t)\, t \, dt + \varepsilon .$$

*Proof.* Let $T_i$ $(i = 0, 1, 2, ..)$ be the times that a new machine becomes available, with $T_0 := 0$. For $k$ large enough, we expect $m(T_i)k(T_{i+1} - T_i)$ jobs to be scheduled between times $T_i$ and $T_{i+1}$. It is straightforward to extend Lemma 2 to this case, which yields for the jobs scheduled between times $T_i$ and $T_{i+1}$,

$$\mathbb{E}\left[\sum_j w_j C_j\right] \leq m(T_i) \int_{T_i}^{T_{i+1}} t\,dt + \varepsilon_i\,.$$

Therefore we get for all jobs,

$$\mathbb{E}\left[\sum_j w_j C_j\right] \leq \sum_i m(T_i) \int_{T_i}^{T_{i+1}} t\,dt + \varepsilon_i \;=\; \int_0^T m(t)\,t\,dt + \sum_i \varepsilon_i\,.$$

So for $\varepsilon = \sum_i \varepsilon_i$ and $k$ accordingly large, the claim is true. □

The next lemma is concerned with the total weighted completion time of short jobs that succeed a set of long jobs.

**Lemma 4.** *Suppose we first schedule $m$ i.i.d. long jobs with processing times $P_j \sim \exp(\lambda)$, followed by $x\,m\,k$ i.i.d. short jobs, with processing times $P_j \sim \exp(k)$ and weights $w_j = 1/k$, where $k$ is large. Let $S_{short}$ be the expected weighted sum of completion times of the short jobs. Then for any $T$ such that $\frac{1}{\lambda}(e^{-\lambda T} - 1) + \frac{(m-1)\,T}{m} \geq x$, and $k$ large enough we have that*

$$S_{short} \leq \int_0^T \left(m(1 - e^{-\lambda t}) - 1\right)\, t\;dt\,. \tag{4}$$

*Proof.* Denote by $m(t)$ the number of machines at time $t$ that are available for processing short jobs, and by $T^*$ the earliest point in time such that we can expect all short jobs to be finished by time $T^*$. Notice that the total expected processing of short jobs equals $x\,m$. Therefore, for $k$ large enough, $T^*$ can be approximated arbitrarily well by the solution of the equation

$$x\,m = \int_0^{T^*} m(t)\,dt\,. \tag{5}$$

With $T^*$ as in (5), Lemma 3 yields that $S_{short} \leq \int_0^{T^*} m(t)t\,dt + \varepsilon$. Now recall that $m(t)$ can be bounded as in Lemma 1. Define

$$f(t) \;=\; m(1 - e^{-\lambda t}) - 1\,. \tag{6}$$

Then $m(t) \geq f(t)$ for all $t \geq 0$. If we require for $T$ that

$$\int_0^T f(t)\,dt \;\geq\; x\,m$$
$$\Leftrightarrow \frac{1}{\lambda}\,(e^{-\lambda T} - 1) + \frac{(m-1)\,T}{m} \;\geq\; x\,,$$

then $x\,m = \int_0^{T^*} m(t)\,dt \leq \int_0^T f(t)\,dt$. We therefore conclude that

$$\int_0^{T^*} m(t)\,t\,dt < \int_0^T f(t)\,t\,dt\,, \tag{7}$$

because $m(t) \geq f(t)$, and $m(t)$ is a step function while $f(t)$ is continuous. Intuitively, the expression $\int_0^T f(t)\,t\;dt$ equals the total weighted sum of completion times for infinitesimally small jobs (i.e., when $k \to \infty$), with total expected processing at least $x\,m$, scheduled on a set of "machines" that become available no earlier than $m(t)$. We finally conclude from (7) that

$$S_{short} \leq \int_0^{T^*} m(t)\,t\,dt + \varepsilon \leq \int_0^T f(t)\,t\,dt\,,$$

because $\varepsilon$ can be chosen arbitrarily small for $k$ large enough. □

A variation of this lemma will be used later in the analysis. Notice that the technical condition on $T$ as stated in Lemma 4 only makes sure that all short jobs can be processed by time $T$ when the machine availability is governed by $f(t)$ rather than $m(t)$. The same approach will be used also in Section 4. Finally, we make a statement about scheduling a block of (short) jobs.

**Lemma 5.** *Suppose we schedule $x\,m\,k$ i.i.d. short jobs with processing times $P_j \sim \exp(k)$ greedily on $m$ machines. Then the average expected machine completion time equals $x$, and for any $\delta > 0$ there exists $k$ large enough such that the earliest expected machine completion time is at time $t \geq x - \delta$.*

*Proof.* The claim about the average expected machine completion time is clear, because the total expected processing is $x\,m$. For the second claim, consider the first time, say $t$, that a machine runs out of jobs. Then there are $m-1$ jobs still in process. We know from Lemma 1 that the last machine that runs out of jobs is expected to be at time $t + \sum_{i=1}^{m-1} \frac{1}{i\,k}$. For $m$ large enough, we have $\sum_{i=1}^{m-1} \frac{1}{i\,k} \leq \frac{1}{k}\,[\ln(m) + \gamma]$. So for $k \geq (m-1)/(\delta(\ln(m)+\gamma))$, the last machine completion time is expected no later than $t + \delta/(m-1)$. Now the claim follows, as the average machine completion time is $x$. □

## 4 The Stochastic Instance

Even though other instances may lead to comparable results, we find it instructive to consider the stochastic analogue of the instance presented by Kawaguchi and Kyan [5]. Indeed, it turns out that the analyses for such instances use identical arguments, the core of which is represented by the lemmas given in Section 3.

We keep all parameters the same as in Section 2, except that the processing times of long jobs will be $P_j \sim \exp(1/(1+\sqrt{2}))$, and the processing times of short jobs will be $P_j \sim \exp(k)$. So the expected processing times of long and short jobs are identical to the deterministic processing times in [5].

### 4.1 Intuition about the Schedules

Suppose we start all long jobs first and then fill up the remaining machines with short jobs. By Lemma 1 we expect the $i^{th}$ long job to finish at time:

$$t_i = \sum_{j=1}^{i} \frac{1+\sqrt{2}}{h-j+1} \tag{8}$$

Therefore, we expect the last short job to be completed significantly earlier than in the deterministic case. For a finite number of machines, the schedule will look like depicted in Figure 2. The crucial point is that the average expected time that

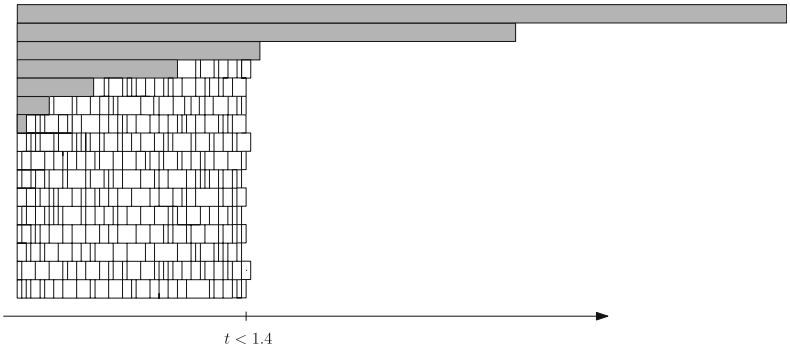



**Fig. 2.** Schedule with value $S^*$: long jobs starting at time 0

machines finish processing short jobs will be smaller than in the deterministic case. This happens because many long jobs finish much earlier, and the late finishing of few long jobs doesn't matter for the short jobs. Hence, the contribution of the short jobs will decrease when compared to the deterministic case.

Suppose on the other hand that we first start all the short jobs. The set of short jobs is not likely to produce the ideal rectangle as it did in the deterministic case. However, as suggested by Lemma 5 the gap between the time the first machine runs out of short jobs and the time the last machine runs out of short jobs can be made arbitrarily small, by letting $k$, the inverse of the expected processing time of short jobs, be large. The crucial point is that, in this situation, the expected cost of the schedule is almost the same as the cost in the deterministic case.

### 4.2 Lower Bound on Performance of WSEPT

Let $S^*$ denote the objective value $\mathbb{E}\left[\sum_j w_j C_j\right] = \sum_j w_j \mathbb{E}[C_j]$ for the case when we first schedule all long jobs. Similarly, let $S_L$ denote the objective value for the schedule that starts long jobs only when there is no short job left to be scheduled. $S^*$ is in fact optimal, whereas $S_L$ is the worst case, but this is inessential. Both are in fact WSEPT, hence the ratio $S_L/S^*$ is a lower bound for

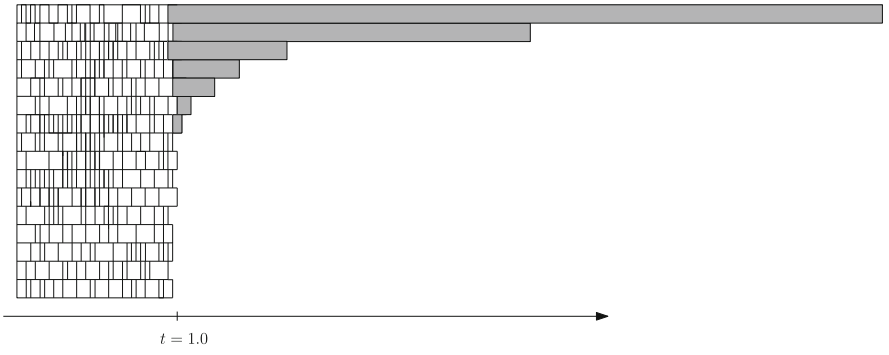



**Fig. 3.** Schedule with value $S_L$: long jobs starting only after short jobs

the approximation ratio of the WSEPT rule in stochastic machine scheduling with exponentially distributed processing times.

We choose $h$ sufficiently large, and $k$, a multiple of $\lfloor(1+\sqrt{2})h\rfloor$, we may choose arbitrarily large in comparison to $h$ (i.e., $k >> h$). In fact, we can make the choice of these two parameters in such a way that all our technical lemmas from Section 3 do apply.

**The optimal case, $S^*$.** We split $S^*$ up into the contribution of the long jobs $S^*_{long}$ and the contribution of the short jobs $S^*_{short}$. So

$$S^* = S^*_{long} + S^*_{short} \tag{9}$$

*The value $S^*_{long}$.* We start all $h$ long jobs at time 0. Their expected completion time is $1+\sqrt{2}$ each. Hence the contribution of the long jobs is simply given by

$$S^*_{long} \;=\; h(1+\sqrt{2})^2\,, \tag{10}$$

which is actually the same as in the deterministic case.

*The value $S^*_{short}$:* This is a bit more complicated to calculate. We expect the short jobs to be located in the light and dark gray areas, $\mathcal{R}$ and $\mathcal{B}$ respectively, as depicted in Figure 4.

The expected total processing of short jobs $\mathcal{B}$ that fit in the light gray area is given by

$$\mathcal{B} = \int_0^T (m-h)\,dt$$

According to (6) in the proof of Lemma 4, the number of finished long jobs at time $t \geq 0$ is at least:

$$f(t) \;=\; h(1-e^{-t/(1+\sqrt{2})}) - 1\,.$$

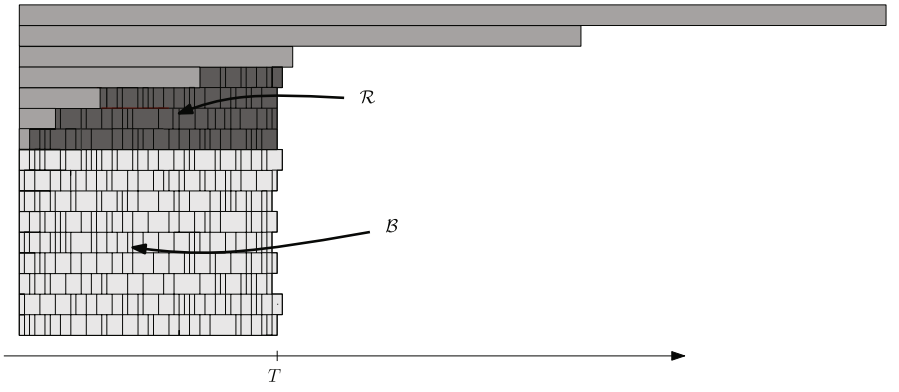



**Fig. 4.** For T large enough, short jobs fit in the light and dark gray areas $\mathcal{R}$ and $\mathcal{B}$

Therefore, the expected total processing of short jobs $\mathcal{R}$ that fit in the dark gray area is bounded by

$$\mathcal{R} \geq \int_0^T f(t)\,dt$$

We want to find a value for $T$ such that all short jobs are expected to be finished by $T$, i.e. $\mathcal{B} + \mathcal{R} \geq m$. We have not attempted to solve this equation analytically, but one can easily check that

$$T = 1.2933 \tag{11}$$

suffices.

Then $S^*_{short}$, the expected weighted sum of completion times for all $mk$ short jobs, can be bounded similarly as in Lemma 4. We now find, for $h$ and $k$ sufficiently large,

$$S^*_{short} \leq \int_0^T (m-h)\,t\,dt \;+\; \int_0^T f(t)t\,dt\,. \tag{12}$$

With (11) and (12) we can calculate

$$S^*_{short} \leq\; 2.266h - 0.836\,. \tag{13}$$

Combining (10) and (13) gives

$$S^* \;=\; S^*_{long} \;+\; S^*_{short} \leq (1+\sqrt{2})^2 h + 2.266h - 0.836\,. \tag{14}$$

**The worst case, $S_L$.** Now we switch to the case where we first schedule all the short jobs. Again split the objective value into the two parts contributed by the short and long jobs, respectively.

$$S_L = S_L^{short} + S_L^{long}\,.$$

*The value $S_L^{short}$:* We have $m$ machines working on $mk$ jobs with processing times $P_j \sim \exp(k)$. According to Lemma 5, on average a machine is expected to finish with these jobs at time 1, and for any $\delta > 0$, we can find $k$ large enough so that we expect all machines to be filled with short jobs at least until time $1 - \delta$. Hence, we conclude that the average expected completion time of a short job is arbitrarily close to $1/2$. Therefore, for any $\varepsilon > 0$, there is $k$ large enough so that

$$S_L^{short} \geq \frac{m}{2} - \varepsilon/2\,. \tag{15}$$

*The value $S_L^{long}$:* Remember that the schedule is expected to look like depicted in Figure 3. Using Lemma 5 again, we know that long jobs are expected to start no earlier than $1 - \delta$. So by assuming they all start at this time, we get a lower bound for their completion times (and also for $S_L^{long}$). If all long jobs start at $1 - \delta$, the average expected completion time is $2 - \delta + \sqrt{2}$. Multiplying this by the weight and summing over all $h$ jobs, we may conclude that for any $\varepsilon > 0$ there is $k$ large enough so that

$$S_L^{long} \geq (2 + \sqrt{2})\,(1 + \sqrt{2})h - \varepsilon/2\,. \tag{16}$$

With (15) and (16) we now have

$$S_L = S_L^{short} + S_L^{long} \geq \frac{m}{2} + (2 + \sqrt{2})\,(1 + \sqrt{2})h - \varepsilon\,. \tag{17}$$

**The ratio.** Finally, let $\alpha$ be the approximation ratio of Smith's rule for exponentially distributed processing times. Then

$$\alpha \geq \frac{S_L}{S^*}\,.$$

Remember that $m = h + \lfloor(1 + \sqrt{2})h\rfloor$. Now for carefully chosen $k >> h$, and taking $h \to \infty$, equations (14) and (17) give

$$\frac{S_L}{S^*} \geq \frac{m/2 + (2 + \sqrt{2})\,(1 + \sqrt{2})h - \varepsilon}{(1 + \sqrt{2})^2 h + 2.266h - 0.836} > 1.229\,.$$

So we conclude that $\alpha > 1.229$. Note that this is strictly larger than the approximation ratio for WSPT in the the deterministic case, which is 1.207.

**Optimizing the parameters.** What remains to be done is to optimize over the parameters of the instance to improve the obtained lower bound. To that end, recall that the considered instance has $h$ long jobs and $m = h + \lfloor(1 + \sqrt{2})h\rfloor \approx 3.4h$ machines, and long jobs have processing times $P_j \sim \exp(\frac{1}{1+\sqrt{2}}) \approx \exp(0.41)$. However, these parameters are optimized for the deterministic instance. Taking slightly more long jobs, namely by letting $m = 2.3h$, with somewhat shorter processing times, namely $P_j \sim \exp(0.56)$, we obtain a ratio of at least 1.2436, which proves Theorem 1.

## 5 Conclusion

Note that the numerical calculations have been performed using Mathematica. We also found instances (not discussed in this paper) —with comparable building blocks and features— where WSPT is always optimal for the deterministic case, while WSEPT is not optimal for the stochastic counterpart with exponentially distributed processing times. In conclusion, small improvements in the ratio 1.243 might be possible. Yet, the upper bound $(2-1/m)$ seems out of reach. This leaves the question to improve the upper bound on the performance guarantee for WSEPT; in that respect, it is interesting to note that the analysis of [7] does not explicitly exploit the exponential distribution; it is valid in more generality.

## Acknowledgements

The second and third author thank the organizers of the 2010 Dagstuhl Seminar on Scheduling, which was a starting point for this research.

## References

1. Bruno, J.L., Downey, P.J., Frederickson, G.N.: Sequencing tasks with exponential service times to minimize the expected flowtime or makespan. Journal of the ACM 28, 100–113 (1981)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
3. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5, 287–326 (1979)
4. Kämpke, T.: On the optimality of static priority policies in stochastic scheduling on parallel machines. Journal of Applied Probability 24, 430–448 (1987)
5. Kawaguchi, T., Kyan, S.: Worst case bound on an LRF schedule for the mean weighted flow-time problem. SIAM Journal on Computing 15, 1119–1129 (1986)
6. Möhring, R.H., Radermacher, F.J., Weiss, G.: Stochastic scheduling problems I: General strategies. ZOR - Zeitschrift für Operations Research 28, 193–260 (1984)
7. Möhring, R.H., Schulz, A.S., Uetz, M.: Approximation in stochastic scheduling: The power of LP-based priority policies. Journal of the ACM 46, 924–942 (1999)
8. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems, 2nd edn. Prentice-Hall, Upper Saddle River (2002)
9. Rothkopf, M.H.: Scheduling with random service times. Management Science 12, 703–713 (1966)
10. Skutella, M., Woeginger, G.J.: A PTAS for minimizing the total weighted completion time on identical parallel machines. Mathematics of Operations Research 25, 63–75 (2000)
11. Smith, W.E.: Various optimizers for single-stage production. Naval Research Logistics Quarterly 3, 59–66 (1956)
12. Uetz, M.: When greediness fails: examples from stochastic scheduling. Operations Research Letters 31, 413–419 (2003)
13. Uetz, M.: Algorithms for Deterministic and Stochastic Scheduling. PhD thesis, Institut für Mathematik, Technische Universität Berlin, Germany (2001); published by Cuvillier Verlag, Göttingen, Germany (2002)

# Approximating Survivable Networks with Minimum Number of Steiner Points

Lior Kamma* and Zeev Nutov

The Open University of Israel
lior.kamma@gmail.com, nutov@openu.ac.il

**Abstract.** Given a graph $H = (U, E)$ and connectivity requirements $r = \{r(u, v) : u, v \in R \subseteq U\}$, we say that $H$ satisfies $r$ if it contains $r(u, v)$ pairwise internally-disjoint $uv$-paths for all $u, v \in R$. We consider the Survivable Network with Minimum Number of Steiner Points (SN-MSP) problem: given a finite set $V$ of points in a normed space $(M, \|\cdot\|)$, and connectivity requirements, find a minimum size set $S \subset M - V$ of additional points, such that the unit disc graph induced by $V \cup S$ satisfies the requirements. In the (node-connectivity version of the) Survivable Network Design Problem (SNDP) we are given a graph $G = (V, E)$ with edge costs and connectivity requirements, and seek a min-cost subgraph $H$ of $G$ that satisfies the requirements. Let $k = \max_{u,v \in V} r(u, v)$ denote the maximum connectivity requirement. We will show a natural transformation of an SN-MSP instance $(V, r)$ into an SNDP instance $(G = (V, E), c, r)$, such that an $\alpha$-approximation for the SNDP instance implies an $\alpha \cdot O(k^2)$-approximation algorithm for the SN-MSP instance. In particular, for the most interesting case of uniform requirement $r(u, v) = k$ for all $u, v \in V$, we obtain for SN-MSP the ratio $O(k^2 \ln k)$, which solves an open problem from [3].

**Keywords:** Sensor networks, Unit-disc graphs, Node-connectivity, Approximation algorithms.

## 1 Introduction

### 1.1 Problem Definition and Motivation

Network design problems require finding a minimum cost (sub-)network that satisfies prescribed properties, often connectivity requirements. Classic examples with $0, 1$ connectivity requirements are: Shortest Path, Edge-Cover, Minimum Spanning Tree, Minimum Steiner Tree/Forest, and others. Examples of problems with high connectivity requirements are: Min-Cost $k$-Flow, Edge-Multicover, $k$-Edge/Node-Connected Spanning Subgraph, Steiner Network, and others. Such problems were studied extensively for edge/node-connectivity and edge/node-costs, see [1, 15, 13, 4, 17, 5, 6, 9, 20, 19, 18] for only a small sample of papers in the area.

* Part of this work was done as a part of author's M.Sc. Thesis at The Open University of Israel.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 154–165, 2011.

We consider node-connectivity only. Let $H = (U, E)$ be an undirected graph, possibly with parallel edges. For $u, v \in U$, let $\kappa_H(u, v)$ denote the maximum number of pairwise internally-disjoint $uv$-paths in $H$. Given non-negative integer connectivity requirements $r = \{r(u, v) : u, v \in R \subseteq U\}$, we say that $H$ satisfies $r$ if $\kappa_H(u, v) \geq r(u, v)$ for all $u, v \in R$. In the Survivable Network Design Problem (SNDP) we are given a graph $G = (V, E)$ with edge-costs $\{c_e : e \in E\}$ and *node-connectivity* requirements $r = \{r(u, v) : u, v \in R \subseteq V\}$. The goal is to find a minimum cost subgraph $H$ of $G$ that satisfies $r$. SNDP problems were extensively studied, especially the $k$-Connected Subgraph problem when $r(u, v) = k$ for all $u, v \in V$, see [4, 17, 20].

Related SNDP problems have strong motivation in wireless networks [3, 2, 7, 21, 22, 16]. One way to increase connectivity is to assign to each terminal a high transmission power to satisfy the connectivity requirements. However, the power needed to transmit through a distance $d$ might be proportional to $d^4$ [8]. Since energy budget is a primary constraint in wireless networks design, one may prefer adding sensors rather than increasing power. Thus the problem of adding a minimum number of (identical) sensors to increase the connectivity arises. Note that in this setting, reliability of the network is measured by node-connectivity, as it models sensor failures. For further discussion and motivation we refer the reader to [3] and the references therein.

We consider SNDP problems on unit-disc graphs in normed spaces, where the goal is to select a minimum number of Steiner Points (transmitters) to satisfy the requirements between the terminals. Namely, given a normed space, $(M, \|\cdot\|)$, a finite set $V \subseteq M$, and connectivity requirements $\{r(u, v) \mid u, v \in V\}$, we wish to adjust the network to satisfy the requirements between the terminals by adding a minimum number of transmitters (Steiner points).

**Definition 1.** *Given a finite set of points $V \subset M$ in a metric space $(M, d)$, the* unit disc graph $G[V]$ induced by $V$ *has node set $V$ and edge set $\{uv : 0 < d(u, v) \leq 1,\ u, v \in V\}$.*

Formally, we consider the following problem:

Survivable Network with Minimum Number of Steiner Points (SN-MSP)

*Instance:* A finite set $V$ of points in a metric space $(M, d)$ and pairwise connectivity requirements $r = \{r(u, v) : u, v \in R \subseteq V\}$.

*Objective:* Find a minimum size set of points $S \subset M - V$ such that $G[V \cup S]$ satisfies $r$.

An important special case is that of *uniform requirements*, when $r(u, v) = k$ for all $u, v \in V$. We call this particular case $k$-Connectivity with Minimum Number of Steiner Points ($k$-C-MSP). We also consider the following types of requirements.

- *Rooted requirements:* there is $s \in V$ so that $r(u, v) > 0$ implies $u = s$ or $v = s$; in *rooted-uniform* requirements $r(s, v) = k$ for all $v \in V - \{s\}$.
- *Subset uniform requirements:* there is $R \subseteq V$ such that $r(u, v) = k$ for all $u, v \in R$, and $r(u, v) = 0$ otherwise; ($k$-C-MSP is the case of *uniform requirements* when $R = V$).

### 1.2 Our Results

Given an instance of SNDP or of SN-MSP, let $k = \max_{u,v \in R} r(u,v)$ denote the maximum connectivity requirement. As in practical networks $k$ is rather small, we focus on obtaining approximation ratios that depend on $k$ only. For $k = 1$, SN-MSP with uniform requirements is the Steiner Tree with Minimum Number of Steiner Points problem (ST-MSP). In the Euclidean plane, this problem admits a 2.5-approximation algorithm [7]. On graphs with unit edge lengths ST-MSP includes the Set-Cover problem [15], and thus has an $\Omega(\ln |V|)$-approximation threshold. Hence for SN-MSP one cannot expect in arbitrary metric spaces a ratio that depends on $k$ only. We will consider instances of SN-MSP defined on a *normed space* $(M, \|\cdot\|)$, when the metric $d$ is induced by the norm $\|\cdot\|$.

One can easily reduce SN-MSP to an SNDP variant with unit weights on the nodes rather than with costs on the edges; this reduction invokes a constant loss factor in the approximation ratio. In this reduction however, uniform requirements in SN-MSP instance become subset uniform requirements in the SNDP instance. The currently best known ratios for SNDP with node weights are: $O(k^2 \log |V|)$ for rooted requirements, $O(k^3 \log |V|)$ for subset uniform requirements, and $O(k^4 \log^2 |V|)$ for general requirements [18]. The factor $O(\log |V|)$ in these ratios is unavoidable even for $k = 1$, as even for $k = 1$ the problem generalizes the Set-Cover problem [15].

Obtaining for $k$-C-MSP in $\mathbb{R}^2$ an approximation ratio that depends on $k$ only was posed as an open problem in [3]. We will prove a much more general result. Our ratios are expressed in terms of $k$ and a parameter $\Delta$ that depends on the normed space. Let $\Delta = \Delta(M, \|\cdot\|)$ be the minimum number so that for any $V \subseteq M$ contained in a unit ball, $G[V]$ has a dominating set of size at most $\Delta$. It is known that $\Delta = 5$ in $\mathbb{R}^2$ and $\Delta = 11$ in $\mathbb{R}^3$. In [23] it is proved that for $M = \mathbb{R}^\ell$ with the norm $\|(x_1, x_2, \ldots, x_\ell)\|_p = \left(\sum_{i=1}^{\ell} |x_i|^p\right)^{1/p}$, $\Delta$ is at most the Hadwiger number of the unit ball; (The Hadwiger number of an open convex set $X$ in a normed space $M$ is the maximal number of disjoint translations of $X$ which share a boundary point with $X$); thus, for the Euclidean space $\mathbb{R}^\ell$, $\Delta \leq 2^{0.401\ell(1+o(1))}$, by [14]. Let $\rho(k) = (\Delta + 3)k^2 + 7k + 2$. Our main result is:

**Theorem 1.** *An $\alpha$-approximation algorithm for* SNDP *(on multigraphs) implies an $\alpha \cdot \rho(k)$-approximation algorithm for* SN-MSP*, and this is so also for subset uniform, uniform, rooted, rooted subset uniform, and rooted uniform requirements.*

In SNDP problems, the input graph is usually assumed to be simple, while in Theorem 1 it may have parallel edges. One novelty in our approach is considering SNDP on multigraphs, and proving that the best known ratios for SNDP with different requirement types remain the same on multigraphs. Specifically, we will prove the following statement in Sect. 2.

**Lemma 1.** *There is an approximation ratio preserving reduction from* SNDP *on multigraphs to* SNDP *on simple graphs, if the approximation ratios do not depend on* $|V|$*. The reduction is requirement type preserving for uniform, rooted, and subset uniform requirements. In the case of rooted uniform requirements, the problem on multigraphs admits a* 2*-approximation algorithm.*

The best known values of $\alpha$ are as follows. For $k$-Connected Subgraph on simple graphs, an $O(\log k)$-approximation algorithm for $k = O(\sqrt{n})$ [4] was obtained long time ago. This ratio was recently extended to almost all values of $n, k$ in [20]; specifically, the ratio in [20] is $O(\ln \frac{|V|}{|V|-k} \cdot \ln k)$ (which is $O(\ln k)$ unless $k = |V| - o(|V|)$). For other SNDP problems, the currently best known approximation ratios are: 2 for rooted uniform requirements [11], $O(k \ln k)$ for rooted requirements [18], $O(k^2 \ln k)$ for subset uniform requirements [18], and $O(k^3 \ln |R|)$ for general requirements [5].

By substituting the currently best known values of $\alpha$ in Theorem 1, we obtain:

**Corollary 1.** $k$-C-MSP *admits an approximation ratio of* $O\left(\ln \frac{|V|}{|V|-k} \cdot \ln k\right) \cdot \rho(k) = O(k^2 \ln k)$*. Other* SN-MSP *problems admit the following approximation ratios:* $2\rho(k) = O(k^2)$ *for rooted uniform requirements,* $O(k \ln k) \cdot \rho(k) = O(k^3 \ln k)$ *for rooted requirements, and* $O(k^2 \ln k) \cdot \rho(k) = O(k^4 \ln k)$ *for* subset uniform requirements*,*

Corollary 1 solves an open problem of Bredin, Demaine, Hajiaghayi, and Rus [3], by giving the first non-trivial approximation algorithm for $k$-C-MSP with $k \geq 2$. In [3] the problem of adding a minimum set $S$ of Steiner points so that the entire graph $G[V \cup S]$ is $k$-connected was considered (note that in $k$-C-MSP we require $k$-connectivity only between the set $V$ of terminals). For this problem in $\mathbb{R}^2$, [3] gave a reduction that invokes a loss of $O(k^4)$. They also conjectured that for $k$-C-MSP their reduction can be used to reduce the instance to an SNDP instance with subset uniform requirements, thus leading to an approximation ratio that depends on $k$ only, provided existence of such an approximation for SNDP with subset uniform requirements. Even if this conjecture was proved, it leads to ratio $O(k^4 \cdot k^2 \log k)$, which is much worse than the ratio $O(k^2 \cdot \log k)$ proved in this paper. The reason is not only the worse reduction fee, but also since [3] reduces instances with uniform requirements into instances with subset uniform requirements, while our reduction preserves the requirements type; consequently, our result for $k$-C-MSP rely on algorithms for $k$-Connected Subgraph only (e.g. [4], published few years before [3]), and not on recently discovered algorithms for SNDP with subset uniform requirements [18]. Furthermore, our algorithm works for arbitrary normed spaces, and for various connectivity requirement types. However, in the proof of our main result we do use some ideas from [3].

We also note that Theorem 1 together with the $O(k^3 \ln |R|)$-approximation algorithm for SNDP of [5] implies the ratio $O(k^3 \ln |R|) \cdot \rho(k) = O(k^5 \ln |R|)$ for SN-MSP with arbitrary requirements. But in this case we conjecture that a ratio that depends on $k$ only can be achieved.

## 2 Proof of the Main Result

We start by proving Lemma 1, which is restated here for the convenience of the reader.

**Lemma 1.** *There is an approximation ratio preserving reduction from* SNDP *on multigraphs to* SNDP *on simple graphs, if the approximation ratios do not depend on* $|V|$*. The reduction is requirement type preserving for uniform, rooted, and subset uniform requirements. In the case of rooted uniform requirements, the problem on multigraphs admits a* 2*-approximation algorithm.*

*Proof.* First we consider instances with non-uniform requirements. Given an SNDP instance (with parallel edges), insert a new node into every edge, and divide (arbitrarily) the cost of the edge between the corresponding two new edges. Clearly, the obtained graph is simple. It is easy to see that an $\alpha$-approximation for the modified instance implies an $\alpha$-approximation for the original instance and that this transformation is requirement type preserving for subset uniform, rooted, and rooted subset uniform requirements. It remains therefore to consider uniform and rooted uniform requirements.

We now consider the case of uniform requirements, when feasible solutions are $k$-connected spanning subgraphs of $G$. Let $H = (V, E)$ be a minimally $k$-connected multi-graph (so $H - e$ is not $k$-connected for every $e \in E$). We claim that if $|V| \geq k + 1$ then $H$ is simple (thus we can keep for every maximal set of pairwise parallel edges of $G$ only the cheapest one), and if $|V| \leq k$ then $H$ has exactly $k + 2 - |V|$ edges between every pair of it nodes (thus an optimal solution is found by taking the $k + 2 - |V|$ cheapest edges in $G$ between every pair of nodes). Assume that $|V| \geq k + 1$. Then the simple underlying graph $H'$ of $H$ is $k$-connected by the theorem of Whitney (c.f. [10]): *If* $\kappa_{H'}(u, v) \geq k$ *for every* $u, v \in V$ *so that* $uv \notin E'$*, then* $H'$ *is* $k$*-connected.* This holds in our case, since $k$ pairwise internally-disjoint $uv$-paths in $H$ have no parallel edges. If $|V| \leq k$, then note that if $H$ has exactly $k + 2 - |V|$ edges between every pair of its nodes then $H$ is $k$-connected. Hence it is sufficient to prove that there are at least $k + 2 - |V|$ edges between every two nodes of $H$. To see this, consider a set of $k$ internally disjoint $uv$-paths in $H$. At most $|V| - 2$ of these paths may not be edges between $u, v$, thus at least $k - (|V| - 2)$ of these paths are edges between $u, v$.

Finally, for rooted uniform requirements, we note that the existing 2-approximation algorithm in [11] does not have the restriction that $G$ is simple, and hence works also for multi-graphs.

To prove Theorem 1, we will prove the following statement.

**Lemma 2.** *There exists a polynomial time algorithm that, given an instance* $V, r$ *of* SN-MSP*, constructs an instance* $G = (V, E), c, r$ *of* SNDP *so that: any solution of cost* $C$ *to* SNDP *can be converted in polynomial time to a solution of size* $\leq C$ *to* SN-MSP*, and for every solution* $S$ *to* SN-MSP *there exists a*

*solution $J$ of cost $\leq |S| \cdot \rho(k)$ to* SNDP. *Furthermore, the construction preserves the requirement type (subset uniform, uniform, rooted, rooted subset uniform, and rooted uniform).*

Theorem 1 easily follows from Lemma 2. To see this, consider the following approximation algorithm for SN-MSP:

**Algorithm 1.** *Approximation algorithm for* SN-MSP

**Approximate-SN-MSP**$(V, r = \{r(u,v) : u, v \in R \subseteq V\})$

1. Construct the SNDP instance $(G = (V, E), c, r)$ as in Lemma 2 from the SN-MSP instance $(V, r)$.
2. Compute a subgraph $J \subseteq G$ satisfying $r$ using an $\alpha$-approximation algorithm.
3. Construct from $J$ a feasible solution $S$ to SN-MSP.

Lemma 2 ensures that the algorithm runs in polynomial time and computes a feasible solution $S$ to the SN-MSP instance. We prove the approximation ratio. Let $J^*$ be a minimum cost subgraph of $G$ satisfying $r$, and let $S^*$ be a minimum size set of points so that $G[V \cup S^*]$ satisfies $r$. Then

$$|S| \leq c(J) \leq \alpha \cdot c(J^*) \leq \alpha \cdot |S^*| \cdot \rho(k)\,.$$

The second inequality is since $J$ is computed using an $\alpha$-approximation algorithm, and the last inequality is by Lemma 2.

In the rest of this section we prove Lemma 2.

**Definition 2.** *Given a finite set of points $V \subset M$ and an integer $k \geq 1$, the graph $K_V$ is obtained by connecting every $u, v \in V$ by $k$ parallel edges, one of cost $\lceil d(u,v) \rceil - 1$ the others of cost $\lceil d(u,v) \rceil$.*

Clearly, given an SN-MSP instance, $(V, r)$, the graph $K_V$ with the corresponding edge costs $c$ can be constructed in polynomial time. The triple $(K_V, c, r)$ will serve as the SNDP instance guaranteed in Lemma 2. The construction preserves the requirement types listed in Lemma 2. Let $J$ be a subgraph of $K_V$. Let $u, v \in V$ be adjacent in $J$ by $j + 1 \leq k$ edges. Place $\lceil d(u,v) \rceil - 1$ new points uniformly on the line segment between $u$ and $v$, dividing the segment into $\lceil d(u,v) \rceil$ subsegments, each of length $\frac{d(u,v)}{\lceil d(u,v) \rceil} \leq 1$. Recall that since $M$ is a normed space, and thus also a linear space, this can be easily done, and in fact, for $1 \leq i \leq \lceil d(u,v) \rceil - 1$, the $i$th point is of the form

$$\left(1 - \frac{i}{\lceil d(u,v) \rceil}\right) u + \frac{i}{\lceil d(u,v) \rceil} v$$

On each subsegment, place uniformly $j$ new points in a similar fashion. Let $S(u,v)$ be the set of added points. Denote by $S(J)$ the union of $S(u,v)$ over all adjacent pairs $u, v \in V$. The following statement is straightforward.

**Proposition 1.** *$|S(J)| \leq c(J)$ holds for any subgraph $J$ of $K_V$. Furthermore, if $H = G[V \cup S(J)]$ is the unit disc graph induced by $V \cup S(J)$ then $\kappa_H(u,v) \geq \kappa_J(u,v)$ for all $u, v \in V$.*

Clearly, $S(J)$ can be computed from $J$ in polynomial time. This proves all parts of Lemma 2, except the one stating that for every solution $S$ to SN-MSP there exists a solution $J$ of cost $c(J) \leq |S| \cdot \rho(k)$ to SNDP; this will be proved in the rest of this section.

For a subset $C$ of nodes of a graph $G$ let $\Gamma_G(C)$ denote the set of neighbors of $C$ (outside of $C$) in $G$. We need the following lemma on connectivity of graphs.

**Lemma 3.** *Let $V$ be a subset of nodes of a graph $G$, let $k$ be an integer, and let $C$ be a connected component of $G - V$. Let $J_C$ be a set of new edges on $\Gamma_G(C)$ such that the following holds:*

(i) *If $|\Gamma_G(C)| \leq k$ then $J_C$ has $\min\{\ell_{uv}, k - |I_{uv}|\}$ $uv$-edges for any $u, v \in \Gamma_G(C)$, where $I_{uv}$ is the set of $uv$-edges in $G$ and $\ell_{uv}$ is the maximum number of internally disjoint $uv$-paths in the subgraph of $G$ induced by $\{u, v\} \cup C$.*
(ii) *If $|\Gamma_G(C)| \geq k+1$ then the graph induced by $\Gamma_G(C)$ in $G + J_C$ is $k$-connected.*

*Let $J = G - C + J_C$. Then $\kappa_J(u,v) \geq \min\{\kappa_G(u,v), k\}$ for all $u, v \in V$.*

*Proof.* The case $|\Gamma_G(C)| \leq k$ easily follows from the following construction. Let $u, v \in G - C$. Given a set $\Pi$ of at most $k$ internally disjoint $uv$-paths in $G$, for every $P \in \Pi$ do the following. For every maximal $u'v'$-subpath of $P$ that visits $C$ and has all its internal nodes in $C$, replace this subpath by a $u'v'$-edge $e$ not used by any other path in $\Pi$. Such $e$ is chosen to be an edge of $G$ if $\{u', v'\} \neq \{u, v\}$ and $I_{u'v'} \neq \emptyset$ or if $\{u', v'\} = \{u, v\}$ and $\min\{\ell_{uv}, k - |I_{uv}|\} = 0$. Otherwise, $e$ is a new edge added to $G$. This gives a set of $|\Pi|$ internally disjoint $uv$-paths that do not visit $C$. Since the paths in $\Pi$ are internally disjoint, the set of edges added to $G$ may have parallel edges only between $u$ and $v$, and by the construction, the number of $uv$-edges added, if any, can be at most $\min\{\ell_{uv}, |\Pi| - |I_{uv}|\} \leq \min\{\ell_{uv}, k - |I_{uv}|\}$.

Now suppose that $|\Gamma_G(C)| \geq k+1$, so $\Gamma_G(C)$ induces in $G + J_C$ a $k$-connected graph. Let $u, v \in G - C$. Let $I_{uv}$ be a set of $uv$-edges in $J$. Let $A$ be a minimum size subset of nodes of $J$ so that $J - (A + I_{uv})$ has no $uv$-path. By Menger's Theorem $\kappa_J(u,v) = |A| + |I_{uv}|$. Thus if $|A| + |I_{uv}| \geq k$ then $\kappa_J(u,v) \geq k \geq \min\{\kappa_G(u,v), k\}$. We claim that if $|A| + |I_{uv}| \leq k - 1$ then $G - (A + I_{uv})$ has no $uv$-path, hence by Menger's Theorem $\kappa_J(u,v) = |A| + |I_{uv}| \geq \kappa_G(u,v)$. Suppose to the contrary that $G - (A + I_{uv})$ has a $uv$-path $P$. Going along $P$ from $u$ to $v$, let $u'$ be the first and $v'$ the last node in $\Gamma_G(C)$; such $u', v'$ exist since $P$ must contain at least one node from $C$, as $P$ is not a $uv$-path in $J - (A + I_{uv})$. As $J$ has $k$ internally disjoint $u'v'$-paths and $|A| + |I_{uv}| \leq k - 1$, the graph $J - (A + I_{uv})$ has at least one $u'v'$-path $P'$. Replacing the $u'v'$-subpath of $P$ by $P'$ gives a $uv$-path in $J - (A + I_{uv})$, contradicting the definition of $A$.

Let $S$ be a feasible solution to an SN-MSP instance, so $G = G[V \cup S]$ satisfies $r$. The key step in constructing a solution $J$ to SNDP of cost $c(J) \leq |S| \cdot \rho(k)$

is replacing every connected component $C$ of $G - V$ by an edge set $J_C$ as in Lemma 3. Obviously, $\Gamma_G(C) \subseteq V$, and thus $J_C \subseteq K_V$. The following lemma shows that there exists such $J_C$ of low cost.

**Lemma 4.** *For every connected component $C$ of $G-V$ there exists a subset $J_C$ of edges of $K_V$ as in Lemma 3 of cost $c(J_C) \leq \rho(k) \cdot |C|$.*

The proof of Lemma 4 is somewhat long, so we prove it after the following corollary, which easily implies the last part of Lemma 2.

**Corollary 2.** *Let $\mathcal{C}$ be the set of connected components of $G-V$. For $C \in \mathcal{C}$ let $J_C$ be as in Lemma 4. Then $J = G - S + (\bigcup_{C \in \mathcal{C}} J_C)$ is a subgraph of $K_V$ of cost $c(J) \leq \rho(k) \cdot |S|$ that satisfies $r$.*

*Proof.* It is easy to see that for any $u, v \in V$ the number of $uv$-edges in $J$ is at most $k$. Hence $J$ is a subgraph of $K_V$. As $\mathcal{C}$ is a partition of $S$, we have by Lemma 4 $c(J) = c(\bigcup_{C \in \mathcal{C}} J_C) \leq \sum_{C \in \mathcal{C}} c(J_C) \leq \sum_{C \in \mathcal{C}} \rho(k) \cdot |C| = \rho(k) \cdot |S|$. To prove that $J$ satisfies $r$, let $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$. For $1 \leq j \leq m$ let $G_j = G - (\bigcup_{i=1}^{j} C_i) + (\bigcup_{i=1}^{j} J_{C_i})$. Using Lemma 3, a simple induction shows that for all $1 \leq j \leq m$, $G_j$ satisfies $r$. In particular, this is so for $J = G_m$.

Now we prove Lemma 4. Let $C \in \mathcal{C}$. We start with the easier case $|\Gamma_G(C)| \leq k$. Then $J_C$ consists of no more than $\ell_{uv}$ edges for every $u, v \in \Gamma_G(C)$. Let $u, v \in \Gamma_G(C)$. Since there are $\ell_{uv}$ internally disjoint $uv$-paths in the subgraph of $G$ induced by $\{u, v\} \cup C$, there is one such path containing no more than $\lfloor |C|/\ell_{uv} \rfloor$ points in $C$. Therefore $d(u, v) \leq |C|/\ell_{uv} + 1$. Consequently, the total cost of $uv$-edges in $J_C$ is bounded by $\ell_{uv} \cdot (\frac{|C|}{\ell_{uv}} + 1) = |C| + \ell_{uv} \leq 2|C|$. Thus as $|\Gamma_G(C)| \leq k$ we have

$$c(J_C) \leq \binom{k}{2} \cdot 2|C| \leq \frac{1}{2}k(k-1) \cdot 2|C| \leq k^2 \cdot |C| \leq \rho(k) \cdot |C| \ .$$

This finishes the proof of Lemma 4 for the case $|\Gamma_G(C)| \leq k$.

We now turn to prove Lemma 4 for the case $|\Gamma_G(C)| \geq k+1$. We will use the following two easy statements (Lemma 5 is well known and therefore its proof is omitted):

**Lemma 5.** *Let $H$ be a $k$-connected graph on at least $k+1$ nodes. Then the graph obtained from $H$ by adding a new node and joining it to some $k$ nodes of $H$ is also $k$-connected.*

**Lemma 6.** *Let $U', U''$ be two subsets of the node set $V$ of a graph $H$ so that their union is $V$, and so that each of $U', U''$ has at least $k+1$ nodes and induces in $H$ a $k$-connected graph. If $H$ contains a matching $M$ between $U' - U''$ and $U'' - U'$ of size at least $k - |U' \cap U''|$ then $H$ is $k$-connected.*

*Proof.* By the theorem of Whitney cited earlier, a graph $H$ on at least $k+1$ nodes is $k$-connected if, and only if, $\kappa_H(u,v) \geq k$ for every $u,v \in V$ so that $uv \notin H$. Therefore, it is sufficient to prove that if $u,v \in V$ and $uv \notin H$ then $H-A$ contains a $uv$-path for any $A \subseteq V-\{u,v\}$ so that $|A| \leq k-1$. If $u,v \in U'$ or if $u,v \in U''$ then the statement is obvious, so assume that $u \in U'-U''$ and $v \in U''-U'$. If there is $w \in (U' \cap U'')-A$, then there is a $uw$-path and a $wv$-path in $H-A$. Therefore there is a $uv$-path in $H-A$. Otherwise, $U' \cap U'' \subseteq A$, and in particular $|U' \cap U''| \leq k-1$. Thus $H-A$ has an edge $e=w'w'' \in M$. We then obtain a $uv$-path in $H-A$ by the same argument as before.

**Lemma 7.** *Let $C_U = \{u \in C : |\Gamma_G(u) \cap V| \geq k+1\}$ and $C_W = C - C_U$. Let $U = \bigcup_{u \in C_U} \Gamma_G(u) \cap V$ and $W = \bigcup_{w \in C_W} \Gamma_G(w) \cap V$.*

(i) *If $U \neq \emptyset$ then $|U| \geq k+1$ and there is a set $E_U$ of edges of $K_V$ on $U$ such that the graph $(U, E_U)$ is $k$-connected and $c(E_U) \leq |C_U|\left(\Delta k^2 + 2k + 2\right) + 2k|C|$.*

(ii) *If $|W| \geq k+1$ then there is a set $E_W$ of edges of $K_V$ on $W$ such that the graph $(W, E_W)$ is $k$-connected and $c(E_W) \leq |C_W|(2k^2+2k) + |C|(3k^2+2k) - k$.*

*Proof.* Let $T$ be a spanning tree in the subgraph induced in $G$ by $C$. Order the nodes in $C_U$ and in $C_W$ in the order of some Eulerian tour of $T$, say $C_U = \{u_1, \ldots, u_p\}$ and $C_W = \{w_1, \ldots, w_q\}$. Let $U_i = \Gamma_G(u_i) \cap V$ and $W_i = \Gamma_G(w_i) \cap V$. Let $P_i$ be the part of the Eulerian Tour from $u_i$ to $u_{i+1}$, and let $Q_i$ be the part of the Eulerian Tour from $w_i$ to $w_{i+1}$, Clearly, $\sum_{i=1}^{p-1} |P_i| = 2|C| - 2 \leq 2|C|$ and $\sum_{i=1}^{q-1} |Q_i| = 2|C| - 2 \leq 2|C|$.

To prove part (i) of the lemma, assume $U$ is non-empty. By the definition of $C_U$, $|U| \geq k+1$. For every $1 \leq i \leq p$, $|U_i| \geq k+1$, and we will construct a $k$-connected graph on $U_i$ of cost $\leq \Delta k^2 + 2$. Then we will add a matching $M_i$ between $U_i - U_{i+1}$ and $U_{i+1} - U_i$ so that $|M_i| \geq k - |U_i \cap U_{i+1}|$ and $c(M_i) \leq k(|P_i|+2)$. The union of the constructed graphs will be a $k$-connected graph, by Lemma 6. The total cost of the matchings is $\leq 2k|C| + 2kp$. Consequently, we get a $k$-connected graph on $U$ of cost $\leq p(\Delta k^2+2) + 2k|C| + 2kp \leq |C_U|\left(\Delta k^2 + 2k + 2\right) + 2k|C|$ as required.

Fix $1 \leq i \leq t$. We now construct a $k$-connected graph on $U_i$. By the definition of $\Delta$, since $U_i \subseteq \Gamma_G(u_i)$, there is a dominating set $U_i^1$ of size at most $\Delta$ in $G[U_i]$. By the same arguments, if $U_i - U_i^1$ is non-empty, there is a dominating set $U_i^2$ of size at most $\Delta$ in $G[U_i - U_i^1]$. Repeating the process $k$ times, and accumulating the dominating sets, we obtain a set $U_i'$, so that $|U_i'| \leq \Delta k$ and for every $u \in U_i - U_i'$, $u$ has at least $k$ neighbors from $U_i'$ in $G[U_i]$. By a theorem of Harary (c.f. [12]), there is a $k$-connected graph on $U_i'$ containing $\lceil \Delta k^2/2 \rceil$ edges. Since $U_i \subseteq \Gamma_G(u_i)$, and by the triangle inequality, every such edge has cost $\leq 2$. We get a $k$-connected graph on $U_i'$ of cost $\leq 2\lceil \Delta k^2/2 \rceil \leq \Delta k^2 + 2$. Every node in $U_i - U_i'$ is connected to at least $k$ nodes in the constructed graph, therefore by Lemma 5, the constructed graph is a $k$-connected graph on $U_i$.

To prove part (ii) of the lemma, assume $|W| \geq k+1$. We construct a $k$-connected graph on $W$. The construction is as follows. Let $W_i' = W_i - \bigcup_{j=1}^{i-1} W_j$. Then the nonempty sets from $W_1', \ldots, W_q'$ partition $W$. Traversing the sequence

$W'_1, W'_2, \ldots, W'_q$ from left to right, we can partition it into blocks, each consisting of consecutive sets from the sequence, such that: the number of nodes in the union of the sets in each block is between $k+1$ and $2k$, except maybe that the last block has less than $k+1$ nodes. We will construct a clique on the nodes of each block. We then add a matching $M_t$ as in Lemma 6 between each block $t$ and block $t+1$, except that if the last block has less than $k+1$ nodes, then we connect each of its nodes to the preceding block as described in Lemma 5.

Consider the first block, say $W'_1, \ldots, W'_\ell$, and let $B_1$ be the union of the sets in this block. Note that $k+1 \leq |B_1| \leq 2k$. We bound the cost of a clique on $B_1$ as follows. In $G$, each $W'_i$ is connected by a star with center $w_i$, and $w_i$ is joined to $w_{i+1}$ by the path $Q_i$. An edge connecting a node in $W'_i$ to a node in $W'_{i+j}$ shortcuts at most one edge from the star of each of $W'_i, W'_{i+j}$, and each of the paths $Q_i, \ldots, Q_{i+j-1}$. Thus by the triangle inequality, each such edge adds at most $2 + |Q_i| + \cdots + |Q_{i+j-1}|$ to the cost. Clearly, over all edges, we shortcut every $Q_i$ at most $|B_1|^2/4$ times. In addition, every edge adds at most 2 to the cost, which sums to at most $2\binom{|B_1|}{2} \leq |B_1|^2$ for all edges. Denoting $L_1 = \sum_{i=1}^{\ell} |Q_i|$ and recalling that $|B_1| \leq 2k$ we obtain that the cost of a clique on $B_1$ is bounded by

$$\frac{|B_1|^2}{4} \cdot \sum_{i=1}^{\ell-1} |Q_i| + |B_1|^2 \leq k^2 L_1 + 2k|B_1| \ .$$

A similar argument applies on every block $t$. Since $\sum_t L_t = 2|C| - 2$ and $\sum_t |B_t| \leq |W| \leq k|C_W|$, the overall cost of the cliques on the blocks is bounded by

$$k^2 \sum_t L_t + 2k \sum_t |B_t| \leq 2k^2|C| - 2k^2 + 2k^2|C_W|.$$

For every $t$, we choose a set $B'_t$ of $k$ nodes from $B_t$ arbitrarily. Next we construct consecutive matchings between $B'_t$ and $B'_{t+1}$ for all $t$. For all $i$, $Q_i$ is shortened at most $k$ times. In addition, by previous arguments, each edge may shortcut at most one edge from the stars around some $w'_i$ and $w'_j$. Thus the cost of all matchings is bounded by

$$k \sum_{i=1}^{q} |Q_i| + 2k|C_W| \leq 2k|C| - 2k + 2k|C_W| \ .$$

Finally, if the last block has at most $k$ nodes, we connect every its node to $k$ nodes from the preceding block, thus constructing a $k$ connected graph on $W$ by Lemma 5. By the triangle inequality, for every $u, v \in \Gamma_G(C)$, $d(u,v) \leq |C| + 1$, thus every edge is of cost no greater than $|C| + 1$, and the added edges add a cost of at most

$$k^2(|C| + 1) \leq k^2|C| + k^2 \ .$$

The total cost of the edges added is bounded by

$$|C_W|(2k^2 + 2k) + |C|(3k^2 + 2k) - k^2 - 2k \leq |C_W|(2k^2 + 2k) + |C|(3k^2 + 2k) - k$$

This completes the proof of Lemma 7.

Now we finish the proof of Lemma 4. If $|W| \geq k+1$ and $U$ is non-empty, then there is a matching $E_{UW}$ of size $k$ between $U$ and $W$. By the triangle inequality $c(E_{UW}) \leq k(|C|+1)$. By Lemma 7 and Lemma 6, the edge set $J_C = E_U \cup E_W \cup E_{UW}$ forms a $k$-connected graph on $\Gamma_G(C)$, of cost at most (assuming $\Delta \geq 2$):

$$\begin{aligned} c(E_U) + c(E_W) + c(E_{UW}) \leq \\ &\leq |C_U|(\Delta k^2 + 2k + 2) + |C_W|(2k^2 + 2k) + |C|(3k^2 + 5k) \leq \\ &\leq (|C_U| + |C_W|)(\Delta k^2 + 2k + 2) + |C|(3k^2 + 5k) \leq \\ &\leq |C| \left((\Delta + 3)k^2 + 7k + 2\right) = \rho(k)|C| \ . \end{aligned}$$

If $|W| \leq k$, then $U$ is non-empty, since $U \cup W = \Gamma_G(C)$ and $|\Gamma_G(C)| \geq k+1$. Then in addition to $E_U$, we connect every node in $W$ to $k$ arbitrary nodes in $U$. This gives a $k$-connected graph on $\Gamma_G(C)$, by Lemma 5. By the triangle inequality, the cost of added edges is $\leq k^2(|C|+1) \leq 2k^2|C|$. Thus the total cost is $\leq \rho(k)|C|$.

This finishes the proof of Lemma 4, and thus also the proof of Lemma 2 is complete.

**A tight example:** The following example shows that our analysis is tight (up to constants). Given $k$ points in a ball of radius $1/2$ with uniform requirements as an instance for SN-MSP, an optimal solution size is 1 – add one Steiner Point in the ball. An optimal solution for the SNDP instance has cost $\binom{k}{2}$, as it is a union of two cliques on $V$: in one clique every edge $uv$ has cost $\lceil d(u,v) \rceil - 1 = 0$, while in the other every edge $uv$ has cost $\lceil d(u,v) \rceil = 1$.

## References


1. Agrawal, A., Klein, P.N., Ravi, R.: When trees collide: An approximation algorithm for the generalized Steiner problem on networks. SIAM J. Computing 24(3), 440–456 (1995)
2. Althaus, E., Calinescu, G., Mandoiu, I., Prasad, K., Tchervenski, N., Zelikovsky, A.: Power efficient range assignment for symmetric connectivity in static ad hoc wireless networks. Wireless Networks 12(3), 287–299 (2006)
3. Bredin, J., Demaine, E., Hajiaghayi, M., Rus, D.: Deploying sensor networks with guaranteed capacity and fault tolerance. In: MobiHoc, pp. 309–319 (2005)
4. Cheriyan, J., Vempala, S., Vetta, A.: An approximation algorithm for the minimum-cost $k$-vertex connected subgraph. SIAM J. Comput. 32(4), 1050–1055 (2003)
5. Chuzhoy, J., Khanna, S.: Algorithms for single-source vertex-connectivity. In: FOCS, pp. 105–114 (2008)
6. Chuzhoy, J., Khanna, S.: An $O(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. In: FOCS, pp. 437–441 (2009)
7. Du, D.-Z., Wang, L., Xu, B.: The euclidean bottleneck steiner tree and steiner tree with minimum number of steiner points. In: Wang, J. (ed.) COCOON 2001. LNCS, vol. 2108, pp. 509–518. Springer, Heidelberg (2001)

8. Estrin, D., Girod, L., Pottie, G., Srivastava, M.: Instrumenting the world with wireless sensor networks. In: International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 2033–2036 (2001)
9. Fackharoenphol, J., Laekhanukit, B.: An $O(\log^2 k)$-approximation algorithm for the $k$-vertex connected subgraph problem. In: STOC, pp. 153–158 (2008)
10. Frank, A.: Connectivity and network flows. In: Graham, R.L., Grötschel, M., Lovász, L. (eds.) Handbook of Combinatorics, ch. 2, pp. 111–177. Elsevier Science, Amsterdam (1995)
11. Frank, A., Tardos, É.: An application of submodular flows. Linear Algebra and its Applications 114/115, 329–348 (1989)
12. Harary, F.: The maximum connectivity of a graph. Natl. Acad. Sci, 1142–1146 (1962)
13. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21(1), 39–60 (2001)
14. Kabatjansky, G.A., Levenstein, V.: Bounds for packing of the sphere and in space. Prob. Information Trans. 14, 1–17 (1978)
15. Klein, C., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted steiner trees. Journal of Algorithms 19(1), 104–115 (1995)
16. Kortsarz, G., Mirrokni, V.S., Nutov, Z., Tsanko, E.: Approximating minimum-power degree and connectivity problems. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 423–435. Springer, Heidelberg (2008)
17. Kortsarz, G., Nutov, Z.: Approximating $k$-node connected subgraphs via critical graphs. SIAM J. Comput. 35(1), 247–257 (2005)
18. Nutov, Z.: Approximating minimum cost connectivity problems via uncrossable bi-families and spider-cover decompositions (2010) (manuscript); Preliminary version in FOCS 2009, pp. 417–426
19. Nutov, Z.: Approximating Steiner networks with node weights. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 411–422. Springer, Heidelberg (2008)
20. Nutov, Z.: An almost $O(\log k)$-approximation for $k$-connected subgraphs. In: SODA, pp. 912–921 (2009)
21. Nutov, Z.: Approximating minimum-power $k$-connectivity. Ad Hoc & Sensor Wireless Networks 9(1-2), 129–137 (2010)
22. Nutov, Z., Yaroshevitch, A.: Wireless network design via 3-decompositions. Inf. Process. Lett. 109(19), 1136–1140 (2009)
23. Robins, G., Salowe, J.S.: Low-degree minimum spanning trees. Discrete Comput. Geom. 14, 151–165 (1999)

# A 3/2-Approximation Algorithm for Rate-Monotonic Multiprocessor Scheduling of Implicit-Deadline Tasks

Andreas Karrenbauer[1] and Thomas Rothvoß[2]

[1] Zukunftskolleg, University of Konstanz, Germany
andreas.karrenbauer@uni-konstanz.de
[2] Institute of Mathematics, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
thomas.rothvoss@epfl.ch

**Abstract.** We present a new approximation algorithm for rate-monotonic multiprocessor scheduling of periodic tasks with implicit deadlines. We prove that for an arbitrary parameter $k \in \mathbb{N}$ it yields solutions with at most $(\frac{3}{2} + \frac{1}{k})OPT + 9k$ many processors, thus it gives an asymptotic $3/2$-approximation algorithm. This improves over the previously best known ratio of $7/4$. Our algorithm can be implemented to run in time $O(n^2)$, where $n$ is the number of tasks. It is based on custom-tailored weights for the tasks such that a greedy maximal matching and subsequent partitioning by a first-fit strategy yields the result.

## 1 Introduction

In this paper, we consider the *synchronous rate-monotonic real-time scheduling problem with implicit deadlines*. That is, we are given a set of $n$ tasks $V := \{\tau_1, \ldots, \tau_n\}$ attributed with *execution times* $c(\tau_i)$ and *periods* $p(\tau_i)$. Each task releases a job at time 0 and subsequently at each integer multiple of its period (hence *synchronous*). Furthermore, each job of a task has to be finished before the next job of the same task is released. In other words the relative deadlines of jobs are *implicitly* given by the periods. We allow *preemption*, but we require *fixed priorities* to reduce the overhead during runtime. That is, the current job is preempted, if a new job with a higher priority is released. In this context, Liu and Layland [1] have shown that if there are feasible fixed priorities then *rate-monotonic* priorities, which are higher for smaller periods, also define a feasible schedule. See the book of Buttazzo [2] for a comparison of fixed-priority versus dynamic-priority scheduling policies.

Since multi-core and multi-processor environments become more and more popular, we consider the problem of assigning the tasks to a minimum number of processors such that there is a feasible rate-monotonic schedule for each processor. Formally

> Given tasks $V = \{\tau_1, \ldots, \tau_n\}$, running times $c : V \to \mathbb{Q}_+$, and periods $p : V \to \mathbb{Q}_+$, where each task $\tau$ generates a job of length $c(\tau) \leq p(\tau)$ and relative deadline $p(\tau)$ at $z \cdot p(\tau)$, for all $z \in \mathbb{Z}_{\geq 0}$, find the minimum $\ell$ such that there is a partition of $V = P_1 \dot\cup \ldots \dot\cup P_\ell$ subject to RM-schedulability of $P_j$ for each $j$.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 166–177, 2011.

Here we *forbid migration*, i.e. jobs of the same task must always be processed on the same machine. This scheduling problem has received considerable attention in the real-time and embedded-systems community. This popularity is due to the fact that more and more safety-critical control applications are carried out by microprocessors and in particular by multiprocessor environments. Such scheduling problems are today a major algorithmic challenge in the automotive and aviation industry.

The idea for our algorithm is as follows: Suppose all tasks had utilization larger than $\frac{1}{3}$. Then at most 2 tasks can be assigned to each processor. Define an undirected graph $G = (V, E)$ with the tasks being the nodes and an edge $\{\tau_1, \tau_2\}$ for each pair such that $\tau_1$ and $\tau_2$ can be RM-scheduled on a single processor. Then the size of a maximum matching plus the number of nodes not covered by that matching gives $OPT$. We incorporate the existence of small tasks by only including an edge $\{\tau_1, \tau_2\} \in E$ if $w(\tau_1)+w(\tau_2)$ exceeds a certain threshold. Here $w(\tau)$ is a proper weight function which is monotonically increasing with the *utilization* $u(\tau) = \frac{c(\tau)}{p(\tau)}$.

### 1.1 Related Work

The famous *Bin Packing* problem is an important special case of our scheduling problem. The objective of Bin Packing is to find a partition of a set of items of different sizes, say $u_i \in (0, 1]$ for $i = 1, \ldots, n$, into a minimum number of bins such that the total size of each bin does not exceed 1. The similarity to our scheduling problem becomes apparent by introducing the notion of the utilization of a task, i.e. $u(\tau) = c(\tau)/p(\tau)$. If all periods are the same, e.g. the common denominator of rational item sizes, then the priorities for the rate-monotonic scheduling problem become irrelevant and a set of tasks is feasible for one processor, if and only if their total utilization does not exceed 1.

Successful heuristics for Bin Packing are *First Fit*, *Next Fit* and *Best Fit*. In all variants the items are assigned in a consecutive manner to a bin, which has enough space (or a new one is opened). For First Fit the current item is assigned to the bin with the smallest index, in Best Fit it is assigned to the bin, whose item sum is maximal. For Next Fit an active bin is maintained. If the current item does not fit into it, a new bin is opened, now being the active one; old bins are never considered again. In *First Fit Decreasing* the items are first sorted by decreasing sizes and then distributed via First Fit. In the worst-case Next Fit produces a 2-approximation, while First Fit needs $\lceil \frac{17}{10} OPT_{\text{BinPacking}} \rceil + 1$ many bins [3]. Asymptotically, Best and First Fit Decreasing have an approximation ratio of $11/9$ [4]. Furthermore, there is an asymptotic PTAS [5] and even an asymptotic FPTAS exists [6]. More on Bin Packing can be found in the excellent survey of Coffman et al. [7].

The *utilization* of a task set $V'$ is defined as $u(V') = \sum_{\tau \in V'} c(\tau)/p(\tau)$. If $V'$ is feasible (i.e. RM-schedulable on a single machine), then the utilization $u(V')$ is at most 1. However, $V'$ can be infeasible, even if $u(V') < 1$. Liu and Layland [1] have shown that $V'$ is feasible, if $u(V')$ is bounded by $n'(2^{1/n'} - 1)$, where $n' = |V'|$. This bound tends to $\ln(2)$ and the condition is not necessary for feasibility, as the example with equal periods shows. Stronger, but still not necessary conditions for feasibility are given in [8,9,10].

The *response time* of a job is the difference of release time and completion time. The response time of a task is defined as the maximal response time of any of its jobs. In our synchronous setting, this value is attained for the first job (which is released at time 0), see [1].

If $p(\tau_1) \leq \ldots \leq p(\tau_n)$ then the response time for $\tau_i$ in a rate-monotonic, uniprocessor schedule is given by the smallest value $r(\tau_i) \geq 0$ with

$$r(\tau_i) = c(\tau_i) + \sum_{j<i} \left\lceil \frac{r(\tau_i)}{p(\tau_j)} \right\rceil c(\tau_j).$$

Of course $\tau_1, \ldots, \tau_n$ are feasible if and only if $r(\tau_i) \leq p(\tau_i)$ for $i = 1, \ldots, n$. But it was proved in [11] that such response times cannot even be approximated in polynomial time within a constant factor, unless $\mathbf{NP} = \mathbf{P}$. Nevertheless in practice response times can be efficiently computed using a fix-point iteration approach [12]. Furthermore Baruah and Fisher [13] showed that there is an FPTAS for computing the minimum processor speed, which is needed to make a task system RM-schedulable. However, the complexity status of verifying, whether the RM-schedule of a set of implicit deadline tasks on a single machine is feasible, remains an open problem [14]. Fortunately for $n = 2$ there is an simple exact criterion (cf. [15], chapter 32): The task set $\{\tau_1, \tau_2\}$ with $p(\tau_1) \leq p(\tau_2)$ is RM-schedulable if and only if

$$c(\tau_2) \leq \left\lfloor \frac{p(\tau_2)}{p(\tau_1)} \right\rfloor (p(\tau_1) - c(\tau_1)) + \max\left\{0, p(\tau_2) - \left\lfloor \frac{p(\tau_2)}{p(\tau_1)} \right\rfloor p(\tau_1) - c(\tau_1)\right\}. \quad (1)$$

This constant time test will be used in our algorithm.

Most popular algorithms for rate-monotonic multiprocessor scheduling first sort the tasks in a suitable way and then distribute them in a First Fit or Next Fit manner using a sufficient feasibility criterion. See the following table for an overview (with our algorithm in the last row, for the sake of comparability).

| **algorithm** | **references** | **sorting** | **distribution** | **ratio** | **time** |
|---|---|---|---|---|---|
| RMNF | [16,17] | inc. $p(\tau)$ | Next Fit | 2.67 | $O(n \log n)$ |
| RMFF | [16,17] | inc. $p(\tau)$ | First Fit | 2.00 | $O(n \log n)$ |
| RRM-FF | [18] | - | First Fit | 2.00 | $O(n \log n)$ |
| RRM-BF | [18] | inc. $p(\tau)$ | Best Fit | 2.00 | $O(n \log n)$ |
| FFDU | [17] | dec. $u(\tau)$ | First Fit | 2.00 | $O(n \log n)$ |
| RMST | [8] | inc. $S(\tau)$ | Next Fit | $\frac{1}{1-\alpha}$ | $O(n \log n)$ |
| RMGT | [8] | - | First Fit + RMST | 1.75 | $O(n^2)$ |
| FFMP | [19] | inc. $S(\tau)$ | First Fit | 2.00 | $O(n \log n)$ |
| $k$-RMM | - | - | Matching + FFMP | 1.50 | $O(n^2)$ |

Here $S(\tau) = \log_2 p(\tau) - \lfloor \log_2 p(\tau) \rfloor$ and $\alpha = \max_{\tau \in V} u(\tau)$. In the table, column "ratio" denotes the best known upper bounds on the asymptotic approximation ratio. The *Rate-monotonic general task* algorithm [8] distributes tasks with utilization at most $1/3$ using RMST and the rest separately with First Fit. Also the algorithms RRM-FF and RRM-BF apply the same grouping strategy. A more detailed description can be found in [17].

Furthermore there is an asymptotic PTAS under resource augmentation, computing for any fixed $\varepsilon > 0$ a solution with $(1+\varepsilon)OPT + O(1)$ processors, where the tasks on each processor can be feasibly scheduled after increasing the processor speed by a factor of $1+\varepsilon$ [20]. In the same paper it was proved that unless $\mathbf{P} \neq \mathbf{NP}$, no asymptotic FPTAS can exist for this multiprocessor scheduling problem. But it is still an open question whether an asymptotic PTAS is possible. We refer to the article [21] for an overview on complexity issues of real-time scheduling.

### 1.2 Our Contribution

We present a new polynomial time algorithm for rate-monotonic real-time scheduling, which is based on matching techniques and yields solutions of at most $(\frac{3}{2}+\frac{1}{k})OPT+9k$ many processors. The asymptotic approximation ratio tends to $3/2$ (for growing $k$), improving over the previously best known value of $7/4$. Moreover, we provide experimental evidence that our new algorithm outperforms all other existing algorithms.

## 2 Preliminaries

During our algorithm it will happen, that we discard a set of (in general small) tasks $V' \subseteq V$ and schedule them using a simple heuristic termed *First Fit Matching Periods* (FFMP), which was introduced in [19]. For a task $\tau$ define

$$S(\tau) := \log_2 p(\tau) - \lfloor \log_2 p(\tau) \rfloor \qquad \text{and} \qquad \beta(V) := \max_{\tau \in V} S(\tau) - \min_{\tau \in V} S(\tau)$$

then the FFMP heuristic can be stated as follows

**Algorithm 1.** FFMP

(1) Sort tasks such that $0 \leq S(\tau_i) \leq \ldots \leq S(\tau_n) < 1$
(2) FOR $i = 1, \ldots, n$ DO
  (3) Assign $\tau_i$ to the processor $P_j$ with the least index $j$ such that

$$u(P_j \cup \{\tau_i\}) \leq 1 - \beta(P_j \cup \{\tau_i\}) \cdot \ln(2)$$

The idea for this ordering of the tasks is that consecutive tasks will have periods that are nearly multiples of each other and hence the bin packing aspect of the problem becomes dominant. Let FFMP$(V)$ denote the value of the solution, which FFMP produces, if applied to $V$. One can prove the following lemma using well known techniques from [8] (see also [15]).

**Lemma 1.** *Given periodic tasks $V = \{\tau_1, \ldots, \tau_n\}$ and $k \in \mathbb{N}$. FFMP always produces feasible solutions such that*

- *If $u(\tau_i) \leq \alpha \leq \frac{1}{2}$ for all $i = 1, \ldots, n$, then FFMP$(V) \leq \frac{1}{1-\alpha} u(V) + 3$.*
- *If $u(\tau_i) \leq \frac{1}{2} - \frac{1}{k}$ for all $i = 1, \ldots, n$, then FFMP$(V) \leq \frac{n}{2} + \frac{k}{2}$.*

The RMST algorithm of Liebeherr et al. [8] also fulfills the same properties. But on average the First Fit distribution for FFMP behaves much better than the Next Fit distribution of RMST. However just for a worst-case analysis one could replace FFMP by RMST.

## 3 Matchings and Schedules

As a powerful tool, we will use matchings in our algorithm. To this end, we define an undirected graph $G = (V, E)$ such that the nodes correspond to the tasks. If there is an edge between the nodes $\tau_1$ and $\tau_2$, then the corresponding tasks can be scheduled on one processor. Suppose for the time being that all tasks have a utilization of more than $\frac{1}{3}$ and thus at most two tasks fit on one processor. Then the maximum cardinality matching in $G$ determines a schedule with a minimum number of processors by reserving one processor for each edge in the matching and one processor for each unmatched node.

For the general setting of tasks with arbitrary utilization, this basic idea for our algorithm persists: Compute a matching in $G$, schedule each pair of matched tasks together on one processor, and distribute the remaining tasks by FFMP. Of course, the matching should be in such a way that we use the processors efficiently. To this end, we will assign weights to the nodes depending on the utilization of the corresponding tasks. We will later define the weights exactly. For now, let the weights be a function $w : V \to [0, 1]$ and let the *price* of a matching $M \subseteq E$ be

$$price(M) := |M| + w(\overline{M}),$$

where $\overline{M} := \{v \in V \mid \forall e \in M : v \notin e\} \subseteq V$ is the set of unmatched nodes and $w(\overline{M}) := \sum_{v \in \overline{M}} w(v)$. That is, we have to allocate 1 processor for each matched pair of tasks and also some more processors for distributing the remaining unmatched tasks. Note that finding the matching with minimum price is equivalent to computing the maximum weight matching with edge weights $w(e) := w(u) + w(v) - 1$ for each edge $e = \{u, v\}$, since

$$w(M) := \sum_{e \in M} w(e) = \sum_{v \in V} w(v) - \sum_{v \in \overline{M}} w(v) - |M| = w(V) - price(M).$$

While a maximum weight matching in a graph with $n$ nodes and $m$ edges can be found in $O(n(m + n \log n))$ [22], we will see that it is sufficient for our purpose to compute an inclusion-wise maximal matching greedily. That is, we maintain the property, that for all $e \in E \setminus M$ we have $w(e') \geq w(e)$ for all $e' \in M$ or there is an edge $e' \in M$ with $e \cap e' \neq \emptyset$ and $w(e') \geq w(e)$, throughout the algorithm. Furthermore, the algorithm iterates until $\overline{M}$ does not contain an edge, i.e. $|e \cap \overline{M}| < 1$ for all $e \in E$. Note that such a *greedy maximal matching* can be computed in $O(n^2)$ by sorting the tasks by decreasing weight and searching for each task $\tau_i$ the first unmatched $\tau_j$ with $\{\tau_i, \tau_j\} \in E$. Although we do not have an explicit representation of the edges, the check whether a pair of nodes forms an edge takes only constant time. The interested reader is pointed to [23] or [24] for an extensive account on matchings.

## 4 The Algorithm

As indicated in the previous section, we compute a weighted matching to find a good schedule. It remains to define the weights properly. Note that each edge yields a processor in the partition. Hence, we do not want to match two nodes which do not use the

processor to some extent. Moreover, each unmatched node is first discarded and later scheduled via FFMP. We are now going to define node weights $w$ in such a way, that a matching with costs $\gamma$ can be turned into a feasible schedule of roughly $\gamma$ many processors. Intuitively, the weight $w(\tau) \in [0,1]$ will denote the average number of processors per task, which the FFMP algorithm needs to schedule a large number of tasks, if all tasks have the same utilization as $\tau$. Here we distinguish 3 categories of tasks:

- *Small tasks* ($0 \leq u(\tau) \leq \frac{1}{3}$): Consider tasks $\tau_1, \ldots, \tau_m$ with a small utilization, i.e. $u(\tau_i) \leq \alpha$ for all $i = 1, \ldots, m$ and $\alpha \leq 1/3$. Then we may schedule such tasks with FFMP using $u(\{\tau_1, \ldots, \tau_m\})\frac{1}{1-\alpha} + 3 \leq m \cdot \alpha \frac{1}{1-\alpha} + 3$ many processors (see Lemma 1), thus we choose $w(\tau) := \frac{u(\tau)}{1-u(\tau)}$ for a small task $\tau$.
- *Medium tasks* ($\frac{1}{3} < u(\tau) \leq \frac{1}{2} - \frac{1}{12k}$): Suppose we have tasks $\tau_1, \ldots, \tau_m$ whose utilization is at least $1/3$, but bounded away from $1/2$, say $u(\tau_i) \leq \frac{1}{2} - \frac{1}{12k}$, where $k$ is an integer parameter that we determine later. Then FFMP$(\{\tau_1, \ldots, \tau_m\}) \leq m/2 + O(k)$ (see again Lemma 1), thus we choose $w(\tau) := 1/2$ for medium tasks.
- *Large tasks* ($u(\tau) > \frac{1}{2} - \frac{1}{12k}$): For a large task one processor is sufficient and possibly needed, thus $w(\tau) := 1$ in this case.

**Algorithm 2.** $k$-Rate-Monotonic-Matching algorithm ($k$-RMM)

(1) Construct $G = (V, E)$ with edges $e = \{\tau_1, \tau_2\} \in E \Leftrightarrow \{\tau_1, \tau_2\}$ RM-schedulable (according to condition (1)) and $w(e) > 0$.
(2) Sort the edges by decreasing weight (ties are broken arbitrarily) and compute the greedy maximal matching $M$ w.r.t. this order.
(3) For all $\{\tau_1, \tau_2\} \in M$ create a processor with $\{\tau_1, \tau_2\}$
(4) Define
  - $V_i = \{\tau \in \overline{M} : \frac{1}{3} \cdot \frac{i-1}{k} \leq u(\tau) < \frac{1}{3} \cdot \frac{i}{k}\}\ \forall i = 1, \ldots, k$
  - $V_{k+1} = \{\tau \in \overline{M} : \frac{1}{3} \leq u(\tau) \leq \frac{1}{2} - \frac{1}{12k}\}$
  - $V_{k+2} = \{\tau \in \overline{M} : u(\tau) > \frac{1}{2} - \frac{1}{12k}\}$
(5) Distribute $V_{k+2}, V_{k+1}, \ldots, V_1$ via FFMP.

The reason to define the weights in this way becomes clear with the proof of the following Theorem, saying that the number of used machines is essentially determined by the price of the matching.

**Theorem 1.** *Let $M$ be an arbitrary matching in $G$. The schedule created from $M$ as described in Algorithm 2, uses at most*

$$\left(1 + \frac{1}{2k}\right) \cdot price(M) + 9k$$

*many processors.*

*Proof.* We create $|M|$ processors, covering pairs of tasks $\{\tau_1, \tau_2\} \in M$. For scheduling the tasks in $V_{k+1}$ we know that according to Lemma 1

$$\text{FFMP}(V_{k+1}) \leq \frac{|V_{k+1}|}{2} + \frac{12k}{2} = \sum_{\tau \in V_{k+1}} w(\tau) + 6k$$

using that the utilization of all tasks in $V_{k+1}$ lies between $\frac{1}{3}$ and $\frac{1}{2} - \frac{1}{12k}$. Of course $\text{FFMP}(V_{k+2}) \leq |V_{k+2}| = \sum_{\tau \in V_{k+2}} w(\tau)$. For each $V_i$ ($i = 1, \ldots, k$) we know that the utilization of each task is sandwiched by $\frac{1}{3} \cdot \frac{i-1}{k}$ and $\frac{1}{3} \cdot \frac{i}{k}$. Consequently

$$\text{FFMP}(V_i) \leq \frac{1}{1 - \frac{i}{3k}} \cdot u(V_i) + 3 \leq \left(1 + \frac{1}{2k}\right) \frac{1}{1 - \frac{i-1}{3k}} \cdot u(V_i) + 3 \leq \left(1 + \frac{1}{2k}\right) w(V_i) + 3$$

by applying again Lemma 1 together with the fact that $w(\tau) \geq u(\tau) \cdot \frac{1}{1-(i-1)/(3k)}$ for all $\tau \in V_i$. We conclude that the total number of processors in the produced solution is

$$\begin{aligned}
|M| + \sum_{i=1}^{k+2} \text{FFMP}(V_i) &\leq |M| + \sum_{\tau \in V_{k+1} \cup V_{k+2}} w(\tau) + 6k \\
&\quad + \left(1 + \frac{1}{2k}\right) \sum_{\tau \in V_1 \cup \ldots \cup V_k} w(\tau) + 3k \\
&\leq |M| + \left(1 + \frac{1}{2k}\right) \sum_{\tau \in V_1 \cup \ldots \cup V_{k+2}} w(\tau) + 9k \\
&\leq \left(1 + \frac{1}{2k}\right) \cdot price(M) + 9k.
\end{aligned}$$

□

It remains to show that the price of the matching computed by Algorithm 2 is at most roughly $\frac{3}{2}$ times the number of necessary processors. To this end, we first show that for any partition, there is a matching with the appropriate price.

**Theorem 2.** *For any feasible partition $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ of the tasks, there is a matching $M_\mathcal{P}$ with*

$$price(M_\mathcal{P}) \leq \left(\frac{3}{2} + \frac{1}{12k}\right) \cdot |\mathcal{P}|$$

*such that no $e \in M_\mathcal{P}$ crosses a $P_i \in \mathcal{P}$, i.e. either $e \subseteq P_i$ or $e \cap P_i = \emptyset$.*

*Proof.* Consider a processor $P_i$. After reordering let $\tau_1, \ldots, \tau_q$ be the tasks on $P_i$, sorted such that $u(\tau_1) \geq \ldots \geq u(\tau_q)$. First suppose that $q \geq 2$. We will either cover two tasks in $P_i$ by a matching edge or leave all tasks uncovered. But in any case we guarantee, that the tasks in $P_i$ contribute at most $(\frac{3}{2} + \frac{1}{12k})$ to $price(M_\mathcal{P})$. We distinguish two cases, depending on whether $P_i$ contains a large task or not.

**Case $\tau_1$ not large:** We leave all tasks in $P_i$ uncovered. Note that all tasks in $P_i$ are either of small or medium size, hence $w(\tau_j) \leq \frac{3}{2} u(\tau_j)$ for $j = 1, \ldots, q$. The contribution of $P_i$ is $\sum_{j=1}^{q} w(\tau_j) \leq \frac{3}{2} \sum_{j=1}^{q} u(\tau_j) \leq \frac{3}{2}$.

**Case $\tau_1$ large:** We add $\{\tau_1, \tau_2\}$ to the matching. We may do so since both tasks are RM-schedulable, the weight of the edge is positive because $\tau_1$ is large, and hence $\{\tau_1, \tau_2\} \in E$. The contribution is

$$1 + \sum_{j=3}^{q} w(\tau_j) \leq 1 + \underbrace{\sum_{j=3}^{q} u(\tau_j)}_{\leq 1-u(\{\tau_1,\tau_2\})} \cdot \frac{1}{1 - \underbrace{u(\tau_j)}_{\leq u(\tau_2)}} \leq 1 + \frac{1 - u(\tau_1) - u(\tau_2)}{1 - u(\tau_2)} \quad (2)$$

$$\leq 1 + \frac{\frac{1}{2} + \frac{1}{12k} - u(\tau_2)}{1 - u(\tau_2)} \leq \frac{3}{2} + \frac{1}{12k}$$

using that $\tau_3, \ldots, \tau_q$ are small and $\frac{a-x}{1-x}$ is monotone decreasing if $a < 1$.

If $q = 1$, then we do not cover $\tau_1$. The contribution is at most 1. Moreover, the above construction guarantees that no edge in $M_{\mathcal{P}}$ crosses a processor $P_i$. □

If we compute a maximum weight matching in our algorithm (say in running time in $O(n^3)$), by simply combining Theorems 1 and 2, we can already obtain a bound of

$$\Big(\frac{3}{2} + \frac{1}{12k}\Big) \cdot \Big(1 + \frac{1}{2k}\Big) \cdot OPT + 9k \leq \Big(\frac{3}{2} + \frac{1}{k}\Big) OPT + 9k$$

on the number of used processor. However, we do not want to fall short of the running time of $O(n^2)$ of the $7/4$-approximation algorithm of Liebeherr et al. [8]. Hence, we use a greedy matching instead, which can be computed in $O(n^2)$. Observe that in the previous proof, in particular for the second case, we left some slack to the approximation ratio. This will become useful in the proof of the next theorem, saying that for any feasible partition it is sufficient to consider a greedy maximal matching.

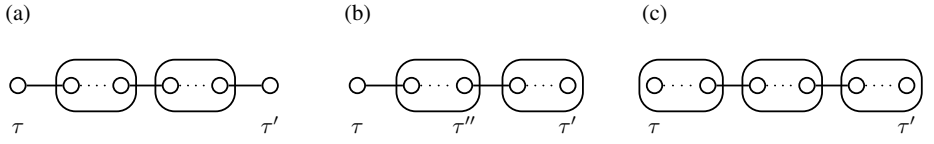
**Theorem 3.** *If $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ be a feasible partition, then we have for a greedy matching $M$ that*

$$price(M) \leq \Big(\frac{3}{2} + \frac{1}{6k}\Big)|\mathcal{P}|.$$

*Proof.* This proof is based on a comparison of $M$ with the matching $M_{\mathcal{P}}$, constructed in Theorem 2. To this end, we consider the symmetric difference of the two matchings, i.e. let $E' := M \Delta M_{\mathcal{P}}$. Note that $E'$ is a collection of disjoint paths and cycles, i.e. for all $v \in V$, we have $|\{e \in E' : v \in e\}| \leq 2$. First, we consider a cycle $C \subseteq E'$. Observe that $|C \cap M| = |C \cap M_{\mathcal{P}}|$ by the fundamentals of matching theory. Let $q := |C \cap M|$ and let $P_1, \ldots, P_q$ be the processors that contain edges from $C \cap M_{\mathcal{P}}$. Note that each edge in $M_{\mathcal{P}}$ is contained in exactly one processor and moreover that $M$ matches all nodes in $P_1 \cup \cdots \cup P_q$ that $M_{\mathcal{P}}$ does. Hence, we have

$$|C \cap M| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) = |C \cap M_{\mathcal{P}}| + \sum_{i=1}^{q} w(P_i \cap \overline{M}_{\mathcal{P}}) \leq \Big(\frac{1}{2} + \frac{1}{12k}\Big) q.$$

Next, we consider a path $Q \subseteq E'$. Again let $P_1, \ldots, P_q$ be the processors containing edges from $M_{\mathcal{P}} \cap Q$. We distinguish the three cases, when both, one, or none of the end-nodes of the path $Q$ are matched in $M$ as illustrated below. The solid edges belong to $M$ and the dashed ones belong to $M_{\mathcal{P}}$. The boxes represent the processors of




**Case (a).** If both ends of $Q$ are matched in $M$, then $|M \cap Q| - 1 = |M_{\mathcal{P}} \cap Q| = q$. Hence,

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq |M_{\mathcal{P}} \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}_{\mathcal{P}}) + 1 - w(\tau) - w(\tau')$$

where $\tau, \tau'$ are the both ends of $Q$. If one of $\tau, \tau'$ is large, then there is nothing to show. Suppose that none of them is large. Then there is at least one processor that contains two large tasks, since $Q$ has an odd number of edges and since by definition each edge contains at least one large task. Furthermore by the greedy selection, there is at least one large neighboring task in this path, and by the same parity argument, there is a further processor with two large tasks. Note that $q \geq 2$ if neither $\tau$ nor $\tau'$ is large. If $q = 2$ like in the above example, then all unmatched tasks on the two processors have a smaller weight than $\tau$ or $\tau'$, respectively. Since this yields the claim, we suppose that $q > 2$ in the following.

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq q + 1 + \Big(\frac{1}{2} + \frac{1}{12k}\Big)(q-2) + \frac{2}{6k-1} \leq \Big(\frac{3}{2} + \frac{1}{6k}\Big)q$$

**Case (b).** If exactly one of the endpoints of $Q$ is matched in $M$, say $\tau$, and the other endpoint, say $\tau'$ is matched on processor $P_q$, then

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq |M_{\mathcal{P}} \cap Q| - w(\tau) + w(\tau') + \sum_{i=1}^{q} w(P_i \cap \overline{M}).$$

If $q = 1$, then the greedy selection implies that $w(\tau) \geq w(\tau')$. Hence, we assume that $q \geq 2$. Let $\tau''$ be as in the illustration. By the greedy selection, we have $u(\tau'') \geq u(\tau')$. If $\tau''$ is small, then

$$w(P_{q-1} \cap \overline{M}) + w(P_q \cap \overline{M}) \leq \frac{\frac{1}{2} + \frac{1}{12k} - u(\tau'')}{1 - u(\tau'')} + \frac{\frac{1}{2} + \frac{1}{12k}}{1 - u(\tau')} \leq 1 + \frac{1}{4k}$$

as in Ineq. (2) in the proof of Theorem 2. By a similar argument, the same bound holds if $\tau''$ is medium. If $\tau''$ is large, then either $\tau$ is large itself or there is a processor $P_j$ with $j \in \{1, \ldots, q-1\}$ with two large tasks, since each edge contains at least one large task. In the former case, there is nothing to show, whereas in the latter case, we may assume w.l.o.g. that $j = q - 1$ and hence the bound of $1 + \frac{1}{4k}$ also holds. Note that if $w(\tau') = 1$, then no further unmatched task can be on $P_q$, and hence $w(P_q \cap \overline{M}) = 1$, because they would have been matched by the algorithm. Altogether, this yields

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq q + \Big(\frac{1}{2} + \frac{1}{12k}\Big)(q-2) + 1 + \frac{1}{4k} \leq \Big(\frac{3}{2} + \frac{1}{6k}\Big)q.$$

**Case (c).** If none of the endpoints of $Q$ are matched in $M$, then

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq |M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}_{\mathcal{P}}) - 1 + w(\tau) + w(\tau')$$

where $\tau, \tau'$ are the both ends of $Q$. If neither $\tau$ nor $\tau'$ is large, then there is nothing to show. Hence, we assume w.l.o.g. that $w(\tau) = 1$. Since $\tau$ is not matched in $M$, there is no further task on the same processor that is also unmatched in $M$. Hence,

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq \Big(\frac{3}{2} + \frac{1}{12k}\Big)(q-1) + w(\tau') \leq \Big(\frac{3}{2} + \frac{1}{12k}\Big)q. \qquad \square$$

**Corollary 1.** *Algorithm 2 produces a solution of cost* $(\frac{3}{2} + \frac{1}{k})OPT + 9k$ *in time* $O(n^2)$.

*Proof.* Note that for each set $\{\tau_1, \tau_2\}$ RM-schedulability can be tested in constant time using condition (1). Sorting the tasks by decreasing utilization takes $O(n \log n)$ time and is subsumed by the time necessary to create $G$, which is $O(n^2)$. In fact, it is only necessary to scan each large task and check with every other task with smaller utilization whether they can be scheduled together. If so both tasks are marked as matched provided that none of them has been matched before. However, this procedure still requires quadratic running time since all tasks might be large in the worst case. The running time of FFMP is $O(n' \log n')$ for scheduling $n'$ tasks, thus the total running time is $O(n^2)$.

The approximation guarantee follows from Theorem 3 and Theorem 1, since we may combine them to show that the number of processors produced does not exceed

$$\Big(1 + \frac{1}{2k}\Big) \cdot \Big(\frac{3}{2} + \frac{1}{6k}\Big)OPT + 9k \leq \Big(\frac{3}{2} + \frac{1}{k}\Big) \cdot OPT + 9k. \qquad \square$$

## 5 Experimental Results

We have implemented and compared our $k$-RMM algorithm experimentally with the ones, which are known from the literature and have already been mentioned in Sect. 1.1. To this end, we have randomly generated instances with the number of tasks $n$ ranging from 10 to $10^5$. That is, for each given $n$, we have generated 100 samples, where integer periods have been chosen out of $(0, 500)$ uniformly at random and independently utilizations from $(0, 1)$ u.a.r. All algorithms have been tested on the same instances to allow also a direct comparison. With a choice of $k = \lfloor \sqrt{n} \rfloor$, our new algorithm has outperformed the others on almost all instances (in fact it has been 1 processor worse on only 4 instances). For $n = 10$ and $n = 20$, we have also computed the optimum solutions by a configuration-based ILP solved with CPLEX. For 82% of the instances with 10 tasks and 76% of the instances with $n = 20$, our $k$-RMM has found the optimum solution, and in the remaining cases it only fell short by one processor. Looking at the average processor load, i.e. the total utilization divided by the number of allocated processors, in Fig. 1, one can see that our $k$-RMM algorithm uses the processor much more efficiently than the other approximation algorithms.

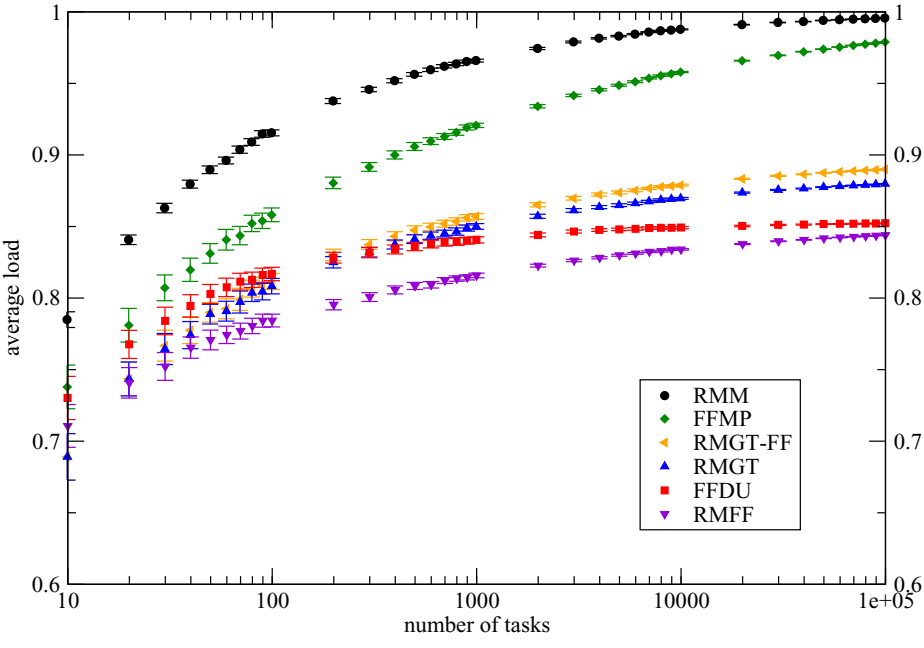



**Fig. 1.** A comparison of our algorithm with the ones known from the literature w.r.t. the average processor load

Figure 1 suggests that the average load for $k$-RMM converges to 1 as $n$ goes to infinity. In fact, it is not hard to prove that the *waste* of $k$-RMM, i.e. the difference between the allocated processors and the total utilization, scales sub-linearly with the number of tasks on random instances. More precisely, the same bound of $O(n^{3/4}\log^{3/8} n)$ for the waste of FFMP, which has been shown in [19], also holds for our new algorithm. However, experiments suggest that this bound for $k$-RMM might be something closer to $\sqrt{n}$. A further interesting open question is whether there exists an asymptotic PTAS for the rate-monotonic multiprocessor scheduling problem.

## References

1. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20(1), 46–61 (1973)
2. Buttazzo, G.: Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series) (2004)
3. Garey, M.R., Graham, R.L., Johnson, D.S., Yao, A.C.C.: Resource constrained scheduling as generalized bin packing. J. Combin. Theory Ser. A 21, 257–298 (1976)
4. Johnson, D.S.: Near-optimal bin packing algorithms. PhD thesis, MIT, Cambridge, MA (1973)
5. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1+\varepsilon$ in linear time. Combinatorica 1(4), 349–355 (1981)

6. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: FOCS 1982, pp. 312–320. IEEE, Los Alamitos (1982)
7. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin-packing—an updated survey. In: Algorithm Design for Computer System Design. Springer, Heidelberg (1984)
8. Liebeherr, J., Burchard, A., Oh, Y., Son, S.H.: New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. Comput. 44(12), 1429–1442 (1995)
9. Liu, J.: Real-Time Systems. Prentice Hall PTR, Upper Saddle River (2000)
10. Bini, E., Buttazzo, G., Buttazzo, G.: A hyperbolic bound for the rate monotonic algorithm. In: ECRTS 2001, p. 59 (2001)
11. Eisenbrand, F., Rothvoß, T.: Static-priority Real-time Scheduling: Response Time Computation is NP-hard. In: RTSS (2008)
12. Audsley, A.N., Burns, A., Richardson, M., Tindell, K.: Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal, 284–292 (1993)
13. Fisher, N., Baruah, S.: A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In: ECRTS 2005, pp. 117–126 (2005)
14. Baruah, S.K., Pruhs, K.: Open problems in real-time scheduling. Journal of Scheduling (2009)
15. Leung, J.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Inc., Boca Raton (2004)
16. Dhall, S.K., Liu, C.L.: On a real-time scheduling problem. Operations Research 26(1), 127–140 (1978)
17. Dhall, S.K.: Approximation algorithms for scheduling time-critical jobs on multiprocessor systems. In: Leung, J.Y.T. (ed.) Handbook of Scheduling — Algorithms, Models, and Performance Analysis. Chapman & Hall/CRC, Boca Raton (2004)
18. Oh, Y., Son, S.H.: Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. 9(3), 207–239 (1995)
19. Karrenbauer, A., Rothvoß, T.: An average-case analysis for rate-monotonic multiprocessor real-time scheduling. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 432–443. Springer, Heidelberg (2009)
20. Eisenbrand, F., Rothvoß, T.: A PTAS for static priority real-time scheduling with resource augmentation. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 246–257. Springer, Heidelberg (2008)
21. Baruah, S., Goossens, J.: Scheduling real-time tasks: Algorithms and complexity. In: Handbook of Scheduling — Algorithms, Models, and Performance Analysis (2004)
22. Gabow, H.N.: Data structures for weighted matching and nearest common ancestors with linking. In: SODA 1990 (1990)
23. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. John Wiley, New York (1997)
24. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Algorithms and Combinatorics, vol. 24. Springer, Heidelberg (2003)

# Online Tracking of the Dominance Relationship of Distributed Multi-dimensional Data

Tak-Wah Lam*, Chi-Man Liu, and Hing-Fung Ting

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

**Abstract.** We consider the online problem for a root (or a coordinator) to maintain a set of filters for the purpose of keeping track of the dominance relationship of some distributed multi-dimensional data. Such data keep changing from time to time. The objective is to minimize the communication between the root and the distributed data sources. Assume that data are chosen from the $d$-dimensional grid $\{1, 2, \cdots, U\}^d$, we give an $O(d \log U)$-competitive algorithm for this online problem. The competitive ratio is asymptotically tight as it is relatively easy to show an $\Omega(d \log U)$ lower bound.

## 1 Introduction

In this paper, we study competitive algorithms for online tracking of distributed data, which is a relatively new class of online problems with the following setting. There are a number of observers (or data streams), each of which keeps track of a private function whose value changes from time to time. At any time, the observers may need to communicate with a root (or coordinator) to enable it to recover some kind of information about the current values of the functions tracked by the observers. For example, consider a wireless sensor network monitoring the temperature of different places and the root wants to know which place has the lowest temperature. Depending on the information to be recovered, the root may not need to know the accurate values of the observers, and this would allow the root and the observers to save some unnecessary communication. In general, we need online algorithms to determine when and how the observers communicate to the root so as to minimize the total communication cost for tracking them.

Based on the above setting, this paper studies a problem called the online dominance tracking problem. This problem is formulated more intuitively in a geometric way: There are $k$ observers, each keeps track of a point moving in a $d$-dimensional grid. Let $D = \{1, 2, \ldots, d\}$. At any time, consider two observers $b_1$ and $b_2$ reporting the points with coordinates $(p_1, p_2, \ldots, p_d)$ and $(q_1, q_2, \ldots, q_d)$, we say that $b_1$ dominates $b_2$ if $p_i < q_i$ for some $i \in D$ and $p_j \leq q_j$ for all other $j \in D$. Furthermore, we say that $b_1$ dominates $b_2$ over a nonempty subset $D'$ of $D$ if, when projected to the dimensions in $D'$, $b_1$ dominates $b_2$. Consider any set $H$ of $k$ hyper-rectangles (a hyper-rectangle is the Cartesian product of $d$ intervals,

* Partially supported by the GRF Grant HKU-713909E.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 178–189, 2011.

one from each dimension), each encloses the current point of an observer. We say that $H$ forms a set of *dominance-preserving filters*, or simply *filters*, if no matter where the points tracked by the observers move within their filters, the dominance relationship between any two observers over any $D' \subseteq D$ remains the same (see the formal definition in Section 2). For example, the smallest filter is the hyper-rectangle containing only the current point of an observer. In general, we are interested in big filters.

We are now ready to define the online dominance tracking problem. At any time, the root and the $k$ observers want to maintain a set of $k$ dominance-preserving filters. Based on these filters, the root can answer any query on the dominance relationship of any two observers over any subset of dimensions. Filters save unnecessary communication in the following sense. As long as an observer's point is within its filter, it does not need to communicate to the server about the exact location of the current point. When a point moves out of its filter (i.e., we call this a filter violation), it must inform the root, which then computes a new filter for the observer. To do that, the root possibly needs to probe some other observers for their current points and update some of the filters eventually. The problem is how to compute the most appropriate filters so that the total number of messages exchanged with the root over a period of time is minimized. Note that this is an online problem as the computation of the filters cannot assume the knowledge of future points.

A trivial algorithm is to use the current point of each observer as its filter. Then every slight movement at an observer would trigger communication, and using such a filter is the same as communicating to the root for every possible movement. In this paper we study online algorithms with better communication cost. To analyze the performance, we compare the online algorithm against an optimal offline algorithm *Opt*, which is given in advance the point of every observer at every time step. Note that each time *Opt* modifies the filter of an observer, it uses one message (as the root has to inform its observer). Thus the communication cost of *Opt* is the total number of times it changes the filters. Denote the time interval in concern as $I$, and the filters used by *Opt* at time $t$ as $R_1^t, R_2^t, \ldots, R_k^t$; then *Opt* minimizes $\sum_{t \in I} |\{j \mid R_j^{t+1} \neq R_j^t\}|$ (i.e., the sum over all time $t$ of the number of filter changes from time $t$ to time $t+1$). For any $c \geq 1$, an online algorithm is said to be $c$-competitive if, over any sequence of possible movements of the points, its number of messages exchanged is at most $c$ times that of *Opt*.

Assume that the coordinates of the points are chosen from the $d$-dimensional grid $\{1, 2, \cdots, U\}^d$. It is relatively easy to show that any online algorithm for the online dominance tracking problem is $\Omega(d \log U)$-competitive on communication cost. The main result of this paper is an $O(d \log U)$-competitive algorithm. It is worth-mentioning that this online algorithm remains competitive if we change the problem to maintaining filters that preserve the dominance counts of the observers, where the dominance count of an observer is the number of other observers dominating it. Furthermore, the algorithm (and its competitiveness) is robust against the definition of dominance. In particular, it works for the following simple definition: $b_1$ is said to dominate $b_2$ if $p_j \leq q_j$ for all $j \in D$.

**Related work.** Distributed online tracking finds applications in database and network monitoring. In the database and algorithms literature, there are algorithms with worst-case performance analysis for problems like heavy hitters, quantiles, and threshold counts [3,5,4,6,2]. Yi and Zhang [7] are the first to apply competitive analysis to distributed online tracking. It is natural because for many online tracking problems, different algorithms (including the offline algorithm) would have more or less the same communication cost in some worst cases, and the usual worst-case measure of performance may not reflect which algorithms are better. Similar situation has happened in other online problems like online paging (see, e.g., [1] ).

The pioneering work of Yi and Zhang [7] used competitive analysis to study the following tracking problem. Each observer $b_i$ tracks a multi-valued integer function over time, denoted as $b_i(t)$. The observer needs to communicate with the root such that at any time $t$, the root can have an estimate $\hat{b}_i(t)$ of $b_i(t)$ with $|b_i(t) - \hat{b}_i(t)| \leq \Delta$, where $\Delta$ is a pre-specified error bound. Yi and Zhang gave an online algorithm for each observer to determine when to update the root. Their algorithm is $O(d^2 \log(d\Delta))$-competitive, where $d$ is the number of dimensions. In other words, the total communication cost of this algorithm over any period of time is at most $O(d^2 \log(d\Delta))$ times that of the optimal offline algorithm, which is given in advance the values of $b_i(t)$ for all time $t$ to determine the best strategy for an observer to communicate to the root.

We observe that for some applications, the values of the observers' functions (or points) are not important, instead their relationships are. Furthermore, the relationship of the observers may remain the same even if their values change a lot from time to time. Although one can apply the algorithm of Yi and Zhang to track the individual functions and then deduce their relationship, the root obtains over-detailed information and the communication would be too much to allow any competitive algorithm. It is perhaps not surprising that the problem of tracking the quantitative information (e.g., the values of the $b_i(t)$'s) is indeed different from the problem of tracking the qualitative information (e.g., the dominance relationship of the $b_i(t)$'s).

The online dominance tracking problem studied in this paper is motivated by the work of Zhang et al. [8], who are the first to exploit filters to track another form of dominance relationship called skylines. However, our approach is very different from theirs. They assumed the knowledge that the functions (or the points) are changing according to a uniform distribution and gave algorithms to compute the filters accordingly. These algorithms performed well empirically when distribution is satisfied; however, in the worst case these algorithms have competitive ratios with growth rate in the form of $k^d$. For future work, we believe that it is interesting to study competitive algorithms that assume some knowledge of the data distribution and try to predict the data.

**Organization of the paper.** Section 2 presents the $\Omega(d \log U)$ lower bound. In the rest of this paper, we restrict our attention to two-dimensional data when describing our algorithm. In particular, Section 3 is about some structural concepts and properties, Section 4 describes the algorithm, and Section 5 gives the

analysis. The details of generalizing the two-dimensional algorithm to higher dimensions will be given in the full paper.

**Details of the observers-and-root model.** Before we show the lower bound and upper bound results, we would like to detail how tracking works. An observer only communicates with the root, but not with the other observers. An observer starts with an empty filter. Whenever the point (or function) it is tracking moves out of the filter (we call this a filter violation), it sends a message to the root reporting the position of the current point and waits for a new filter from the root. An observer may occasionally receive a probe from the root, then it would report the position of the current point to the root (even though there is no filter violation) and then waits for a new filter. From the root's perspective, in each time step, if no observer reports a filter violation then it has nothing to update, and it can answer any dominance query based on the existing filters. When the root receives some filter violations, it finds out which additional observers it wants to probe and sends them a request. After hearing from those observers, the root computes the new filters and sends them to the corresponding observers.

Regarding the communication cost over a period of time, we count the number of messages between the root and the observers, or equivalently, the total number of filter violations, probes, and filter updates. For the optimal offline algorithm *Opt*, it is given how the points are moving in advance, and we only count its total number of filter updates.

## 2 Definitions and Lower Bound

Recall that we consider points chosen from a $d$-dimensional grid. At any time $t$, let $p_1, \ldots, p_k$ be the points currently tracked by the observers $b_1, \ldots, b_k$, respectively. Consider any $k$ hyper-rectangles $R_1, \ldots, R_k$ containing $p_1, \ldots, p_k$, respectively. We say that these rectangles form a set of *filters* if, for any two observers $b_i$, $b_j$ and for any subset of dimensions $D'$, the truth value of the statement "$b_i$ dominates $b_j$ over $D'$" remains the same no matter where the points of $b_i$ and $b_j$ move within $R_i$ and $R_j$, respectively.

We say that two hyper-rectangles $A$ and $B$ *overlap in the $\ell$-th dimension* if there exist two points $p \in A$, $q \in B$ such that $p$ and $q$ have the same coordinate in the $\ell$-th dimension. We say that $A$ and $B$ *overlap* if they overlap in at least one dimension. Below is a trivial yet useful property of filters.

**Fact 1.** *Let $p_1, \ldots, p_k$ be the points currently tracked by the $k$ observers. Consider any $k$ hyper-rectangles $R_1, \ldots, R_k$ that contain $p_1, \ldots, p_k$, respectively.*

- *Suppose that these $k$ points have distinct coordinates in every dimension. Then $R_1, \ldots, R_k$ form a set of filters if and only if no two of them overlap.*
- *In general, $R_1, \ldots, R_k$ form a set of filters if and only if for any pair of $b_i$ and $b_j$, (i) if $p_i$ and $p_j$ have different coordinates in the $\ell$-th dimension, $R_i$ and $R_j$ do not overlap in the $\ell$-th dimension; (ii) otherwise, $R_i$ and $R_j$, when projected on the $\ell$-th dimension, become a single-point interval containing exactly the $\ell$-th coordinate of $p_i$.*

The following is a lower bound result on maintaining filters online.

**Lemma 1.** *Any online algorithm $A$ for the online dominance tracking problem over the $d$-dimensional grid $\{1, 2, \ldots, U\}^d$ must have a competitive ratio at least $\frac{1}{2}d(\log U - 1)$.*

*Proof.* Suppose $A$ needs to track two points. Below we describe an adversary that gives a sequence of movement of the two points such that (i) $A$ needs to make at least $d(\log U - 1)$ filter updates, and (ii) we can find a filter for each point such that the point only moves within this filter. Note that the offline algorithm can use two filter updates to send the filters in (ii) to the observers at the beginning, and then no more filter updates or probings are necessary. The lemma follows.

Initially, there is a point $P$ at $(1, 1, \ldots, 1)$ and another point $Q$ at $(U, U, \ldots, U)$. The adversary checks the filters assigned by $A$. If $Q$ has a filter $[x_1, y_1] \times [x_2, x_2] \times \ldots \times [x_d, y_d]$ with $x_1 \leq U/2$, then the adversary moves $P$ to $(U/2, 1, \ldots, 1)$; otherwise it moves $Q$ to $(U/2, U, \ldots, U)$. Then, $A$ needs to make one filter updates (in order to restore the dominance-preserving property for the former case, and make sure $Q$ is within its filter for the latter), and the distance between $P$ and $Q$ is reduced by half in the first dimension. Repeating this step $\log U - 1$ times, we conclude that $A$ has at least $\log U - 1$ filter updates before the distance between $P$ and $Q$ in the first dimension becomes 1. Repeating this argument for the other dimensions, we conclude that $A$ needs at least $d(\log U - 1)$ filter updates before $P$ and $Q$ is next to each other, i.e., $P$ is at $(p_1, p_2, \ldots, p_d)$ and $Q$ is at $(p_1 + 1, p_2 + 1, \ldots, p_d + 1)$. Note that during the process, $P$ is moving within the hyper-rectangle $[1, p_1] \times [1, p_2] \times \ldots \times [1, p_d]$ and $Q$ within $[p_1 + 1, U] \times [p_2 + 1, U] \times \ldots \times [p_d + 1, U]$. □

## 3 Structures and Crossings

To ease our discussion, we first focus on two-dimensional points and assume that at any time, the $k$ points tracked by the observers have distinct x-coordinates and distinct y-coordinates. We will show how to remove the distinctness assumption in Section 5.

For an observer $b$, we denote its point at time $t$ to be $b^t$, and we refer to the x-coordinate and y-coordinate of $b^t$ as $X(b^t)$ and $Y(b^t)$, respectively. We say that $c$ is the *left neighbor* of $b$ at time $t$ if $c^t$ is closest to $b^t$ among all streams on the left of $b^t$ after projecting onto the x-axis, i.e. $X(c^t)$ is the largest among all streams with x-coordinate strictly less than $X(b^t)$. Neighbors in the other 3 directions are defined similarly. That is, each observer can have up to 4 neighbors. Furthermore, we denote the filter of $b$ at time $t$ as $R_b^t$ or $R_b$ (if it is clear that $t$ is the current time).

In designing an online algorithm, a primary concern is how to limit those unnecessary filter violations and updates, especially when the observers have no drastic movement, and the optimal offline algorithm *Opt* does not need any filter update. In this section, we show a strategy of setting a filter such that a filter

violation of an observer is always associated with a crossing event of the observer or its neighbor, and every observer has at most $O(\log U)$ crossing events during any period *Opt* has no filter update for that observer. This allows us to bound the number of filter violations. In Section 4, we will further present the details of our algorithm and show how to upper bound the number of extra probes and filter updates (not due to filter violation).

### 3.1 Cell Intersection and Smallest Rectangles

We define the notions of *cell intersection* and *smallest rectangle*, which depend on the inputs (but not the algorithms) and are used to upper bound and lower bound any possible filter.

**Definition 1.** *The* cell *of an observer $b$ at time $t$, denoted by $C_b^t$, is the largest rectangle containing $b^t$ but not overlapping with the current point of any other observer. Equivalently, $C_b^t$ is the largest rectangle containing $b^t$ but not overlapping with the point of any of $b$'s current neighbors.*

The following is a corollary of Fact 1.

**Corollary 1.** *Let $R_1^t, \ldots, R_k^t$ be a set of filters at time $t$. Then, for each observer $b$, $R_b^t \subseteq C_b^t$.*

**Definition 2.** *Let $t_0$ be some reference time. For any time $t \geq t_0$, define the* cell intersection *of $b$ over the time interval $[t_0, t]$, denoted by $A_b(t_0, t)$, to be the intersection of the cells $C_b^{t_0}, \ldots, C_b^t$. Note that $A_b(t_0, t)$ is also a rectangle.*

Applying Corollary 1 to *Opt*'s filters gives the following corollary.

**Corollary 2.** *Suppose that Opt uses the same filter $R_b^*$ for an observer $b$ throughout a period $[t_0, t]$. Then $R_b^* \subseteq A_b(t_0, t)$.*

Intuitively, $A$ gives an upper bound for $R$. The following definition gives a lower bound.

**Definition 3.** *Let $t_0$ be some reference time. For $t \geq t_0$, define the* smallest rectangle *of $b$ over $[t_0, t]$, denoted by $S_b(t_0, t)$, to be the smallest axis-parallel rectangle enclosing $b^{t_0}, \ldots, b^t$.*

The following lemma is the key to obtain a lower bound on the number of filter updates in *Opt*.

**Lemma 2.** *Suppose that Opt uses the same filter for an observer $b$ throughout a period $[t_0, t]$. Then $S_b(t_0, t) \subseteq A_b(t_0, t)$.*

*Proof.* Let $R_b^*$ be $b$'s filter in *Opt* during this period. By the definition of $S_b$, we have $S_b(t_0, t) \subseteq R_b^*$, since the filter always contains the current point. By Corollary 2, we have $R_b^* \subseteq A_b(t_0, t)$. □

### 3.2 Crossings

We now describe a mid-point strategy for setting filters. Suppose that *Opt* uses the same filter for an observer $b$ starting from time $t_0$ up to current time $t$. Consider any time $t'$ in $[t_0, t]$, $A_b(t_0, t')$ is monotonically shrinking and $S_b(t_0, t')$ is monotonically expanding as $t'$ increases. By Lemma 2, $S_b(t_0, t)$ is always enclosed by $A_b(t_0, t)$. For the online algorithm, a natural way to set $b$'s filter at time $t$ would be the mid way between the current $A_b(t_0, t)$ and $S_b(t_0, t)$.[1] Below we show that such an online strategy cannot cause too many filter violations as each of them is associated with a crossing event (to be defined below), reducing the gap between $A_b(t_0, t)$ and $S_b(t_0, t)$ by half.

When $t = t_0$, we define the mid-rectangle $M_b$ as follows: the left boundary of $M_b$ is the vertical line passing through the mid-point of the left boundaries of $A_b(t_0, t_0)$ and $S_b(t_0, t_0)$, and the other three boundaries are defined similarly. By definition, we have the inclusion relation that $S_b(t_0, t_0) \subseteq M_b \subseteq A_b(t_0, t_0)$. As time goes by, the smallest rectangle would get bigger and the cell intersection would get smaller. When the inclusion relation with $M_b$ is violated, we say a crossing occurs and we obtain a new $M_b$. See the general definition below.

**Definition 4.** *Consider any time $t > t_0$. Let $t_1$ be the last time before $t$ a crossing occurred (if $t_1$ does not exist, let $t_1 = t_0$), and let $M_b$ be the mid-rectangle with respect to $A_b(t_0, t_1)$ and $S_b(t_0, t_1)$. A* crossing *occurs at $t$ if* **(i)** *$b^t$ moves out of $M_b$; or* **(ii)** *$C_b^t \not\supseteq M_b$.*

We have the following bound on the number of crossings.

**Lemma 3.** *Suppose that Opt uses the same filter for an observer $b$ throughout a period $[t_0, t]$. Then the number of crossings in $b$ during $[t_0, t]$ is $O(\log U)$.*

*Proof.* By Lemma 2, $S_b(t_0, t) \subseteq A_b(t_0, t)$. Consider the four distances between corresponding boundaries of $A_b$ and $S_b$. Every crossing in $b$ reduces one (or more) of the distances by at least half. The initial distances are at most $U$, thus after at most $4 \log U$ crossings, all distances become 0. Any crossing after that would lead to $S_b \not\subseteq A_b$. □

The above lemma naturally suggests an online algorithm to set the filter to be the mid-rectangle $M_b$ after a crossing occurs. Then a filter violation would imply a crossing, and there would be at most $O(\log U)$ filter violations within a period during which *Opt* does not change the filter. However, this will not work since the mid-rectangles of two observers may overlap. In this case, we need to set the filters of one (or both) of them to be a proper sub-rectangle of its mid-rectangle. Intuitively, we "squeeze" a filter until it no longer overlaps with any other mid-rectangle. Formally, let $c$ be the left neighbor of $b$. At any time, we say that $c$ squeezes $b$ from the left if $M_c$ overlaps with $M_b$ in the x-dimension. In that case,

[1] For the time being, we assume that the root knows $S_b(t_0, t)$ and $A_b(t_0, t)$. In the actual algorithm, we will show how to replace them with reasonable approximations that can be maintained without extra communication.

shrink the left boundary of $b$'s filter to $r+1$, where $r$ is the right boundary of $M_c$. Symmetrically, we also shrink the right boundary of $c$ to the left boundary of $M_b$ minus one. Note that $M_c$ and $M_b$ may change later, but as long as they overlap, their filters will be shrunk according to their $M$'s boundaries. We perform the same operation to the other three neighbors of $b$.

The squeezing operation causes $b$'s filter to be strictly smaller than $M_b$. If, at some later time, $b$ moves out of its filter but still lies inside $M_b$, a filter violation would occur without a crossing. We call this a *no-crossing violation*. The following lemma suggests that we can charge these no-crossing violations against the crossings of other observers.

**Lemma 4.** *Suppose that at time $t$, $b$ has a no-crossing violation on the left. Let $c$ be the left neighbor of $b$ at $t-1$. Then $c$ has a crossing at $t$. Similar statements hold for other directions.*

*Proof.* Suppose that the left boundary of $R_b^{t-1}$ is $r+1$. Then the right boundary of $M_c^{t-1}$ is $r$. At $t$, $b$ violates its left filter boundary, thus $X(b^t) \leq r$. $b^t$ still lies in $M_b^{t-1}$, so $b^t$ overlaps with $M_c^{t-1}$ in the x-dimension. We see that a crossing occurs in $c$ at $t$, since $C_c^t \not\supseteq M_c^{t-1}$. □

In summary, whenever a filter violation occurs at an observer, we can charge it to a crossing of the observer itself or its neighbor. On the other hand, a filter update can be triggered by a filter violation of the observer itself or its neighbor. Below we will give an algorithm based on the above concepts, which would allow us to upper bound the number of filter violations and filter updates.

There is however a technical issue. In the online algorithm, the root does not maintain $S$ and $A$, as doing so would cost too many messages. Instead, the root maintains an approximate smallest rectangle $L_b(t_0, t)$, which is always contained in $S_b(t_0, t)$. It also maintains an approximate cell intersection $U_b(t_0, t)$, which always contains $A_b(t_0, t)$. The idea is simple: we only update $L_b$ or $U_b$ if doing so would lead to a crossing in $b$. The tricky part is to find a small enough $U$ with few communication so that the corresponding filter would fall completely in the current cell, and a large enough $L$ so that the filter would contain the current point.

## 4 Algorithm

We first give a high level framework of our $O(\log U)$-competitive algorithm. From the root's perspective, each observer $b$ proceeds in rounds. At any time $t$, the root remembers, for each observer $b$, a reference time $t_0(b)$ (or simply $t_0$ if the observer is clear from the context) $\leq t$, which is the start time of the current round of $b$, and the root is interested in what has happened to $b$ during $[t_0, t]$. In particular, it keeps four rectangles $R_b(t_0, t), L_b(t_0, t), U_b(t_0, t)$ and $M_b(t_0, t)$ (or simply $R_b, L_b, U_b$ and $M_b$ when there is no confusion). $R_b$ is the filter assigned to $b$. The root would ensure that $L_b \subseteq S_b$ and $A_b \subseteq U_b$. Intuitively, $L_b$ is a lower bound for $S_b$ and $U_b$ an upper bound of $A_b$; the algorithm uses them to

approximate $S_b$ and $A_b$. $M_b$ is the mid-rectangle of $L_b$ and $U_b$. Note that the observer $b$ itself only knows $R_b$ among these four rectangles.

Within each round, $L_b$ is monotonically expanding, $U_b$ is monotonically shrinking, and we try to keep the invariant $L_b \subseteq U_b$. When this invariant is violated, this implies $S_b(t_0, t) \not\subseteq A_b(t_0, t)$, and by Lemma 2, *Opt* must have updated $b$'s filter within the current round. Then, the root starts a new round for $b$, reinitializing $L_b$ to the rectangle that encloses only the current point of $b$, and $U_b$ to the current cell of $b$. Thus, within each round, *Opt* updates $b$'s filter at least once, and $b$ has at most $O(\log U)$ crossings (see Lemma 3). We will argue below that the total number of filter updates and probes (over all observers) is asymptotically bounded by the total number of crossings (over all observers). It follows that the competitive ratio of the online algorithm is $O(\log U)$.

To maintain the four rectangles, the root at every time step $t$, after receiving all the violation notifications, carries out the following steps.

(1) Probe additional observers (if necessary) and determine the new dominance relationships for all pairs of observers.
(2) For each observer $b$, check if $b$ has a crossing, and update $L_b$, $U_b$, $M_b$ if so. Start a new round for $b$ if $L_b \not\subseteq U_b$, and in such case probe the four neighbors of $b$ (if not yet done), and then initialize the rectangles $L_b = b^t$ and $U_b = C_b^t$.
(3) For each observer $b$, set $R_b = M_b$ and then squeeze it from every direction.

The algorithm is shown in Figure 4. The following lemma states the properties maintained by the algorithm, which also implies the correctness of the algorithm.

**Lemma 5.** *The following properties hold at the end of each time step $t$ for every observer $b$.*

(i) $L_b^t \subseteq U_b^t$;
(ii) $M_b^t$ *is the mid-rectangle of* $L_b^t$ *and* $U_b^t$;
(iii) $b^t \in M_b^t$ *and* $M_b^t \subseteq C_b^t$;
(iv) $b^t \in R_b^t$ *and* $R_b^t \subseteq M_b^t$;
(v) $R_b^t$ *does not overlap with* $M_c^t$ *for any observer* $c \neq b$.

Properties (iv) and (v) imply that no two $R_i$'s overlap. By Fact 1, these $R_i$'s form a set of filters.

*Proof.* Property (i): When the root starts a new round for $b$, this property holds. As soon as this property is violated, the root starts a new round for $b$ again. Hence this property always holds. Property (ii): $M_b$ is assigned to be the mid-rectangle of $L_b$ and $U_b$ whenever either is updated (Step 2). Property (iii): If we start a new round for $b$, this holds according to the initialization. Otherwise, if $b$ has no crossing, then $M_b^t = M_b^{t-1}$, and $b^t \in M_b^{t-1} \subseteq C_b^t$ by the definition of crossing. Otherwise, $b$ has a crossing. Step 2 guarantees that $b^t \in L_b^t \subseteq M_b^t$ and $M_b^t \subseteq U_b^t \subseteq C_b^t$. Property (iv): It is obvious that $R_b^t \subseteq M_b^t$. By property (iii), $b^t \in M_b^t$. Applying property (iii) to any neighbor $c$ of $b$ shows that $M_c^t$ does not overlap with $b^t$. Therefore $b^t$ still lies inside the squeezed filter $R_b^t$. Property (v): Step 3 guarantees that this property holds for any neighbor $c$ of $b$. If $c$ is not a neighbor of $b$, then $C_b^t$ and $C_c^t$ are disjoint (separated by some observer between $b$ and $c$). By property (iii), $M_b^t$, and so $R_b^t$, does not overlap with $M_c^t$. □

1. **Computing new dominance relationships.** For every pair of observers $b, c$, the root computes their new dominance relationship as follows. If neither of them violates, their dominance relationship remains the same as time $t-1$. If both of them have filter violations, their dominance relationship can be determined from $b^t$ and $c^t$ without probing. Suppose exactly one of them violates, say $b$. Then, their dominance relationship can be determined without probing if $b^t$ does not overlap with $R_c^{t-1}$; otherwise, we probe $c$. The neighbors of all observers are also identified.
2. **Updating $L$, $U$, $M$ due to crossings.** First, the root checks, for every observer $b$, whether $b$ has a crossing, or equivalently, whether either (i) $b^t$ moves out of $M_b^{t-1}$, or (ii) $c^t$ overlaps with $M_b^{t-1}$ for some neighbor $c$ of $b$. Note that after receiving the filter violation notifications, the root can check (i) and (ii) without probing because condition (i) is a filter violation for $b$, and by property (v) of Lemma 5 below (applied to $c$ at time $t-1$), condition (ii) implies $c$ has a filter violation, and thus the root already knows $c^t$ for checking (ii).
   - If $b$ has no crossing, set $L_b^t = L_b^{t-1}$, $U_b^t = U_b^{t-1}$, $M_b^t = M_b^{t-1}$.
   - Otherwise, set $L_b^t$ to be the smallest rectangle enclosing both $L_b^{t-1}$ and $b^t$. Set $U_b^t$ to be $U_b^{t-1} \cap C_b^t$. $b^t$ and $C_b^t$ can be determined by probing $b$ and all its neighbors. If $L_b^t \not\subseteq U_b^t$, start a new round for $b$, and reset $L_b^t = b^t$ and $U_b^t = C_b^t$. Then, set $M_b^t$ to be the mid-rectangle of $L_b^t$ and $U_b^t$.
3. **Squeezing $R$.** In Step 2, $L$, $U$ and $M$ were fixed for all observers. In this step, the root initializes every filter and squeezes it from each direction. For each observer $b$, do the following steps. Initialize $R_b^t = M_b^t$. If $b$ has a left (i.e. negative x) neighbor $c$, and $M_c^t$ overlaps with $M_b^t$ in the x-dimension, set the left boundary of $R_b^t$ to $r+1$, where $r$ is the right boundary of $M_c^t$. Do similarly for other directions.

**Fig. 1.** The online algorithm

## 5 Analysis

In this section, we prove that our algorithm is $O(\log U)$-competitive. The total cost of our online algorithm is the sum of the numbers of probes, filter violations, and filter updates. Since a filter violation is always followed by a filter update, we only count those updates for which there is no violation. We show that each of these quantities is asymptotically bounded by the total number of crossings for all observers. In previous sections, we argued that each observer has $O(\log U)$ crossings in a single round. Summing over all rounds for all observers, we can bound the total cost of our algorithm to be $O(r \log U)$, where $r$ is the total number of rounds for all observers since time 0. For the optimal offline algorithm *Opt*, we show that its cost is at least $r$, thus giving the claimed competitive ratio.

We first account for the number of probes in our algorithm. For convenience, we assume that a crossing always leads to a filter update, even if the resulting filter happens to remain unchanged (we see this as a bogus update).

**Lemma 6.** *For any time $t \geq 0$, the number of probes at $t$ is $O(1)$ times the number of filter updates at $t$.*

*Proof.* In Step 1, an observer $c$ is probed only if $R_c^{t-1}$ overlaps with $b^t$ for some $b \neq c$. By property (v) of Lemma 5 (applied to $c$), $R_c^{t-1}$ cannot be $c$'s filter at $t$ as it overlaps with $M_b^t \ni b^t$. Hence, $c$ needs a filter update at $t$.

In Step 2, when updating $L$'s and $U$'s, an observer $b$ is probed only if (i) $b$ has a crossing; or (ii) one of $b$'s neighbors has a crossing. Each observer has at most 4 neighbors. Each observer with a crossing needs a filter update.

In Step 2, when starting a new round for $b$, the root probes all neighbors of $b$ to determine $C_b^t$. This takes $O(1)$ probes. □

Next, we bound the number of filter violations (and the associated filter updates).

**Lemma 7.** *Suppose that observer $b$ has a filter violation at time $t$. Then, either $b$ or one of its neighbors at $t-1$ has a crossing at $t$.*

*Proof.* Suppose that $b$ has no crossing at $t$. Without loss of generality, assume that $b$ violates on the left. Lemma 4 implies that the left neighbor of $b$ at $t-1$ has a crossing at $t$. □

Each observer has at most 4 neighbors at any time. Lemma 7 implies that the number of filter violations is upper bounded by 5 times the number of crossings. This gives the following corollary.

**Corollary 3.** *Let $r$ be the total number of rounds since time 0. Then, the total number of filter violations since time 0 is $O(r \log U)$.*

The number of filter updates without violation can be bounded similarly.

**Lemma 8.** *Suppose that observer $b$ has a no-violation filter update at time $t$. Then, there must be a crossing at $t$ in (i) $b$; or (ii) one of $b$'s neighbors at $t$; or (iii) one of $b$'s neighbors at $t-1$.*

*Proof.* If $M_b^t \neq M_b^{t-1}$, $b$ has a crossing at $t$. We assume that $M_b^t = M_b^{t-1}$. Without loss of generality, suppose that the left boundaries of $R_b^t$ and $R_b^{t-1}$ are different. Let $c'$ and $c$ be the left neighbors of $b$ at $t-1$ and $t$, respectively. If $c \neq c'$, the right boundary of $M_{c'}^{t-1}$ is to the right of that of $M_c^{t-1}$, for $c'$ was the left neighbor of $b$ at $t-1$. However, $c$ becomes the left neighbor of $b$ at $t$, so the right boundary of $M_{c'}^t$ is to the left of that of $M_c^t$. Either $M_c$ or $M_{c'}$ changes at $t$, thus either $c$ or $c'$ has a crossing at $t$. If $c = c'$, then $M_c^t \neq M_c^{t-1}$; otherwise the left boundary of $R_b$ would not change. Thus, $c$ has a crossing at $t$. □

Similar to Corollary 3, the following is a corollary of Lemma 8.

**Corollary 4.** *Let $r$ be the total number of rounds since time 0. Then, the total number of no-violation filter updates since time 0 is $O(r \log U)$.*

The following is a corollary of Lemma 2, which helps to lower bound the number of filter updates in *Opt*.

**Corollary 5.** *Suppose that one of $b$'s rounds starts at $t_0$ and ends at $t_1$ due to $L_b(t_0, t_1+1) \not\subseteq U_b(t_0, t_1+1)$. Then, Opt updates $b$'s filter at least once during $[t_0+1, t_1+1]$.*

**Theorem 1.** *At any time $t \geq 0$, let $cost^*$ and cost be the number of messages used by Opt and our algorithm respectively since time 0. Then $cost \leq O(\log U) \times cost^*$.*

*Proof.* Let $r$ be the total number of rounds since time 0. By Lemma 6, Corollary 3 and Corollary 4, $cost = O(r \log U)$. By Corollary 5, $cost^*$ is at least the number of complete rounds, which is at least $r - k$. Adding the $k$ messages for filter initialization at time 0, we have $cost^* \geq r$, giving the desired inequality. □

**Remarks on distinct coordinates.** We have been assuming that all observers have distinct x- and distinct y-coordinates. To handle arbitrary coordinates, we need some modifications. We only consider the x-dimension here. We say that an observer is *hooked-up* if its x-coordinate is not unique among all observers. Fact 1 says that for any hooked-up observer $b$, the projection of $b$'s filter on the x-axis must be a single-point interval containing exactly $X(b^t)$. We can therefore set both the left and right boundaries of $U_b$ to be $X(b^t)$ in Step 2 of the algorithm as soon as $b$ becomes a hooked-up observer. In Step 3, we initialize the left and right boundaries of $R_b^t$ as above instead of using $M_b^t$'s boundaries. The analysis still goes through with a few modifications. For Lemma 7, if $b$ violates in either the left or right direction, the root starts a new round for $b$, which means that $b$ has a crossing. For Lemma 8, if $b$ has a no-violation filter update in either the left or right direction, then it is no longer hooked-up. This implies every observer it hooked up with has got a new x-coordinate, which, as discussed above, leads to a crossing.

## References

1. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (2005)
2. Chan, H., Lee, L., Lam, T., Ting, H.: Continuous monitoring of distributed data streams over a time-based sliding window. In: Proc. STACS, pp. 179–190 (2010)
3. Cormode, G., Garofalakis, M., Muthukrishnan, S., Rastogi, R.: Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In: Proc. SIGMOD, pp. 25–36 (2005)
4. Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed functional monitoring. In: Proc. SODA, pp. 1076–1085 (2008)
5. Keralapura, R., Cormode, G., Ramamirtham, J.: Communication-efficient distributed monitoring of thresholded counts. In: Proc. SIGMOD, pp. 289–300 (2006)
6. Yi, K., Zhang, Q.: Optimal tracking of distributed heavy hitters and quantiles. In: Proc. PODS, pp. 167–174 (2008)
7. Yi, K., Zhang, Q.: Multidimensional online tracking. In: Proc. SODA, pp. 1098–1107 (2009)
8. Zhang, Z., Cheng, R., Papadias, D., Tung, A.: Minimizing the communication cost for continuous skyline maintenance. In: Proc. SIGMOD, pp. 495–507 (2009)

# How to Play Unique Games on Expanders

Konstantin Makarychev[1] and Yury Makarychev[2]

[1] IBM T.J. Watson Research Center
[2] Toyota Technological Institute, Chicago

**Abstract.** In this paper, we improve a result by Arora, Khot, Kolla, Steurer, Tulsiani, and Vishnoi on solving the Unique Games problem on expanders. Given a $(1-\varepsilon)$-satisfiable instance of Unique Games with the constraint graph $G$, our algorithm finds an assignment satisfying at least a $1 - C\varepsilon/h_G$ fraction of all constraints if $\varepsilon < c\lambda_G$ where $h_G$ is the edge expansion of $G$, $\lambda_G$ is the second smallest eigenvalue of the Laplacian of $G$, and $C$ and $c$ are some absolute constants.

## 1 Introduction

In this paper, we study Unique Games on expander graphs.

**Definition 1 (Unique Games Problem).** *Given a constraint graph $G = (V, E)$ and a set of permutations $\pi_{uv}$ on the set $[k] = \{1, \dots, k\}$ (for all edges $(u, v)$), the goal is to assign a label (or state) $x_u$ from $[k]$ to each vertex $u$ so as to satisfy the maximum number of constraints of the form $\pi_{uv}(x_u) = x_v$. The value of a solution is the fraction of satisfied constraints.*

The famous Unique Games Conjecture (UGC) of Khot [8] states that for every positive $\varepsilon$ and $\delta$, there exists $k$ such that it is NP-hard to distinguish between the case where a $1-\varepsilon$ fraction of all constraints is satisfiable and the case where at most a $\delta$ fraction of all constraints is satisfiable. This conjecture has attracted a lot of attention since it implies strong inapproximability results for such fundamental problems as MAX CUT [9], Vertex Cover [10], Maximum Acyclic Subgraph [6], $k$-CSP [7] [11], which are not known to follow from more standard complexity assumptions. Several approximation algorithms for Unique Games were developed in a series of papers by Khot [8], Trevisan [12], Gupta and Talwar [5], Charikar, Makarychev and Makarychev [3], and Chlamtac, Makarychev and Makarychev [4]. These papers, however, did not disprove the Unique Games Conjecture.

In order to better understand Unique Games, we need to identify which instances of Unique Games are easy, and which instances are potentially hard (the quantitative measure of hardness of a family of instances is the "approximation guarantee" of the "optimal" algorithm for this family). That motivates the study of specific families of Unique Games. Arora, Khot, Kolla, Steurer, Tulsiani, and Vishnoi [1] disproved the UGC for Unique Games on spectral expanders. Specifically, they showed how given a $(1 - \varepsilon)$ satisfiable instance of Unique Games

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 190–200, 2011.

(i.e. an instance in which the optimal solution satisfies at least a $(1-\varepsilon)$ fraction of constraints), one can obtain a solution of value

$$1 - C\frac{\varepsilon}{\lambda_G}\log\left(\frac{\lambda_G}{\varepsilon}\right)$$

in polynomial time, here $C$ is an absolute constant and $\lambda_G$ is the second smallest eigenvalue of the Laplacian of $G$ (see Section 2 for definitions).

In this paper, we improve their result and show that, if the ratio $\varepsilon/\lambda_G$ is less than some universal positive constant $c$, one can obtain a solution of value

$$1 - C'\frac{\varepsilon}{h_G}$$

in polynomial time, here $h_G$ is the edge expansion of $G$. In general, $\lambda_G$ can be significantly smaller than $h_G$, then our result gives much better approximation guarantee. For example, if Cheeger's inequality (see below) is tight for a graph $G$, then $\lambda_G \approx h_G^2/8$; and

$$1 - C'\frac{\varepsilon}{h_G} \gg 1 - 8C\frac{\varepsilon}{h_G^2}\log\left(\frac{h_G^2}{8\varepsilon}\right) \approx 1 - C\frac{\varepsilon}{\lambda_G}\log\left(\frac{\lambda_G}{\varepsilon}\right).$$

Say, if $\varepsilon \approx \lambda_G$, the algorithm of Arora, Khot, Kolla, Steurer, Tulsiani, and Vishnoi satisfies only a small constant fraction of all constraints, while our algorithm satisfies almost all constraints. However, even if $\lambda_G \approx h_G$, our bound is asymptotically stronger, since

$$1 - C'\frac{\varepsilon}{h_G} \geq 1 - C'\frac{\varepsilon}{\lambda_G} > 1 - C'\frac{\varepsilon}{\lambda_G}\log\left(\frac{\lambda_G}{\varepsilon}\right)$$

(i.e., our bound does not have a $\log(\lambda_G/\varepsilon)$ factor).

### 1.1 Overview

In this section, we give an informal overview of the algorithm. The algorithm uses the standard SDP relaxation for Unique Games (see Section 2.2). The SDP solution gives a vector $u_i$ for every vertex $u$ and label $i$. For simplicity, let us consider so-called uniform case when all vectors $u_i$ have the same length. Then by scaling all vectors, we can assume that they are unit vectors, and thus vectors $u_1, \ldots, u_k$ (corresponding to one vertex) form an orthonormal frame.

For every two vertices $u$ and $v$, we say that labels $i$ and $j$ are *matched* if $\|u_i - v_j\|^2 < r$, where $r < 1$ is a small threshold value. Note that for every two labels $j_1$ and $j_2$,

$$\|u_i - v_{j_1}\|^2 + \|u_i - v_{j_2}\|^2 \geq \|v_{j_1} - v_{j_2}\|^2 = 2 > 2r.$$

Therefore, each label $i$ is matched with at most one label $j$ for fixed vertices $u$ and $v$. We denote this $j$ by $\sigma_{uv}(i)$ (if it exists).

Now we use a simple prorogation algorithm. We choose a random vector $u$ and assign it a random label $i$. Then we label each vertex $v$ with the label $\sigma_{uv}(i)$, if it is defined; and with an arbitrary label, otherwise. Let $X$ be the set of vertices $v$ s.t. $\sigma_{uv}(i)$ is defined. We prove (see Lemma 9) that for every edge $(v, w)$ with $v, w \in X$, our assignment satisfies the constraint between $v$ and $w$ w.h.p. if the contribution of the edge $(v, w)$ to the SDP objective is small. Intuitively, that happens because both vectors $v_{\sigma_{uv}(i)}$ and $w_{\sigma_{uw}(i)}$ are close to $u_i$, and therefore they are close to each other. On the other hand, the SDP contribution of the edge $(v, w)$ equals

$$\frac{1}{k}\sum_{j=1}^{k} \|v_j - w_{\pi_{vw}(j)}\|^2.$$

Thus if the SDP contribution is small then the vector $v_j$ should be close to $w_{\pi_{vw}}(j)$ for most labels $j$. Since each $v_{\sigma_{uv}(i)}$ is close only to $w_{\sigma_{uw}(i)}$, we have $\sigma_{uw}(i) = \pi_{vw}(\sigma_{uv}(i))$ w.h.p., that is, the constraint between $v$ and $w$ is satisfied.

The crucial step now is to prove that the set $X$ contains almost all vertices, and so we can ignore edges with one or two endpoints outside of $X$. First, we prove that the set $X$ is not very small in Lemma 5 (using a "global correlation" result of Arora, Khot, Kolla, Steurer, Tulsiani, and Vishnoi). Using a standard region growing argument we then show that the cut between $X$ and $V \setminus X$ is very small (if we choose the threshold $r$ randomly; see Lemma 7). Since the graph $G$ is an expander, that implies that either $X$ or $V \setminus X$ is very small. But we know that $X$ is not very small. We conclude that in fact $V \setminus X$ is very small (Lemma 8).

To deal with the general case — when vectors $u_i$ have different lengths — we use the vector normalization machinery developed by Chlamtac, Makarychev and Makarychev [4].

In Section 2, we give basic definitions and describe the semideifnite relaxation for Unique Games. In Section 3, we present the algorithm and its analysis.

## 2 Preliminaries

### 2.1 Expanders: Second Eigenvalue and Edge Expansion

We assume that the underlying constraint graph $G = (V, E)$ is a $d$-regular expander. The two key parameters of the expander $G$ are the edge expansion $h_G$ and the second eigenvalue of the Laplacian $\lambda_G$. The edge expansion gives a lower bound on the size of every cut: for every subset of vertices $X \subset V$, the size of the cut between $X$ and $|V \setminus X|$ is at least

$$h_G \times \frac{\min(|X|, |V \setminus X|)}{|V|}|E|.$$

It is formally defined as follows:

$$h_G = \min_{X \subset V} \left( \frac{|\delta(X, V \setminus X)|}{|E|} \Big/ \frac{\min(|X|, |V \setminus X|)}{|V|} \right),$$

here $\delta(X, V \setminus X)$ denotes the cut — the set of edges going from $X$ to $V \setminus X$. One can think of the second eigenvalue of the Laplacian matrix

$$L_G(u,v) = \begin{cases} 1, & \text{if } u = v \\ -1/d, & \text{if } (u,v) \in E \\ 0, & \text{otherwise.} \end{cases}$$

as of continuous relaxation of the edge expansion. Note that the smallest eigenvalue of $L_G$ is 0; and the corresponding eigenvector is a vector of all 1's, denoted by $\mathbf{1}$. Thus

$$\lambda_G = \min_{x \perp \mathbf{1}} \frac{\langle x, L_G x \rangle}{\|x\|^2}.$$

Cheeger's inequality,

$$h_G^2/8 \leq \lambda_G \leq h_G,$$

shows that $h_G$ and $\lambda_G$ are closely related; however $\lambda_G$ can be much smaller than $h_G$ (the lower bound in the inequality is tight).

### 2.2 Semidefinite Relaxation for Unique Games

We use the standard SDP relaxation for the Unique Games problem.

$$\text{minimize } \frac{1}{2|E|} \sum_{(u,v)\in E} \sum_{i=1}^{k} \|u_i - v_{\pi_{uv}(i)}\|^2$$

subject to

$$\forall u \in V\ \forall i,j \in [k], i \neq j \quad \langle u_i, u_j \rangle = 0 \tag{1}$$

$$\forall u \in V \quad \sum_{i=1}^{k} \|u_i\|^2 = 1 \tag{2}$$

$$\forall u,v,w \in V\ \forall i,j,l \in [k] \quad \|u_i - w_l\|^2 \leq \|u_i - v_j\|^2 + \|v_j - w_l\|^2 \tag{3}$$

$$\forall u,v \in V\ \forall i,j \in [k] \quad \|u_i - v_j\|^2 \leq \|u_i\|^2 + \|v_j\|^2 \tag{4}$$

$$\forall u,v \in V\ \forall i,j \in [k] \quad \|u_i\|^2 \leq \|u_i - v_j\|^2 + \|v_j\|^2 \tag{5}$$

For every vertex $u$ and label $i$ we introduce a vector $u_i$. In the intended integral solution $u_i = 1$, if $u$ is labeled with $i$; and $u_i = 0$, otherwise. All SDP constraints are satisfied in the integral solution; thus this is a valid relaxation. The objective function of the SDP measures what fraction of all Unique Games constraints is *not satisfied*.

## 3 Algorithm

We define the *earthmover distance* between two sets of orthogonal vectors $\{u_1, \ldots, u_k\}$ and $\{v_1, \ldots, v_k\}$ as follows:

$$\Delta(\{u\}_i, \{v\}_i) \equiv \min_{\sigma(i)\in\mathcal{S}_k} \sum_{i=1}^{k} \|u_i - v_{\sigma(i)}\|^2,$$

here $\mathcal{S}_k$ is the symmetric group, the group of all permutations on the set $[k] = \{1, \dots, k\}$. Given an SDP solution $\{u_i\}_{u,i}$ we define the earthmover distance between vertices in a natural way:

$$\Delta(u, v) = \Delta(\{u_1, \dots, u_k\}, \{v_1, \dots, v_k\}).$$

Arora et al. [1] proved that if an instance of Unique Games on an expander is almost satisfiable, then the average earthmover distance between two vertices (defined by the SDP solution) is small. We will need the following corollary from their results:

*For every $R \in (0, 1)$, there exists a positive $c$, such that for every $(1 - \varepsilon)$ satisfiable instance of Unique Games on an expander graph $G$, if $\varepsilon/\lambda_G < c$, then the expected earthmover distance between two random vertices is less than $R$ i.e.*

$$\mathbb{E}_{u,v \in V} [\Delta(u, v)] \le R.$$

In fact, Arora et al. [1] showed that $c \ge \Omega(R/\log(1/R))$, but we will not use this bound. Moreover, in the rest of the paper, we fix the value of $R < 1/4$. We pick $c_R$, so that if $\varepsilon/\lambda_G < c_R$, then

$$\mathbb{E}_{u,v \in V} [\Delta(u, v)] \le R/4. \tag{6}$$

Our algorithm transforms vectors $\{u_i\}_{u,i}$ in the SDP solution to vectors $\{\tilde{u}_i\}_{u,i}$ using a *vector normalization* technique introduced by Chlamtac, Makarychev and Makarychev [4]:

**Lemma 1.** *[4] For every SDP solution $\{u_i\}_{u,i}$, there exists a set of vectors $\{\tilde{u}_i\}_{u,i}$ satisfying the following properties:*

1. *Triangle inequalities in $\ell_2^2$: for all vertices $u$, $v$, $w$ in $V$ and all labels $i$, $p$, $q$ in $[k]$,*
$$\|\tilde{u}_i - \tilde{v}_p\|_2^2 + \|\tilde{v}_p - \tilde{w}_q\|_2^2 \ge \|\tilde{u}_i - \tilde{w}_q\|_2^2.$$
2. *For all vertices $u, v$ in $V$ and all labels $i, j$ in $[k]$,*
$$\langle \tilde{u}_i, \tilde{v}_j \rangle = \frac{\langle u_i, v_j \rangle}{\max(\|u_i\|^2, \|v_j\|^2)}.$$
3. *For all non-zero vectors $u_i$, $\|\tilde{u}_i\|_2^2 = 1$.*
4. *For all $u$ in $V$ and $i \ne j$ in $[k]$, the vectors $\tilde{u}_i$ and $\tilde{u}_j$ are orthogonal.*
5. *For all $u$ and $v$ in $V$ and $i$ and $j$ in $[k]$,*
$$\|\tilde{v}_j - \tilde{u}_i\|_2^2 \le \frac{2\|v_j - u_i\|^2}{\max(\|u_i\|^2, \|v_j\|^2)}.$$

*The set of vectors $\{\tilde{u}_i\}_{u,i}$ can be obtained in polynomial time.*

Now we are ready to describe the rounding algorithm. The algorithm given an SDP solution, outputs an assignment of labels to the vertices.

**Approximation Algorithm**

**Input:** an SDP solution $\{u_i\}_{u,i}$ of cost $\varepsilon$.
**Initialization**

1. Pick a random vertex $u$ (uniformly distributed) in $V$. We call this vertex *the initial vertex*.
2. Pick a random label $i \in [k]$ for $u$; choose label $i$ with probability $\|u_i\|^2$. Note that $\|u_1\|^2 + \cdots + \|u_k\|^2 = 1$. We call $i$ *the initial label*.
3. Pick a random number $t$ uniformly distributed in the segment $[0, \|u_i\|^2]$.
4. Pick a random $r$ in $[R, 2R]$.

**Normalization**

5. Obtain vectors $\{\tilde{u}_i\}_{u,i}$ as in Lemma 1.

**Propagation**

6. For every vertex $v$,
   - Find all labels $p \in [k]$ such that $\|v_p\|^2 \geq t$ and $\|\tilde{v}_p - \tilde{u}_i\|^2 \leq r$. Denote the set of $p$'s by $S_v$:

$$S_v = \left\{p : \|v_p\|^2 \geq t \text{ and } \|\tilde{v}_p - \tilde{u}_i\|^2 \leq r\right\}.$$

   - If $S_v$ contains exactly one element $p$, then assign the label $p$ to $v$.
   - Otherwise, assign an arbitrary (say, random) label to $v$.

Denote by $\sigma_{vw}$ the partial mapping from $[k]$ to $[k]$ that maps $p$ to $q$ if $\|\tilde{v}_p - \tilde{w}_q\|^2 \leq 4R$. Note that $\sigma_{vw}$ is well defined i.e. $p$ cannot be mapped to different labels $q$ and $q'$: if $\|\tilde{v}_p - \tilde{w}_q\|^2 \leq 4R$ and $\|\tilde{v}_p - \tilde{w}_{q'}\|^2 \leq 4R$, then, by the $\ell_2^2$ triangle inequality (see Lemma 1, item 1), $\|\tilde{w}_q - \tilde{w}_{q'}\|^2 \leq 8R$, but $\tilde{w}_q$ and $\tilde{w}_{q'}$ are orthogonal unit vectors, so

$$\|\tilde{w}_q - \tilde{w}_{q'}\|^2 = 2 > 8R.$$

Clearly, $\sigma_{vw}$ defines a partial matching between labels of $v$ and labels of $w$: if $\sigma_{vw}(p) = q$, then $\sigma_{wv}(q) = p$.

**Lemma 2.** *If $p \in S_v$ and $q \in S_w$ with non-zero probability, then $q = \sigma_{vw}(p)$.*

*Proof.* If $p \in S_v$ and $q \in S_w$ then for some vertex $u$ and label $i$, $\|\tilde{v}_p - \tilde{u}_i\|^2 \leq 2R$ and $\|\tilde{w}_q - \tilde{u}_i\|^2 \leq 2R$, thus by the triangle inequality $\|\tilde{v}_p - \tilde{w}_q\|^2 \leq 4R$ and by the definition of $\sigma_{vw}$, $q = \sigma_{vw}(p)$.

**Corollary 3.** *Suppose, that $p \in S_v$, then the set $S_w$ either equals $\{\sigma_{vw}(p)\}$ or is empty (if $\sigma_{vw}(p)$ is not defined, then $S_w$ is empty). Particularly, if $u$ and $i$ are the initial vertex and label, then the set $S_w$ either equals $\{\sigma_{uw}(i)\}$ or is empty. Thus, every set $S_w$ contains at most one element.*

**Lemma 4.** *For every choice of the initial vertex $u$, for every $v \in V$ and $p \in [k]$ the probability that $p \in S_v$ is at most $\|v_p\|^2$.*

*Proof.* If $p \in S_v$, then $i = \sigma_{vu}(p)$ is the initial label of $u$ and $t \le \|v_p\|^2$. The probability that both these events happen is

$$\Pr\left(i \in S_u\right) \times \Pr\left(t \le \|v_p\|^2\right) = \|u_i\|^2 \times \min(\|v_p\|^2/\|u_i\|^2, 1) \le \|v_p\|^2$$

(recall that $t$ is a random real number on the segment $[0, \|u_i\|^2]$).

Denote the set of those vertices $v$ for which $S_v$ contains exactly one element by $X$. First, we show that on average $X$ contains a constant fraction of all vertices (later we will prove a much stronger bound on the size of $X$).

**Lemma 5.** *If $\varepsilon/\lambda_G \le c_R$, then the expected size of $X$ is at least $|V|/4$.*

*Proof.* Consider an arbitrary vertex $v$. Estimate the probability that $p \in S_v$ given that $u$ is the initial vertex. Suppose that there exists $q$ such that $\|v_p - u_q\|^2 \le \|v_p\|^2 \cdot R/2$, then

$$\|\tilde{u}_q - \tilde{v}_p\|^2 \le \frac{2\|u_q - v_p\|^2}{\max(\|u_q\|^2, \|v_p\|^2)} \le R.$$

Thus, $q = \sigma_{vu}(p)$ and $\|\tilde{u}_q - \tilde{v}_p\|^2 \le r$ with probability 1. Hence, if $q$ is chosen as the initial label and $\|v_p\|^2 \ge t$, then $v_p \in S_v$. The probability of this event is $\|u_q\|^2 \times \min(\|v_p\|^2/\|u_q\|^2, 1)$. Notice that

$$\|u_q\|^2 \times \min(\|v_p\|^2/\|u_q\|^2, 1) = \min(\|v_p\|^2, \|u_q\|^2) \ge \|v_p\|^2 - \|u_q - v_p\|^2 \ge \frac{\|v_p\|^2}{2}.$$

Now, consider all $p$'s for which there exists $q$ such that $\|v_p - u_q\|^2 \le \|v_p\|^2 \cdot R/2$. The probability that one of them belongs to $S_v$, and thus $v \in X$, is at least

$$\begin{aligned}\frac{1}{2} \sum_{p:\min_q(\|u_q - v_p\|^2) \le \|v_p\|^2 \cdot R/2} \|v_p\|^2 &= \frac{1}{2}\sum_{p=1}^{k} \|v_p\|^2 - \frac{1}{2} \sum_{p:\min_q(\|u_q - v_p\|^2) > \|v_p\|^2 \cdot R/2} \|v_p\|^2 \\ &\ge \frac{1}{2} - \frac{1}{2} \times \sum_{p=1}^{k} \frac{2}{R} \min_q(\|u_q - v_p\|^2) \\ &\ge \frac{1}{2} - \frac{\Delta(\{u\}_q, \{v\}_p)}{R}.\end{aligned}$$

Since the average value of $\Delta(\{u\}_q, \{v\}_p)$ over all pairs $(u, v)$ is at most $R/4$ (see (6)), the expected size of $X$ (for a random initial vertex $u$) is at least $|V|/4$.

**Corollary 6.** *If $\varepsilon/\lambda_G \le c_R$, then*

$$\Pr\left(|X| > |V|/8\right) > \frac{1}{8}.$$

**Lemma 7.** *The expected size of the cut between $X$ and $V \setminus X$ is at most $6\varepsilon/R|E|$.*

*Proof.* We show that the size of the cut between $X$ and $V \setminus X$ is at most $6\varepsilon/R|E|$ in the expectation for any choice of the initial vertex $u$. Fix an edge $(v, w)$ and estimate the probability that $v \in X$ and $w \in V \setminus X$. If $v \in X$ and $w \in V \setminus X$, then $S_v$ contains a unique label $p$, but $S_w$ is empty (see Corollary 3) and, particularly, $\pi_{vw}(p) \notin S_w$. This happens in two cases:

- There exists $p$ such that $i = \sigma_{vu}(p)$ is the initial label of $u$ and $\|w_{\pi_{vw}(p)}\|^2 < t \le \|v_p\|^2$. The probability of this event is at most

$$\sum_{p=1}^{k} \|u_{\sigma_{vu}(p)}\|^2 \times \left| \frac{\|v_p\|^2 - \|w_{\pi_{vw}(p)}\|^2}{\|u_{\sigma_{vu}(p)}\|^2} \right| \le \sum_{p=1}^{k} \|v_p - w_{\pi_{vw}(p)}\|^2.$$

- There exists $p$ such that $i = \sigma_{vu}(p)$ is the initial label of $u$, $t \le \|v_p\|^2$ and $\|\tilde{u}_i - \tilde{v}_p\|^2 < r \le \|\tilde{u}_i - \tilde{w}_{\pi_{vw}(p)}\|^2$. The probability of this event is at most

$$\begin{aligned}
&\sum_{p=1}^{k} \|u_{\sigma_{vu}(p)}\|^2 \times \frac{\|v_p\|^2}{\|u_{\sigma_{vu}(p)}\|^2} \times \left| \frac{\|\tilde{u}_{\sigma_{vu}(p)} - \tilde{w}_{\pi_{vw}(p)}\|^2 - \|\tilde{u}_{\sigma_{vu}(p)} - \tilde{v}_p\|^2}{R} \right| \\
&\quad \le \sum_{p=1}^{k} \|v_p\|^2 \times \frac{\|\tilde{v}_p - \tilde{w}_{\pi_{vw}(p)}\|^2}{R} \le \sum_{p=1}^{k} \|v_p\|^2 \times \frac{2\|v_p - w_{\pi_{vw}(p)}\|^2}{R \cdot \max(\|v_p\|^2, \|w_{\pi_{vw}(p)}\|^2)} \\
&\quad \le \frac{2}{R} \sum_{p=1}^{k} \|v_p - w_{\pi_{vw}(p)}\|^2.
\end{aligned}$$

Note that the probability of the first event is zero, if $\|w_{\pi_{vw}(p)}\|^2 \ge \|v_p\|^2$; and the probability of the second event is zero, if $\|\tilde{u}_{\sigma_{vu}(p)} - \tilde{v}_p\|^2 \ge \|\tilde{u}_{\sigma_{vu}(p)} - \tilde{w}_{\pi_{vw}(p)}\|^2$.

Since the SDP value equals

$$\frac{1}{2|E|} \sum_{(v,w)\in E} \sum_{p=1}^{k} \|v_p - w_{\pi_{vw}(p)}\|^2 \le \varepsilon.$$

The expected fraction of cut edges is at most $6\varepsilon/R$.

**Lemma 8.** *If $\varepsilon \le \min(c_R\lambda_G, h_G R/1000)$, then with probability at least 1/16 the size of $X$ is at least*

$$\left(1 - \frac{100\varepsilon}{h_G R}\right)|V|.$$

*Proof.* The expected size of the cut $\delta(X, V \setminus X)$ between $X$ and $V \setminus X$ is less than $6\varepsilon/R|E|$. Hence, since the graph $G$ is an expander, one of the sets $X$ or $V \setminus X$ must be small:

$$\mathbb{E}\left[\min(|X|, |V \setminus X|)\right] \le \frac{1}{h_G} \times \frac{\mathbb{E}\left[|\delta(X, V \setminus X)|\right]}{|E|} \times |V| \le \frac{6\varepsilon}{h_G R}|V|.$$

By Markov's Inequality,

$$\Pr\left(\min(|X|, |V \setminus X|) \le \frac{100\varepsilon}{h_G R}|V|\right) \ge 1 - \frac{1}{16}.$$

Observe, that $100\varepsilon/(h_G R)|V| < |V|/8$. However, by Corollary 6, the size of $X$ is greater than $|V|/8$ with probability greater than $1/8$. Thus

$$\Pr\left(|V \setminus X| \le \frac{100\varepsilon}{h_G R}|V|\right) \ge \frac{1}{16}.$$

**Lemma 9.** *The probability that for an arbitrary edge $(v, w)$, the constraint between $v$ and $w$ is not satisfied, but $v$ and $w$ are in $X$ is at most $4\varepsilon_{vw}$, where*

$$\varepsilon_{vw} = \frac{1}{2}\sum_{i=1}^{k} \|v_i - w_{\pi_{vw}(i)}\|^2.$$

*Proof.* We show that for every choice of the initial vertex $u$ the desired probability is at most $4\varepsilon_{vw}$. Recall, that if $p \in S_v$ and $q \in S_w$, then $q = \sigma_{vw}(p)$. The constraint between $v$ and $w$ is not satisfied if $q \ne \pi_{vw}(p)$. Hence, the probability that the constraint is not satisfied is at most,

$$\sum_{p:\pi_{vw}(p)\ne\sigma_{vw}(p)} \Pr\left(p \in S_v\right).$$

If $\pi_{vw}(p) \ne \sigma_{vw}(p)$, then

$$\|\tilde{v}_p - \tilde{w}_{\pi_{vw}(p)}\|^2 \ge \|\tilde{w}_{\pi_{vw}(p)} - \tilde{w}_{\sigma_{vw}(p)}\|^2 - \|\tilde{v}_p - \tilde{w}_{\sigma_{vw}(p)}\|^2 \ge 2 - 4R \ge 1.$$

Hence, by Lemma 1 (5),

$$\|v_p - w_{\pi_{vw}(p)}\|^2 \ge \|v_p\|^2/2.$$

Therefore, by Lemma 4,

$$\sum_{p:\pi_{vw}(p)\ne\sigma_{vw}(p)} \Pr\left(p \in S_v\right) \le \sum_{p:\pi_{vw}(p)\ne\sigma_{vw}(p)} \|v_p\|^2 \le 2\sum_{p=1}^{k} \|v_p - w_{\pi_{vw}(p)}\|^2 = 4\varepsilon_{vw}.$$

**Theorem 10.** *There exists a polynomial time approximation algorithm that given a $(1-\varepsilon)$ satisfiable instance of Unique Games on a $d$-expander graph $G$ with $\varepsilon/\lambda_G \le c$, the algorithm finds a solution of value*

$$1 - C\frac{\varepsilon}{h_G},$$

*where $c$ and $C$ are some positive absolute constants.*

*Proof.* We describe a randomized polynomial time algorithm. Our algorithm may return a solution to the SDP or output a special value *fail*. We show that the algorithm outputs a solution with a constant probability (that is, the probability of failure is bounded away from 1); and conditional on the event that the algorithm outputs a solution its expected value is

$$1 - C\frac{\varepsilon}{h_G}. \tag{7}$$

Then we argue that the algorithm can be easily derandomized — simply by enumerating all possible values of the random variables used in the algorithm and picking the best solution. Hence, the deterministic algorithm finds a solution of value at least (7).

The randomized algorithm first solves the SDP and then runs the rounding procedure described above. If the size of the set $X$ is more than

$$\left(1 - \frac{100\varepsilon}{h_G R}\right)|V|,$$

the algorithm outputs the obtained solution; otherwise, it outputs *fail*.

Let us analyze the algorithm. By Lemma 8, it succeeds with probability at least $1/16$. The fraction of edges having at least one endpoint in $V \setminus X$ is at most $100\varepsilon/(h_G R)$ (since the graph is $d$-regular). We conservatively assume that the constraints corresponding to these edges are violated. The expected number of violated constraints between vertices in $X$, by Lemma 9 is at most

$$\frac{4\sum_{(u,v)\in E}\varepsilon_{uv}}{\Pr\left(|X| \geq 100\varepsilon/(h_G R)\right)} \leq 64 \times \left(\frac{1}{2}\sum_{(u,v)\in E}\|u_i - v_{\pi_{vw}(i)}\|^2\right) \leq 64\varepsilon|E|.$$

The total fraction of violated constraints is at most $100\varepsilon/(h_G R) + 64\varepsilon$.

## References


1. Arora, S., Khot, S., Kolla, A., Steurer, D., Tulsiani, M., Vishnoi, N.: Unique games on expanding constraint graphs are easy. In: Proceedings of the 40th ACM Symposium on Theory of Computing, pp. 21–28 (2008)
2. Austrin, P., Mossel, E.: Approximation resistant predicates from pairwise independence. In: Proceedings of the 2008 IEEE 23rd Annual Conference on Computational Complexity, pp. 249–258. IEEE Computer Society, Los Alamitos (2008)
3. Charikar, M., Makarychev, K., Makarychev, Y.: Near-Optimal Algorithms for Unique Games. In: Proceedings of the 38th ACM Symposium on Theory of Computing, pp. 205–214 (2006)
4. Chlamtac, E., Makarychev, K., Makarychev, Y.: How to Play Unique Games Using Embeddings. In: Proceedings of the 47th IEEE Symposium on Foundations of Computer Science, pp. 687–696 (2006)
5. Gupta, A., Talwar, K.: Approximating Unique Games. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms, pp. 99–106 (2006)
6. Guruswami, V., Manokaran, R., Raghavendra, P.: Beating the Random Ordering is Hard: Inapproximability of Maximum Acyclic Subgraph. In: Proceedings of the 49th IEEE Symposium on Foundations of Computer Science, pp. 573–582 (2008)
7. Guruswami, V., Raghavendra, P.: Constraint satisfaction over a non-boolean domain: Approximation algorithms and unique-games hardness. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 77–90. Springer, Heidelberg (2008)

8. Khot, S.: On the power of unique 2-prover 1-round games. In: Proceedings of the 34th ACM Symposium on Theory of Computing, pp. 767–775 (2002)
9. Khot, S., Kindler, G., Mossel, E., O'Donnell, R.: Optimal inapproximability results for MAX-CUT and other two-variable CSPs? SIAM Journal of Computing 37(1), 319–357 (2007)
10. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within $2-\varepsilon$. In: Proceedings of the 18th IEEE Annual Conference on Computational Complexity (2003)
11. Samorodnitsky, A., Trevisan, L.: Gowers uniformity, influence of variables, and PCPs. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 11–20 (2006)
12. Trevisan, L.: Approximation Algorithms for Unique Games. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science, pp. 197–205 (2005)

# Online Ranking for Tournament Graphs*

Claire Mathieu and Adrian Vladu

Brown University, Department of Computer Science, Providence RI, USA
{claire,avladu}@cs.brown.edu

**Abstract.** We study the problem of producing a global ranking of items given pairwise ranking information, when the items to be ranked arrive in an online fashion. We study both the maximization and the minimization versions of the problem on tournaments (max acyclic subgraph, feedback arc set). We also study the case when the items arrive in random order.

## 1 Introduction

*Context.* Given complete pairwise ranking information between data items, of the form "*player i beats player j*", one seeks to provide a global ranking of the players (or items) that aims to be consistent with the pairwise information, as far as possible. Motivated by scheduling, graph layout, and rank aggregation, this NP-hard problem was extensively studied [20,23,21,10,3,15].

Here, we study the online version on tournaments. How should we insert a newly arrived player without upsetting the current ranking? A new player arrives, along with pairwise information about which players he beats, and the algorithm must extend the current ranking by incorporating the new player. Over time, the algorithm should hedge against the risk that the ranking may gradually drift from the optimal.

In the maximum acyclic subgraph problem, the objective is to maximize consistency, i.e. the number of pairs $uv$ whose ordering in the output ranking agrees with the input information. This measure can be too coarse in cases where the input is almost perfect, since getting 99% of the pairs ordered consistently with the input is not really a good outcome when there exists a perfect ordering (or, say, an ordering with one single upset pair); in the more difficult feedback arc set problem, the objective is to minimize inconsistencies, i.e. the number of pairs $uv$ whose ordering in the output ranking disagree with the input information.

For the feedback arc set problem on tournaments, we show that there is a wide gap between the offline and the online performance of algorithms. Indeed, the offline problem has an approximation scheme [19], yet we prove that no online algorithm, even with randomization, can be better than the greedy algorithm, which we prove is $(n-2)$-competitive. For the easier maximum acyclic subgraph problem on tournament, the gap is smaller. Still, the offline problem has an approximation scheme [4,16,19], yet we prove that no online algorithm, even with randomization, can be $1-\epsilon$ competitive; the greedy algorithm is 1/2-competitive.

* This work was partially funded by NSF grant CCF-0728816.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 201–212, 2011.

For both problems the worst case can only happen if the adversary controls both the input graph and the order of arrival of the graph vertices. The situation is very different when the input graph is arbitrary but the arrivals are a random permutation of the vertices. This online computation model with random order has been the focus of increased attention in recent years. It is particularly relevant to situations where data items arrive from different, independent sources. Although the model was already suggested in the 1990s in the context of best fit bin packing [17], it is increasingly the focus of active research ([5,13] for example).

This paper presents an instance where the random order model is much more powerful than the standard online model: it almost enables a reduction to the offline model! More precisely, for online feedback arc set with random order, we observe the existence of a 3-competitive algorithm, and for online maximum acyclic subgraph with random order, we observe the existence of an asymptotic approximation scheme. Moreover, the results follow easily from prior work on the offline model. This is a striking example where random order resolves most of the difficulties inherent to online computation.

*Definitions and results.* At each time $t$, a new item $v$ arrives, along with a relative ranking with respect to each previously arrived item $u$ (pairwise comparisons). Thus the input after $t$ steps is a tournament over $t$ data items – a directed graph such that for every pair $\{u, v\}$, exactly one of the two arcs $(u, v)$ and $(v, u)$ is in the edge set. Here $(u, v) \in E$ means that $u$ is ranked higher than $v$. The algorithm maintains a total ordering of the items: a newly arrived item $v$ must be inserted in the existing ranking. The final ranking is evaluated as follows: in the maximum acyclic subgraph problem, the value of the output is the number of pairs $uv$ whose ordering in the output ranking agrees with the input information; in the feedback arc set problem, the cost of the output is the number of pairs $uv$ whose ordering in the output ranking disagree with the input information.

*Techniques.* The most interesting proofs are the analysis of the greedy algorithm for minimum feedback arc set and the randomized lower bound for maximum acyclic subgraph. Every time a vertex arrives, Greedy adds it to the current ranking at a position that minimizes the number of induced inconsistent pairs, breaking ties in favor of the position of lowest index.

To analyze Greedy, we argue that if the greedy permutation and the offline optimum are very different, then there must be some combinatorial structures that we call $c$-entanglements (see Figure 1); in turn, $c$-entanglement implies a lower bound on the cost of the optimal ranking. To prove a lower bound on the randomized complexity of maximum acyclic subgraph, we provide a distribution supported by two inputs and show that any algorithm that works well on the first input must be far from optimal on the second input; that is done by a delicate modification of the algorithm's output, that can be analyzed in terms of $L_1$ distance; fortunately it is well-known that the inversion and the $L_1$ distances between permutations are within a factor of 2 of each other (Theorem 3), an essential tool in our proof.

**Theorem 1.** *Consider online feedback arc set on tournaments.*

1. *The greedy algorithm has competitive ratio* $n-2$.
2. *Every (deterministic or randomized) algorithm has competitive ratio at least* $n-2$.
3. *If vertices arrive in random order then there is a* 3*-competitive algorithm.*
4. *Even if vertices arrive in random order, every (deterministic or randomized) algorithm has competitive ratio at least* 1.25.

**Theorem 2.** *Consider online maximum acyclic subgraph on tournaments.*

1. (Folklore) *The greedy algorithm has competitive ratio* $(1/2)$*; this is tight.*
2. *Every deterministic (resp. randomized) algorithm has competitive ratio at least* $(1/2)$ *(resp.* $(1-1/77)$*).*
3. *When vertices arrive in random order, there is a* $(1-\epsilon)$ *asymptotic approximation.*

Theorem 1 is proved in section 2 (with subsections 2.1, 2.3, 2.3 and 2.4 respectively proving parts 1,2,3 and 4 of the Theorem). Theorem 2 is proved in section 3 (with subsections 3.1,3.2 and 3.3 respectively proving parts 1, 2 and 3 of the Theorem).

*Background results.* The following results are known for the offline problems. The maximum acyclic subgraph problem on tournaments and the feedback arc set problem on tournaments are NP-hard [2,1,8,11] . The "Quicksort" ranking algorithm is a randomized 3-approximation for the feedback arc set on tournaments [2]. Moreover, there is a polynomial-time approximation scheme (PTAS) for the feedback arc set on tournaments [19]. There is a PTAS for the maximum acyclic subgraph on tournaments [4,16,19]. Moreover the seeded randomized greedy algorithm is a PTAS for the problem [19].

## 2 Online Feedback Arc Set on Tournaments

### 2.1 Analysis of Greedy

Throughout this section, let $\pi$ denote the ranking obtained by Greedy. Let $G(V,E)$ be a tournament graph and let $k$ denote the $k$th arriving vertex in the online order ($k \leq n$). Greedy maintains a ranking $\pi$ of vertices where $\pi(u)$ denotes the position of vertex $u$ in the ranking at the current time. Let $\sigma$ denote the optimal ranking. A *back edge* of a ranking $\rho$ is a pair of vertices $\{u,v\}$ such that $(u,v) \in E$ but $\rho(v) < \rho(u)$. Let $B_\rho(v)$ denote the number of back edges of $\rho$ induced by the arrival of a vertex $v$. Theorem 1 can be proved from the following two propositions.

For two rankings $\rho$ and $\tau$, let $\mathcal{K}_v(\rho,\tau)$ denote the number of inversions between $\tau$ and $\rho$ induced by the arrival of $v$, *i.e.* the number of vertices $u$ arrived before $v$ and such that $\{u,v\}$ is ordered differently in $\rho$ and in $\tau$. Let $\rho[v \to p]$ denote the ranking of vertices obtained from a ranking $\rho$ by removing $v$ and putting it back in so that its resulting rank is $p+1$.

**Definition 1.** *Given two rankings $\rho$ and $\tau$, two sets of vertices $A$ and $B$ of size $c$ are $c$-entangled if:*

- $\max_{a\in A} \rho(a) < \min_{b\in B} \rho(b)$.
- *For every $i < c$ there is a subset $A'$ of $A$ of size at least $c-i+1$ and a subset $B'$ of $B$ of size at least $i$ such that* $\max_{b\in B'} \tau(b) < \min_{a\in A'} \tau(a)$.

**Proposition 1.** *Given two rankings $\rho$ and $\tau$, let $c = \min_p \mathcal{K}_k(\rho, \tau[k \to p])$. Then there are two subsets $A$ and $B$ of $\{1, 2, \dots k-1\}$ that are $c$-entangled.*

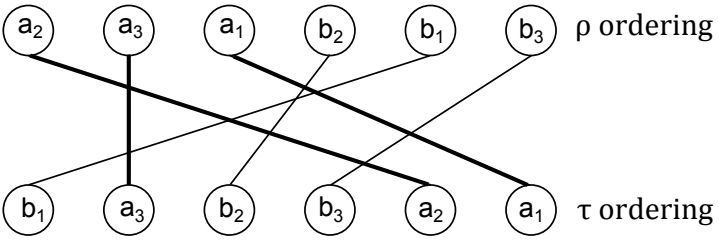



**Fig. 1.** Two sets of vertices that are 3-entangled. The upper row represents the ranking in $\rho$, the lower row gives the ranking in $\tau$.

**Proposition 2.** *Let $G$ be an arbitrary tournament, with arbitrary arrival order, $\pi$ be the greedy ranking, and $\rho$ be an arbitrary ranking. If there are two sets of vertices $A$ and $B$ that are $c$-entangled with respect to $\rho$ and $\pi$, then $\rho$ has cost at least $c$.*

*Proof.* (of part 1 of Theorem 1). The cost of Greedy is $\sum_{k=3}^{n} B_\pi(k)$. By definition of Greedy,

$$B_\pi(k) = \min_p B_{\pi[k\to p]}(k).$$

Since a back edge in $\pi[k \to p]$ is either a back edge in $\sigma$ or a pair ordered differently in $\sigma$ and in $\pi[k \to p]$, we have: $B_{\pi[k\to p]}(k) \le B_\sigma(k) + \mathcal{K}_k(\sigma, \pi[k \to p])$. Combining, we obtain:

$$B_\pi(k) \le B_\sigma(k) + \min_p \mathcal{K}_k(\sigma, \pi[k \to p]).$$

Let $c = \min_p \mathcal{K}_k(\sigma, \pi[k \to p])$. From Proposition 1, there are two subsets $A$ and $B$ of $[1, k-1]$ that are $c$-entangled with respect to $\sigma$ and $\pi$. From Proposition 2, the restriction of $\sigma$ to $[1, k-1]$ has cost at least $c$. In other words, $c \le \sum_{i=1}^{k-1} B_\sigma(i)$. Summing over $k$ and inverting summations concludes the proof.

*Proof.* (of Proposition 1.) Among vertices $\{1, 2, \dots k-1\}$, let $L = \{i : \rho(i) < \rho(k)\}$ and $R = \{i : \rho(i) > \rho(k)\}$. First we claim that $L$ and $R$ both have cardinality at least $c$. Indeed, inserting $k$ in first position in $\tau$ yields $c \le \mathcal{K}_k(\rho, \tau[k \to 0]) = |L|$, and similarly inserting $k$ in the last position yields $c \le |R|$.

Now, let $A$ denote the $c$ vertices $i$ of $L$ such that $\tau(i)$ is maximum, and $B$ denote the $c$ vertices $i$ of $R$ such that $\tau(i)$ is minimum. This defines $A$ and $B$.

Clearly $\max_{a\in A}\rho(a) < \min_{b\in B}\rho(b)$. Now, fix $i < c$ and consider the element $a^*$ of $A$ such that $\tau(a^*)$ is the $i$th largest among elements of $A$. In $\tau^{(k-1)}$, consider inserting $k$ immediately to the right of $a^*$. Then there are only $c-i$ vertices $v$ of $L$ such that $\{k, v\}$ is an inversion, yet for that value of $p$, by definition of $c$ we know that $\mathcal{K}_k(\rho, \tau[k \to p]) \geq c$, so there must be at least $i$ vertices $v$ of $R$ such that $\{k, v\}$ is an inversion, hence such that $\tau(v) < \tau(a^*)$. Therefore there must be at least $i$ vertices $v$ of $B$ such that $\tau(v) < \tau(a^*)$, proving the lemma.

*Proof.* (of Proposition 2.) Among all instances of $(G, \rho)$ such that $\rho$ has minimum cost, we claim that we can choose one such that the following property (P) holds: *let* $L = \{u : \rho(u) \leq \max_{a\in A}\rho(a)\}$ *and* $R = V \setminus L$. *Then* $\pi|L = \rho|L$ *and both have cost 0; similarly,* $\pi|R = \rho|R$ *and both have cost 0.*

The proof is by contradiction. Among all instances such that the cost of $\rho$ is minimum, pick the one such that the inversion distance between $\rho|L$ and $\pi|L$ is minimum. We claim that it is 0: assuming $\rho|L \neq \pi|L$, pick $u, v \in L$ such that $\rho(u) = i$, $\rho(v) = i+1$, and $\pi(u) > \pi(v)$. Then let $\rho'$ be equal to $\rho$ except for transposing $u$ and $v$, and let $G'$ be equal to $G$ with the possible exception of pair $\{u, v\}$: in $G'$, $(v, u) \in G'$. By definition, $\rho'$ is closer to $\pi$ than $\rho$ in inversion distance. Our definition of $G'$ ensures that the cost of $\rho'$ is at most the cost of $\rho$. Moreover, it is easy to see that $\pi(G') = \pi(G)$. Therefore this provides a new min cost instance with smaller inversion distance, a contradiction.

Similarly we can argue that $\rho|R = \pi|R$.

Finally, among all instances such that the cost of $\rho$ is minimum, $\rho|L = \pi|L$ and $\rho|R = \pi|R$, we choose one such that the cost of $\pi$ is minimum. We claim that $\pi|L$ and $\pi_R$ have cost 0: if not, modify $G$ into $G'$ by inverting a back edge $(u, v)$, and argue as above that $\pi(G') = \pi(G)$, hence a contradiction.

From now on we assume that Property (P) holds. Our next step is to find a lower bound for $cost(\rho)$. Define a *right-left matching* to be a matching between pairs $(u, v) \in R \times L$ such that $\pi(u) < \pi(v)$.

Consider the right-left matching $\nu = \bigcup_{i=1}^{m}(r_i, l_i)$ given by the following greedy algorithm. Go through the vertices $r$ from $R$ in increasing order of $\pi(r)$. For each such $r$ find the vertex $l = \arg\min_{\{l\in L:\pi(r)<\pi(l), l \text{ unmatched}\}}\pi(l)$. If such a vertex exists, match it with $r$. Matching $\nu$ is maximum. Indeed, consider a maximum right-left matching $\nu' = \bigcup_{i=1}^{m'}(r'_i, l'_i)$. First, in $\nu'$ we exchange the vertices $r'_i$, in increasing order of $\pi(r'_i)$, with those vertices from $R$ that have the lowest positions in $\pi$: this does not affect feasibility since each vertex $r'_i$ gets replaced by a vertex $r_i$" such that $\pi(r_i") \leq \pi(r'_i)$. Then we exchange each vertex $l'_i$, in increasing order of $\pi(r'_i)$, with the one that has the smallest position in $\pi|L$ such that feasibility is maintained; one can prove that the maximum matching obtained is exactly $\nu$.

We claim that $cost(\rho) \geq |\nu|$ and $|\nu| \geq c$, from which the proposition follows.

To prove the first claim, we argue that if the arrival of a vertex $v$ causes the size of the maximum matching to increase (necessarily by 1), then we must have $B_\rho(v) \geq 1$.

Suppose, for a contradiction, that when inserting some vertex $t$ the size of the maximum right-left matching increases while $B_\rho(t) = 0$. Assume that $t \in L$

(the case $t \in R$ is similar). Since $B_\rho(t) = 0$, we must have $(t, r) \in E$ for every $r \in R$. By definition of Greedy, for every position $z < \pi(t)$ we have $B_{\pi[t \to \pi(t)]}(t) \leq B_{\pi[t \to z]}(t)$. Since $(t, r) \in E$ for every $r$, this implies that $|\{r \in R : r < t \text{ and } z \leq \pi(r) < \pi(t)\}| \leq |\{l \in L : l < t \text{ and } z \leq \pi(l) < \pi(t)\}|$. It is not hard to see that this implies that all vertices $r \in R$ such that $\pi(r) < \pi(t)$ can be matched to a vertex $\ell \in L$ that also has $\pi(\ell) < \pi(t)$. So all vertices of $R$ such that $\pi(r) < \pi(t)$ are matched by the greedy algorithm $\mathcal{M}$. So $\mathcal{M}$ will not match $t$ to anything because all its potential pairs are already matched, so the size of the maximum matching given by $\mathcal{M}$ does not increase: contradiction.

To prove the second claim, take the $c$-entangled sets $A$ and $B$. By definition of $c$-entanglement (first property) and by definition of $L$ and $R$, it follows that $A \subseteq L$ and $B \subseteq R$. By definition of $c$-entanglement (second property), we can get a partition of $A \cup B$ into $c$ disjoint pairs $(b_i, a_i)$ such that $\pi(b_i) < \pi(a_i)$ , $b_i \in B, a_i \in A$. These pairs form a right-left matching of size $c$. So the maximum right-left matching $\nu$ has size $|\nu| \geq c$.

### 2.2 Deterministic and Randomized Lower Bounds

To prove the deterministic lower bound, consider the following two inputs. $I_1$ has $n$ vertices, labeled $1, 2, \ldots, n$ in order of the optimal ranking, and the only back edge of the optimal ranking is edge $(n, 1)$. The optimal cost is 1. The arrival order is $n, 1, 2, \ldots, n-1$. Input $I_2$ has two vertices, labeled $1, 2$ in order of the optimal ranking, and there are no back edges. The optimal cost is 0 and the arrival order is 1,2. In order to be competitive for $I_2$, the algorithm must place the first two vertices so that the cost is 0. Then, for $I_1$, any extension of that ranking has at least one back edge from each of the other $n-2$ vertices, hence the lower bound.

To prove the randomized lower bound, we use Yao's minmax theorem [7,22], and consider the input distribution that is $I_1$ and $I_2$ with equal probability. Input $I_1$ has $n$ vertices labeled $1, 2, \ldots, n$ in order of the optimal ranking, and the only back edge of the optimal ranking is edge $(n, 1)$. The optimal cost is 1. The arrival order is $n, 1, 2, \ldots, n-1$. Input $I_2$ has $n$ vertices labeled $1, 2, \ldots, n$ in order of the optimal ranking, and there are no back edges in the optimal ranking. The optimal cost is 0. The arrival order is $1, n, 2, \ldots, n-1$. The average cost of the optimal output is $1/2$. Let $\mathcal{A}$ be any deterministic algorithm. We will prove that the average cost of the output is at least $(n-2)/2$.

First, consider the case when $\mathcal{A}$ places the first edge forward. With probability $1/2$ the input is $I_1$ and then the output ranking has cost at least $n-2$; with the remaining probability $31/2$, the input is $I_2$ and then the output ranking has cost 0. The average cost of the output is at least $(n-2)/2$. The analysis in the other case is similar (and yields $(n-1)/2 > (n-2)/2$).

### 2.3 Random Order Arrivals: A Better Algorithm

Here is an online algorithm. Upon arrival of vertex $t$, we place it in the current ranking $\rho$ as follows. Let $S_1$ the set of all vertices in $\rho$ and $i_1$ the vertex from

**Algorithm 1.** Insert a new vertex into the current ranking

**Input:** a newly arrived vertex $t$, the current set of vertices $S_1$, the current ranking $\rho$, a set $A$ consisting of all the edges between $t$ and the vertices in $\rho$
**Output:** the updated ranking $\rho'$

```
p ← 1, lo ← 0, hi ← |S_1|
while S_p ≠ ∅ do
  Let i_p the vertex in S_p that arrived first
  if (t, i_p) ∈ A then
    Let S_{p+1} = {v ∈ S_p : (v, i_p) ∈ A}, hi ← ρ(i_p) − 1
  else
    Let S_{p+1} = {v ∈ S_p : (i_p, v) ∈ A}, lo ← ρ(i_p)
  end if
  p ← p + 1
end while
ρ' ← ρ[t → lo]
```

$S_1$ that arrived first. $t$ is before vertex $i_1$ if there is an edge from $t$ to $i_1$, and is after vertex $i_1$ if there is an edge from $i_1$ to $t$. Let $S_2$ the set of vertices in $S_1$ that are in $\rho$ on the same side of $i_1$ where we place $t$. We continue in the same manner as before until $S_p = \emptyset$ and the position of $t$ is entirely determined. So we place $t$ there. This procedure is presented in Algorithm 1.

Since vertices arrive in random order, vertex $i_1$ is like the first pivot used by the Quicksort ranking algorithm of Ailon, Charikar and Newman ([2]), and in fact the above algorithm is an equivalent description. Hence the 3-approximation result carries over to yield a proof that the algorithm is 3-competitive.

### 2.4 Random Order Lower Bounds

Consider the following input. There are 4 vertices, labeled 1,2,3,4 in order of the optimal ranking. The only back edge of the optimal ranking is edge $(4, 1)$. The optimal cost is 1. The arrival order is random. Let $\mathcal{A}$ be any deterministic algorithm. We will prove that the average cost of the output is at least $5/4$.

First, consider the case when $\mathcal{A}$ places the first edge forward. With probability $\binom{4}{2}/4! = 1/4$ the first two arriving vertices are 1 and 4 and then the output ranking has cost at least 2; with the remaining probability $3/4$, the output ranking has cost at least 1. The average cost of the output is at least $5/4$. The analysis in the other case is similar (and yields $7/4 > 5/4$).

## 3 Maximum Acyclic Subgraph

### 3.1 Analysis of Greedy

The upper bound is folklore. To prove tightness, we exhibit an input $I$ on which Greedy's performance is asymptotically $OPT/2$ . Consider the following input.

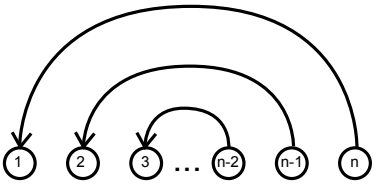



**Fig. 2.** Input showing that the performance of the greedy algorithm is asymptotically $OPT/2$. Only back edges are depicted.

There are $n$ vertices labeled $1, 2, \ldots, n$ in order of the optimal ranking. Here we assume $n$ is even. The back edges of the optimal ranking are edges $(n-i+1, i)$, $1 \leq i \leq n/2$. (It's easy to verify that this ranking is optimal, since all the back edges belong to edge disjoint triangles.) The optimal profit is $\binom{n}{2} - n/2$, which is asymptotic to $n^2/2$. The arrival order is $1, n, 2, n-1$, etc. Then it is easy to see (using the tie-breaking rule) that Greedy produces the ranking $\ldots (n-2), 3, (n-1), 2, n, 1$, with total profit asymptotically $n^2/4$.

### 3.2 Deterministic and Randomized Lower Bounds

To prove the deterministic lower bound, consider the family of inputs defined as follows. Here, we label vertices by order of arrival. First, $(1, 2) \in E$. The rest of the input depends on the algorithm. If the algorithm places 1 before 2, then every future arrival $u$ has an edge $(2, u)$ and an edge $(u, 1)$, else every future arrival $u$ has an edge $(1, u)$ and an edge $(u, 2)$. Then, $(3, 4) \in E$. If the algorithm places 3 before 4, then every future arrival $u$ has an edge $(4, u)$ and an edge $(u, 3)$, else every future arrival $u$ has an edge $(3, u)$ and an edge $(u, 4)$. This guarantees that the output has profit at most $(n/2) + (\binom{n}{2} - (n/2))/2$, which is asymptotically $n^2/4$. On the other hand, there is a ranking with profit at least $\binom{n}{2} - (n/2)$ which is asymptotically equivalent to $n^2/2$, hence the lower bound.

Since the input is adaptive, this does not extend to the randomized setting. To prove the randomized lower bound, we use Yao's minmax theorem [7,22], and consider the input distribution that is $I_1$ with probability $p$ and $I_2$ with probability $1-p$ (in the end we will set $p = 0.967418$.) Input $I_1$ consists of $n$ *red* vertices $R$ whose optimal ranking $r_1 r_2 \ldots r_n$ has no back edges. Its optimal

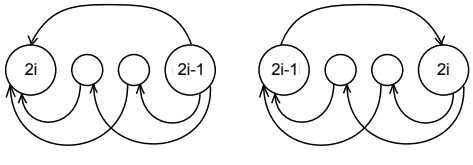



**Fig. 3.** As soon as the algorithm determines whether to place vertex $2i$ before or after $2i-1$, the adversary makes all the subsequent vertices have outgoing edges to the vertex with the lowest rank among $2i-1$ and $2i$ and incoming edges from the other one

profit is $\binom{n}{2}$. Input $I_2$ consisting of $I_1$, followed (in arrival order) by $g(n+1)$ *blue* vertices $B = \{b_{ij} : 1 \le i \le n+1, 1 \le j \le g\}$ (in the end we will set $g = 8$). The ranking

$$b_{11} \dots b_{1g} r_n b_{21} \dots b_{2g} r_{n-1} \dots r_2 b_{n1} \dots b_{ng} r_1 b_{n+1,1} \dots b_{n+1,g}$$

has back edges exactly for vertex pairs in $R \times R$. Let $m = n + g(n+1)$ denote the total number of vertices in $I_2$. The optimal profit of $I_2$ is at least $\binom{m}{2} - \binom{n}{2}$. The average optimal profit is at least $p\binom{n}{2} + (1-p)(\binom{m}{2} - \binom{n}{2})$. Let $\mathcal{A}$ be any deterministic algorithm. We will analyze two cases.

First, consider the case when, in input $I_1$, $\mathcal{A}$ has profit at most $c_1\binom{n}{2}$ (in the end we will set $c_1 = 0.897637$.) The algorithm trivially has profit at most at most $\binom{m}{2}$ on input $I_2$. A short computation shows that

$$\frac{E_I[profit(\mathcal{A}(I))]}{E_I[profit(OPT(I))]} \le 1 - \frac{(1-c_1)\frac{p}{1-p} - 1}{\frac{p}{1-p} + \frac{\binom{m}{2}}{\binom{n}{2}} - 1} \tag{1}$$

Second, consider the case when, in input $I_1$, $\mathcal{A}$ has profit greater than $c_1\binom{n}{2}$. The analysis in that case rests on the following lemma, where $\mathcal{K}(\sigma, \rho)$ is the Kendall-Tau distance (or inversion distance) between permutations, i.e. the number of pairs ordered differently in $\sigma$ and in $\rho$.

**Lemma 1.** *Let $\pi_2$ be a maximum profit ranking of $I_2$ extending $\mathcal{A}(I_1)$ to an .Let $\alpha_2$ be the ranking $b_{11} \dots b_{1g} r_n b_{21} \dots b_{2g} r_{n-1} \dots r_2 b_{n1} \dots b_{ng} r_1 b_{n+1,1} \dots b_{n+1,g}$ of $I_2$ (Figure 3.2). Then:*

$$\mathcal{K}(\pi_2, \alpha_2) \ge \frac{1}{2}(g+1)c_1\binom{n}{2}.$$

Let us defer its proof for a moment. On input $I_1$, $\mathcal{A}(I_1) = \pi_1$ has profit at most $\binom{n}{2}$. On input $I_2$, the profit of $\mathcal{A}(I_2)$ is at most the profit of $\pi_2$. Every inversion between two vertices in $\pi_2$ and $\alpha_2$ that are not both red determines a back edge in $\pi_2$. Therefore, counting out the possible back edges induced by pairs of red vertices, the number of back edges in $\pi_2$ is at most $\mathcal{K}(\pi_2, \alpha_2) - \binom{n}{2}$. So

$$profit(\pi_2) \le \binom{m}{2} - \mathcal{K}(\pi_2, \alpha_2) + \binom{n}{2}.$$

Using Lemma 1 and combining, a short calculation yields

$$\frac{E_I[profit(\mathcal{A}(I))]}{E_I[profit(OPT(I))]} \le 1 - \frac{\frac{c_1(g+1)}{2} - 2}{\frac{p}{1-p} + \frac{\binom{m}{2}}{\binom{n}{2}} - 1} \tag{2}$$

Finally, we numerically find the values for $p, g$ and $c_1$ that minimize the maximum of (1) and (2). For $p = 0.967418, g = 8, c_1 = 0.897637$, both of these are less than $0.986822 \approx 1 - 1/77$.

To prove Lemma 1, it is useful to relate two distances between permutations.

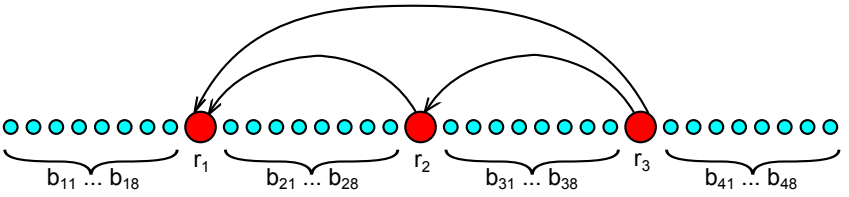



**Fig. 4.** Input $I_2$ for $n = 3$

**Theorem 3.** *([14]) For any two permutations $\sigma$ and $\rho$, we have $\mathcal{K}(\sigma, \rho) \leq L_1(\sigma, \rho) \leq 2\mathcal{K}(\sigma, \rho)$, where $L_1(\sigma, \rho) = \sum_i |\sigma(i) - \rho(i)|$.*

*Proof.* (of Lemma 1) By Theorem 3, $\mathcal{K}(\pi_2, \alpha_2) \geq \frac{1}{2} L_1(\pi_2, \alpha_2)$. Let $\pi'_2$ be the ranking obtained from $\alpha_2$ by reordering the vertices of $R$ according to $\pi_1$ while still leaving them in positions $i(g+1)$ for $1 \leq i \leq n$: thus $\pi'_2(r) = \pi_1(r)(g+1)$ for all $r \in R$ and all the edges in $B \times B$ are forward edges. Since $\pi'_2(b) = \alpha_2(b)$ for every $b \in B$, we can write

$$L_1(\pi_2, \alpha_2) = \sum_{j \in B} |\pi_2(j) - \pi'_2(j)| + \sum_{i \in R} |\pi_2(i) - \alpha_2(i)|.$$

First we need the following relation:

*Claim*
$$\sum_{j \in B} |\pi_2(j) - \pi'_2(j)| \geq \sum_{i \in R} |\pi_2(i) - \pi'_2(i)|$$

*Proof.* In order to minimize the left hand side, we can see that the blue vertices have to appear in the same order in $\pi_2$ as in $\pi'_2$. Indeed, if $u, v \in B$ such that $\pi'_2(u) < \pi'_2(v)$ and $\pi_2(v) < \pi_2(u)$, then swapping the positions of $u$ and $v$ in $\pi_2$ decreases $|\pi_2(u) - \pi'_2(u)| + |\pi_2(v) - \pi'_2(v)|$. For this particular ranking of vertices in $\pi_2$, we can show by a simple calculation that $\sum_{j \in B} |\pi_2(j) - \pi'_2(j)| = \sum_{i \in R} |\pi_2(i) - \pi'_2(i)|$. Therefore the claim holds.

Thus
$$L_1(\pi_2, \alpha_2) \geq \sum_{i \in R} |\pi_2(i) - \pi'_2(i)| + |\pi_2(i) - \alpha_2(i)|.$$

By the triangular inequality, this implies $L_1(\pi_2, \alpha_2) \geq \sum_{i \in R} |\pi'_2(i) - \alpha_2(i)|$. Since $\pi'_2(b) = \alpha_2(b)$ for all $b \in B$, we deduce

$$L_1(\pi_2, \alpha_2) \geq L_1(\pi'_2, \alpha_2).$$

Now, let $\alpha_1 = \alpha_2 | R$. By definition of $\pi'_2$ and of $\alpha_2$, we see that $L_1(\pi'_2, \alpha_2) = (g+1) L_1(\pi_1, \alpha_1)$. From Theorem 3, $L_1(\pi_1, \alpha_1) \geq \mathcal{K}(\pi_1, \alpha_1)$. From our assumption, at least $c_1 \binom{n}{2}$ edges given by the ranking $\pi_1$ are forward edges, and so, $\mathcal{K}(\pi_1, \alpha_1) \geq c_1 \binom{n}{2}$. This concludes the proof.

### 3.3 Random Order Arrivals

Since the optimal ranking has value at least $\binom{n}{2}/2$, it is enough to provide an online algorithm with additive error $O(\epsilon n^2)$. First, observe that it is enough to provide an approximate ranking, identifying all ranks in $[i\epsilon, (i+1)\epsilon)$ – up to an additive error of $O(\epsilon n^2)$, thus we only have $k = 1/\epsilon$ essentially different labels.

Here is the algorithm. We place the first $s = O(1/\epsilon^4)$ vertices arbitrarily, producing a partial ranking $\pi$. At that point, we have a random sample $S$ of the entire set of vertices. The problem can be has one ranking constraint for each pair of vertices $i, j$. The offline algorithm from [19] constructs a ranking of $S$, then proceeds greedily to place the remaining vertices: we execute that part of the algorithm and construct a virtual ranking $\sigma$ of $S$, unrelated to the ranking constructed so far. As in [19], we then insert the remaining vertices in a greedy manner, pretending the original ranking was $\sigma$. As the vertices arrive in random order, the analysis from [19] applies and the result, had we started with the virtual ranking, would be a ranking with additive error $O(\epsilon n^2)$. The fact that $S$ is really ranked according to $\pi$ instead of $\sigma$ induces an additional error of $O(s^2 + sn)$, which is $O(\epsilon n^2)$ assuming that $n = \Omega(1/\epsilon^5)$.

## References


1. Alon, N.: Ranking tournaments. SIAM J. Discrete Math. 20(1), 137–142 (2006)
2. Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: ranking and clustering. Journal of the ACM (JACM) 55, 1–27 (2008)
3. Ailon, N., Mohri, M.: An efficient reduction of ranking to classification. In: Procs. 21st COLT, pp. 87–97 (2008)
4. Arora, S., Frieze, A., Kaplan, H.: A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. Mathematical Programming 92, 1–36 (2002)
5. Babaioff, M., Immorlica, N., Kleinberg, R.: Matroids, secretary problems, and online mechanisms. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 434–443 (2007)
6. Berger, B., Shor, P.: Tight bounds for the maximum acyclic subgraph problem. Journal of Algorithms 25, 118 (1997)
7. Borodin, A., El-Yaniv, R.: On-Line Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
8. Charbit, P., Thomasse, S., Yeo, A.: The minimum feedback arc set problem is NP-hard for tournaments. Combinatorics, Probability and Computing 16, 1–4 (2007)
9. Charikar, M., Makarychev, K., Makarychev, Y.: On the advantage over random for maximum acyclic subgraph. In: 48th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2007, pp. 625–633 (2007)
10. Cohen, W.W., Schapire, R.E., Singer, Y.: Learning to order things. J. Artificial Intelligence Research 10, 243–270 (2007)
11. Conitzer, V.: Computer Slater rankings using similarities among candidates. In: Procs. 21st AAAI, pp. 613–619 (2006)
12. Coppersmith, D., Fleischer, L., Rudra, A.: Ordering by weighted number of wins gives a good ranking for weighted tournaments. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, p. 782. ACM, New York (2006)

13. Devanur, N., Hayes, T.: The Adwords Problem: Online Keyword Matching with Budgeted Bidders under Random Permutations. In: Proc. ACM EC (2009); [19] Moreover, there is a polynomial-time approximation scheme (PTAS) for the feedback arc set on tournaments
14. Diaconis, P., Graham, R.: Spearman's footrule as a measure of disarray. Journal of the Royal Statistical Society, Series B 39(2), 262–268 (1977)
15. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggegation methods for the web. In: Procs. 10th WWW, pp. 613–622 (2001), The NP-hardness proof is in the online-only appendix available from `http://www10.org/cdrom/papers/577/`
16. Frieze, A.M., Kannan, R.: Quick approximation to matrices and applications. Combinatorica 19(2), 175–220 (1999)
17. Kenyon, C.: Best-fit bin-packing with random order. In: Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 359–364 (1996)
18. Mathieu, C., Schudy, W.: Yet another algorithm for dense max cut: Go greedy. In: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 176–182. Society for Industrial and Applied Mathematics, Philadelphia (2008)
19. Mathieu, C., Schudy, W.: How to rank with few errors: a PTAS for weighted feedback arc set on tournaments. In: Procs. 39 th ACM STOC, pp. 95–103 (2007), See rather the journal submission available from `http://cs.brown.edu/people/ws/papers/fast_journal.pdf`
20. Seshu, S., Reed, M.B.: Linear Graphs and Electrical Networks. Addison-Wesley, Reading (1961)
21. Slater, P.: Inconsistencies in a schedule of paired comparisons. Biometrika 48, 303–312 (1961)
22. Yao, A.: Probabilistic computations: Toward a unified measure of complexity. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 222–227 (1977)
23. Younger, D.H.: Minimum feedback arc sets for a directed graph. IEEE Trans. Circuit Theory 10, 238–245 (1963)

# Throughput Maximization for Periodic Packet Routing on Trees and Grids*

Britta Peis and Andreas Wiese

Technische Universität Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany
{peis,wiese}@math.tu-berlin.de

**Abstract.** In the periodic packet routing problem a number of tasks periodically create packets which have to be transported through a network. Due to capacity constraints on the edges, it might not be possible to find a schedule which delivers all packets of all tasks in a feasible way. In this case one aims to find a feasible schedule for as many tasks as possible, or, if weights on the tasks are given, for a subset of tasks of maximal weight. In this paper we investigate this problem on trees and grids with row-column paths[1]. We distinguish between direct schedules (i.e., schedules in which each packet is delayed only in its start vertex) and not necessarily direct schedules. For these settings we present constant factor approximation algorithms, separately for the weighted and the cardinality case.

Our results combine discrete optimization with real-time scheduling. We use new techniques which are specially designed for our problem as well as novel adaptions of existing methods.

## 1 Introduction

In the *periodic packet routing problem* (formally defined below) an infinite number of packets (created periodically by a set of tasks) has to be sent through a network without violating given capacities on the edges. This problem is well studied (see e.g. [2,11]) and has numerous applications in theory and practice (e.g., on communication networks with bounded bandwidth).

Networks occurring in practical applications are mostly of rather simple topology such as paths, trees, or grids. Also, in communication networks the transmission links can be used in both directions without interfering with each other. This motivates us to investigate periodic packet routing primarily on bidirected trees and bidirected grids with row-column paths.

In this paper we study the corresponding optimization problem of finding a subset of tasks with maximum weight (w.r.t. given weights on the tasks) such that a feasible schedule for these tasks exists. This problem, which we call the MAX-TASK-problem, combines discrete optimization with real-time scheduling. For this reason our algorithms partly are based on techniques from both of these two research directions. On

* This work was partially supported by Berlin Mathematical School and by the DFG Focus Program 1307 within the project "Algorithm Engineering for Real-time Scheduling and Routing".

[1] A *row-column* path moves along the row of its start vertex $s_i$ to the column of its destination vertex $t_i$, and then along the column of $t_i$ to $t_i$ itself. Such paths have also been studied in [1,5].

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 213–224, 2011.

the other hand, some of our techniques designed for the MAX-TASK-problem might as well be extendable for related discrete optimization and scheduling problems.

Beside practical applications, there is also a second reason why we restrict to trees and grids when searching for constant-factor approximations: even on the simple class of chain graphs the MAX-TASK-problem contains the MAXIMUM-INDEPENDENT-SET-problem (see [12]) which is known to be $NP$-hard to approximate within a factor of $|T|^{1-\epsilon}$ for all $\epsilon > 0$ [16]. The hardness carries over to the MAX-TASK-problem.

*Periodic packet routing.* We now describe our model formally. The periodic packet routing problem is defined as follows: Let $G = (V, E)$ be a graph and let $p$ be an integer. Let $T = \{\tau_0, \tau_1, ..., \tau_{|T|-1}\}$ be a set of tasks. Each task $\tau_i = (s_i, t_i, P_i)$ creates a new packet at all timesteps $t = j \cdot p$ with $j \in \mathbb{N}$. The packets of $\tau_i$ start in the vertex $s_i$ and have to be delivered to $t_i$ along the given path $P_i$ from $s_i$ to $t_i$. We assume that all packets move simultaneously, it takes one timestep to transfer a packet over an edge and each edge can be used by at most one packet at a time in each direction. The goal is now to determine a feasible schedule such that all packets reach their destination vertices within a finite timespan after they have been released. In order to stress that the edges can be used in both direction independently we will use later speak of *bidirected* graphs.

*Template schedules.* Similar as in [2], we define a *template schedule* for $(G, T, p)$ to be given by a map $task : E \times \{0, 1, ..., p-1\} \to T$ with the interpretation that the packets created by a task $\tau_i$ are allowed to use an arc $e$ at time $t$ only if $task\,(e, t \bmod p) = \tau_i$. We assume that packets do not wait unnecessarily at a node when they would be allowed to use the next edge of their path. Such a template schedule is *feasible* if every packet which is ever created reaches its destination vertex eventually. (Note that if in a feasible template schedule two packets are located on the same vertex at the same time this implies that they were created by different tasks.) A schedule is *direct* if each packet which is ever created is delayed only in its start vertex. For ease of notation we say a schedule is *indirect* if it is not necessarily direct.

In this paper we focus on template schedules. Whenever we say that a feasible schedule for a set of tasks exists, we mean that a template schedule exists. Note that in the setting of indirect schedules a feasible (template) schedule for a set of tasks $T'$ exists if and only if no arc is used by more than $p$ tasks in $T'$ [11].

MAX-TASK-*problem.* An instance of the MAXIMUM-TASK-problem (or short: MAX-TASK-problem) is given by a tuple $(G, T, p)$ where each task $\tau_i \in T$ is additionally equipped with a *weight* $w_i$. The problem is now to find a set $T' \subseteq T$ of maximum weight $w(T') = \sum_{\tau_i \in T'} w_i$ such that there exists a feasible schedule for $(G, T', p)$. We distinguish between instances for which we want to find a task selection which allows a direct schedule and instances where we are interested in a task selection which allows an indirect schedule. For an instance $I$ we denote by $OPT_{indir}(I)$ a subset of the tasks of $I$ of maximum weight such that there exists an indirect schedule for these tasks. Likewise, $OPT_{dir}(I)$ denotes a subset of tasks with maximum weight which allows a direct schedule.

*Price of directness.* We also study the *price of directness*. Denote by $\mathcal{I}$ the set of all MAX-TASK-instances on a certain graph class. Intuitively, the price of directness

measures how much we loose at most when we require a direct schedule rather than an indirect schedule. Formally, it is defined by $\max_{I \in \mathcal{I}} \frac{w(OPT_{indir}(I))}{w(OPT_{dir}(I))}$.

*Greedy algorithm.* The greedy algorithm considers the tasks in a given order and adds them if possible. In the setting of indirect schedules the algorithm has to ensure that no arc is used by more than $p$ tasks. If this property holds, we can define a feasible template schedule as follows: for each edge $e$, we define the values $task(e,k)$ such that for each task $\tau$ that uses $e$ there is one value $k_e \in \{0,...,p-1\}$ such that $task(e,k_e) = \tau$. In the setting of direct schedules the greedy algorithm also computes the actual schedule in a map $task$: For each considered task it checks whether there is a start offset for it such that it does not collide with any previously defined task.

The following example shows that the naive approach to order the tasks by non-increasing weight and assign priorities w.r.t. this order may perform arbitrarily bad: Given a path on vertices $V = \{v_0, \ldots, v_k\}$ let $T$ consist of one task $\tau_0 = (v_0, v_k)$ with weight $1+\epsilon$ (for some small $\epsilon > 0$), and $k$ tasks $\tau_i = (v_{i-1}, v_i)$ with weight $w_i = 1$ $(i = 1, \ldots, k)$. Then, if $p = 1$, this simple greedy strategy would select only $\tau_0$ with weight $1+\epsilon$ while the optimal solution would select all of the remaining tasks resulting in a total weight of $k$. This example also indicates that, even in the cardinality case, the MAX-TASK-problem contains the EDGE-DISJOINT-PATH-problem which is known to be $MAXSNP$-hard even on bidirected trees [4].

*General assumptions and notations.* When considering a tree, we always assume that one (arbitrary) vertex $v_r$ is picked as the root vertex. For each task $\tau_i$ we denote by $v_i$ the vertex on $P_i$ which is closest to $v_r$, and call it the *peak vertex* of $\tau_i$. The distance between $v_i$ and $v_r$ is called the *height* $h(\tau_i)$ of $\tau_i$. We assume that the tasks are ordered by non-increasing height, i.e., $i < j \Rightarrow h(\tau_i) \geq h(\tau_j)$. This is in particular important for the greedy algorithm. W.l.o.g. we assume that $|T| > p$ since if $|T| \leq p$ in the settings which we study in this paper we can definitely schedule all tasks. In particular, $|T| > p$ implies that the period $p$ is bounded by a polynomial in the input size.

MAX-TASK *on directed trees.* Note that the MAX-TASK-problem on directed trees (i.e., each edge $e$ is equipped with an orientation and tasks use $e$ only in this one direction) can be solved optimally in polynomial time. Formulated as a linear program, the resulting matrix is a network matrix. Network matrices are uni-modular and linear programs on them can be reduced to the minimum cost flow problem which is polynomial time solvable [13,14].

### 1.1 Our Contribution

In the subsequent Section 2 we study the unweighted MAX-TASK-problem on bidirected trees. We prove that the greedy algorithm with the task ordering given above results in a 2-approximation for the setting of indirect schedules. The analysis of the greedy algorithm uses a reduction of the problem to an integral multicommodity flow problem and a careful adaption of the primal-dual scheme of Garg et al. [6] for the latter problem on undirected trees. A completely different argumentation shows that in the setting of direct schedules the greedy algorithm is a $\max\{2, 3-\frac{2}{p}\}$-approximation. Surprisingly, the resulting set is also a $\max\{2, 3-\frac{2}{p}\}$-approximation in comparison with the largest set of tasks which allows an *indirect* schedule.

In contrast to the cardinality case, the greedy algorithm may perform arbitrarily bad if weights on the tasks are given. In order to obtain a constant factor approximation also for the weighted version we thus need more sophisticated algorithms. In Section 3 we reduce the weighted MAX-TASK-problem on a bidirected tree to a special case of the weighted MAXIMUM-INDEPENDENT-SET-problem, and apply certain integer linear programming techniques (based on ideas of [5]). We obtain an algorithm which is a 3-approximation for both, the direct *and* the indirect setting. For the indirect setting, this result can even be improved: we extend the techniques of [5] to a more general setting than INDEPENDENT-SET-problems and obtain an $\max\{2, 3 - \frac{2}{p}\}$-approximation algorithm. The key idea is to solve the LP-relaxation of our problem and cover the solution with fractions of certain integral solutions. The new concept is that these integral solutions are not necessarily independent sets. This could have applications for many more problems in the fields of integral multicommodity flow and integer packing problems.

In Section 4 we show that the price of directness on bidirected trees is at least $6/5$ and at most 2. Even more, we present an algorithm which splits any set of tasks which allows an indirect schedule into two sets which both allow a direct schedule. For the setting of directed trees we use certain path-coloring techniques to show that any feasible schedule can be turned into a feasible direct schedule. This implies that the price of directness on directed trees is 1.

Finally, in Section 5 we consider bidirected grid graphs where all tasks have row-column paths. In this setting the EDGE-DISJOINT-PATH-problem – and hence the MAX-TASK-problem – are still $MAXSNP$-hard [5]. We prove that we loose at most a factor of 2 if we restrict either to tasks which move left and up or right and down or to tasks which move right and up or left and down. In either case we can split the set of tasks into another two subsets which can be handled separately (with all tasks moving in the same direction). For the unweighted case in the setting of indirect schedules we employ a careful charging argument to get a 4-approximation. For the other settings we can employ similar techniques as for the bidirected tree. Due to space restrictions we spare the details and refer to our technical report [12].

Table 1 shows a summary of the approximation factors of our algorithms for the respective settings.

**Table 1.** Overview of approximation factors for the respective versions of the MAX-TASK-problem

| Graph class | indirect schedules | | direct schedules | | Complexity |
|---|---|---|---|---|---|
| | cardinality | weighted | cardinality | weighted | (all cases) |
| Bidirected tree | 2 | $\max\left\{2, 3 - \frac{2}{p}\right\}$ | $\max\left\{2, 3 - \frac{2}{p}\right\}$ | 3 | $MAXSNP$-hard [4] |
| Bidirected grid | 4 | $\max\left\{4, 6 - \frac{4}{p}\right\}$ | $\max\left\{4, 6 - \frac{4}{p}\right\}$ | 6 | $MAXSNP$-hard [5] |

### 1.2 Related Work

The maximum EDGE-DISJOINT-PATH-problem (a special case of the MAX-TASK-problem that arises when $p = 1$) has been studied by Erlebach et al. [4,5] who present algorithms for the problem on bidirected trees and grid graphs with row-column-paths.

They also show that the problem is $MAXSNP$-hard in these settings, implying that there can be no PTAS unless $P = NP$. For general graphs there are $\sqrt{m}$-approximation algorithms [8,15] which is best possible unless $P = NP$ [7].

In [1] Adler et al. consider the problem of scheduling a maximum number of packets with given release times and deadlines through a network. Like in [4,5] they also consider trees and and grid graphs in which the paths are row-column-paths. For this non-periodic setting, they study the cardinality case as well as the weighted case, providing approximation factors of 3 and 10, respectively. In [5] Erlebach et al. improve the approximation factor for the weighted case to 3. All these schedules are direct.

As mentioned above, we will reduce the problem of finding a feasible schedule to an integral maximum multicommodity flow problem. For the latter problem on undirected trees there is a 2-approximation algorithm for the cardinality case [6] and a 4-approximation for the weighted case [3]. The latter algorithm can be extended to a 4-approximation for bidirected trees [3]. In our case, all edges have the same capacity. If the capacity of each edge is at least 2 there is even a 3-approximation for the weighted case on undirected trees [9]. However, this algorithm cannot be adjusted directly to the setting of bidirected trees.

For the periodic packet routing problem on general graphs, Andrews et al. [2] prove the existence of a feasible template schedule which delivers each packet within $O\left(d_i + 1/r_i\right)$ steps where $d_i$ denotes the length of its path and $r_i$ denotes the insertion rate of its task ($1/r_i = p$ in our notation). For trees and equal period lengths, Peis et al. [11] give improved bounds on the delivery time of each packet. Also, they show that it is $NP$-hard to decide whether for a set of tasks a direct schedule exists.

## 2 Unweighted Tasks in Bidirected Trees

As mentioned above, already on bidirected trees the MAX-TASK-problem is $MAXSNP$-hard, as it contains the EDGE-DISJOINT-PATH-problem. This holds even if all tasks have the same weight and no matter whether we want to compute a direct or indirect schedule. However, we show that the greedy algorithm is a 2-approximation for the case of indirect schedules and a $\eta(p)$-approximation for the case of direct schedules, with $\eta(p) = \max\{3 - \frac{2}{p}, 2\}$. Surprisingly, the resulting set of tasks in the direct case is also by at most a factor of $\eta(p)$ smaller than $w\left(OPT_{indir}\right)$, where $OPT_{indir}$ denotes an optimal set of tasks which allows an *indirect* schedule.

Let $GREEDY_{indir}(I)$ denotes the set of tasks returned by the greedy algorithm for the setting of indirect schedules.

**Theorem 1.** *Let $I$ be an unweighted* MAX-TASK*-instance on a bidirected tree. Then* $|OPT_{indir}(I)| \leq 2 \cdot |GREEDY_{indir}(I)|$.

*Proof.* The claim can be shown using the primal-dual scheme as used in [6] for showing a 2-approximation for the maximum integral multicommodity flow problem (IMCF) on undirected trees. In fact, our problem is a special case of the IMCF-problem: each task corresponds to one commodity with source $s_i$ and sink $t_i$ and each arc is given capacity $p$. To model that each task can be assigned at most once we add an arc with capacity 1 to each $s_i$-$t_i$-path.

As for the IMCF-problem on undirected trees, the dual problem is the minimum multicut problem: We are looking for a set of edges $S$ such that each $s_i$-$t_i$-path uses at least one of the edges in $S$. The construction of such a set can be done as described in [6]. Also, it is possible to show that each $s_i$-$t_i$-path uses at most two arcs from the constructed set. With the primal-dual scheme this proves an approximation factor of two. □

Now we analyze the greedy algorithm in the setting of direct schedules. Denote by $GREEDY_{dir}(I)$ the set of tasks computed by the algorithm. We use a new technique to charge the weight of an optimal solution to the weight of the $GREEDY_{dir}(I)$.

**Theorem 2.** *Let $I$ be an unweighted* MAX-TASK*-instance on a bidirected tree. It holds that $|GREEDY_{dir}(I)| \geq \frac{1}{\eta(p)} |OPT_{indir}(T)| \geq \frac{1}{\eta(p)} |OPT_{dir}(T)|$ with $\eta(p) = \max\left\{2, 3 - \frac{2}{p}\right\}$.*

*Proof.* We consider the set $DIFF := OPT_{indir}(T) \setminus GREEDY_{dir}(I)$. For each task $\tau_j \in GREEDY_{dir}(I)$ we introduce a variable $\beta_j$. Now let $\tau_i \in DIFF$ be a task. For each possible delay $d \in \{0, 1, ..., p-1\}$ there must be an edge $e_j \in P_i$ and a task $\tau_{i(d)} \in GREEDY_{dir}(I)$ with $i(d) < i$ such that $task(e_j, (d+j) \bmod p) = \tau_{i(d)}$. For each task $\tau_{i(d)}$ with $d \in \{0, 1, ..., p-1\}$ we increase $\beta_{i(d)}$ by $\frac{1}{p}$. We say that the task $\tau_{i(d)}$ *pays* for the task $\tau_i$. We do this procedure for all tasks $\tau_i \in DIFF$. Note that after this $\sum_i \beta_i = DIFF$.

We define $T_1' := GREEDY_{dir}(I) \cap OPT_{indir}(T)$ and $T_2' := GREEDY_{dir}(I) \setminus OPT_{indir}(T)$. We claim that $\beta_i \leq 2 \cdot \frac{p-1}{p}$ for all tasks $\tau_i \in T_1'$ and $\beta_i \leq 2$ for all tasks $\tau_i \in T_2'$. Let $\tau_i \in GREEDY_{dir}(I)$ and denote by $P_i$ the path of $\tau_i$ and by $v_i \in P_i$ the vertex on $P_i$ which is closest to $v_r$. Denote by $\bar{e}$ and $\tilde{e}$ the edges on $P_i$ which are incident to $v_i$. Let $\tilde{\tau}$ be a task which $\tau_i$ pays for. Since $h(\tilde{\tau}) \leq h(\tau_i)$ we conclude that $\tilde{\tau}$ either uses $\bar{e}$ or $\tilde{e}$. Since in $OPT_{indir}(T)$ there can be at most $2p$ such tasks we conclude that $\beta_i \leq 2$. Moreover, if $\tau_i \in OPT_{indir}(T)$ then there can be at most $2(p-1)$ tasks in $DIFF$ which use $\bar{e}$ or $\tilde{e}$. Thus, if $\tau_i \in OPT_{indir}(T)$ then $\beta_i \leq 2 \cdot \frac{p-1}{p}$. We complete the proof by calculating that $|OPT_{dir}(T)| = \sum_{i:\tau_i \in T_1'} \beta_i + |T_1'| + \sum_{i:\tau_i \in T_2'} \beta_i \leq 2 \cdot \frac{p-1}{p} \cdot |T_1'| + |T_1'| + 2 \cdot |T_2'|$ which is bounded from above by $\max\left\{2, 3 - \frac{2}{p}\right\} |GREEDY_{dir}(I)|$. This shows that $|GREEDY_{dir}(I)| \geq \frac{1}{\eta(p)} |OPT_{indir}(T)|$. The fact that $|OPT_{indir}(T)| \geq |OPT_{dir}(T)|$ completes the proof. □

## 3 Weighted Tasks on Bidirected Trees

Now we study the weighted MAX-TASK-problem. We use techniques based on [5]. The idea is to formulate the problem as an integer program and solve the LP-relaxation. Then we employ a technique to cover the solution of the LP with fractions of certain integral solutions. We can show that the weight of one of these integral solutions is by at most a constant factor smaller than the weight of the fractional solution.

First, we present a 3-approximation algorithm for the setting of direct schedules, using a reduction to a special case of weighted MAXIMUM-INDEPENDENT-SET. By a

careful analysis of the LP-relaxations of the respective problems we show that the total weight of the resulting set is also by at most a factor of 3 smaller than $w(OPT_{indir})$ and hence it also works as a 3-approximation in the setting of indirect schedules. Finally, we present a $\max\{2, 3 - \frac{2}{p}\}$-approximation algorithm for the setting of indirect schedules. As a novel concept in this algorithm the integral solutions used for the covering are not necessarily independent sets. (In particular, here we extend the techniques from [5].) We also prove that the respective LP has an integrality gap of at most $\max\{2, 3 - \frac{2}{p}\}$.

*Reduction to Weighted* MAXIMUM-INDEPENDENT-SET. Let $I = (G, T, p)$ be an instance of the weighted MAX-TASK-problem on a bidirected tree $G$ in the setting of direct schedules. We reduce the problem to an instance of the weighted MAXIMUM-INDEPENDENT-SET-problem. We define the graph $G_{MIS} = (V_{MIS}, E_{MIS})$ as follows: for each task $\tau_i$ we introduce $p$ vertices $\langle \tau_i, k \rangle$ with $0 \leq k \leq p-1$. A vertex $\langle \tau_i, k \rangle$ corresponds to scheduling the task $\tau_i$ such that it uses the first edge on its path at times $t$ with $t \bmod p = k$. We call such a value $k$ the *offset* of $\tau_i$. We connect two vertices $\langle \tau_i, k \rangle$, $\langle \tau_j, \ell \rangle$ by an edge if and only if either

- $\tau_i = \tau_j$ or
- $P_i$ and $P_j$ use an edge in the same direction and if $\tau_i$ and $\tau_j$ had the offsets $k$ and $\ell$ their packets would collide.

We assign each vertex $\langle \tau_i, k \rangle$ the weight $w_i$. Then any solution for weighted maximum independent set on $G_{MIS}$ corresponds to a solution for $I$ with the same weight and vice versa.

Note that the size of $G_{MIS}$ is bounded by a polynomial in the size of $I$. We define the following linear programs $LP_I$ and $LP_w$. First, we consider the $LP$-relaxation $LP_I$ of the weighted maximum independent set problem on $G_{MIS}$. For defining the linear program we need certain maximal cliques $\{\langle \tau_{i_1}, k_1 \rangle, \langle \tau_{i_2}, k_2 \rangle, ..., \langle \tau_{i_m}, k_m \rangle\}$: The cliques that arise because $\tau_{i_1} = \tau_{i_2} = \tau_{i_3} = ... = \tau_{i_m}$ and the cliques which arise because there is an edge $e$ which is used by each of the tasks $\tau_{i_1}, ..., \tau_{i_m}$ at the same time if they are assigned the offsets $k_1, ..., k_m$. Denote by $\mathcal{C}$ the set of all these cliques. Note that the size of $\mathcal{C}$ is bounded by a polynomial in the size of $G_{MIS}$. We define $LP_I$ by

$$
\begin{aligned}
(LP_I)\ \max \quad & \sum_{\langle \tau_i, k \rangle \in V_{MIS}} w_i \cdot x_{i,k} & \\
\text{s.t.} \quad & \sum_{\langle \tau_i, k \rangle \in C} x_{i,k} \leq 1 & \forall C \in \mathcal{C} \\
& 0 \leq x_{i,k} \leq 1 & \forall \langle \tau_i, k \rangle \in V_{MIS}
\end{aligned}
$$

Since the size of $LP_I$ is bounded by a polynomial we can solve it optimally in polynomial time.

*Fractional Coloring.* Let $x^*$ be an optimal solution of $LP_I$. We now interpret each value $x^*_{i,k}$ as a cost value and define $c_{i,k} := x^*_{i,k}$. We define a linear program $LP_w$ which computes a fractional coloring for the vertices in $V_{MIS}$. Each vertex $\langle \tau_i, k \rangle \in V_{MIS}$ has to be colored with colors whose total weight is at least $c_{i,k}$. This can be seen as assigning

each vertex a set of disjoint intervals of total length $c_{i,k}$ such that the intervals of two adjacent vertices do not intersect. For the definition of $LP_w$ we denote by $\mathcal{J}$ the set of all independent sets in $G_{MIS}$ (note that a coloring can be understood as a partition into independent sets).

$$
\begin{aligned}
(LP_w)\ \min \sum_{J\in\mathcal{J}} y_J & \\
\text{s.t.} \sum_{J\in\mathcal{J}\mid\langle\tau_i,k\rangle\in J} y_J &\geq c_{i,k} && \forall\,\langle\tau_i,k\rangle \in V_{MIS}\\
0 \leq y_J &\leq 1 && \forall J\in\mathcal{J}
\end{aligned}
$$

In ordinary graph coloring, the clique number $\omega(G)$ of a graph $G$ is a lower bound on the number of needed colors. Likewise, we define a fractional clique number $\omega_{\mathcal{C}}\left(G_{MIS}, c\right) := \max_{C\in\mathcal{C}(G)} \sum_{\langle\tau_i,k\rangle\in C} c_{i,k}$ which is also a lower bound on the total weight of the needed colors in our setting. After having computed the optimal solution to $LP_I$, we compute an approximative solution for $LP_w$ as described in the following lemma.

**Lemma 1.** *There is a polynomial time algorithm which computes a solution $y$ for $LP_w$ with $\sum_{J\in\mathcal{J}} y_J \leq 3\cdot\omega_{\mathcal{C}}\left(G_{MIS}, c\right)$.*

*Proof.* We order the vertices $\langle\tau_i,k\rangle$ non-descendingly by the height of their peak vertex $v_i$. Then we assign each vertex $\langle\tau_i,k\rangle$ (greedily) color intervals of total length $c_{i,k}$ such that the intervals of two adjacent vertices do not intersect. From this we can extract a value $y_J$ for each independent set $J$. When considering a vertex $\langle\tau_i,k\rangle$ we observe that colors of weight at most $2\cdot\omega_{\mathcal{C}}\left(G_{MIS}, c\right) - 2c_{i,k}$ cannot be used due to vertices $\langle\tau_j,\ell\rangle$ with $\tau_j\neq\tau_i$ such that $\tau_j$ uses one of the edges on $P_i$ adjacent to $v_i$. Also, at most $\omega_{\mathcal{C}}\left(G_{MIS}, c\right) - c_{i,k}$ cannot be used due to vertices $\langle\tau_i,\ell\rangle$ with $\ell\neq k$. All other colors are available. Therefore, colors of total weight $3\cdot\omega_{\mathcal{C}}\left(G_{MIS}, c\right)$ suffice for the greedy algorithm. □

Having computed the solution $y$ for $LP_w$, we output the independent set $J\in\mathcal{J}$ of maximum weight for which $y_J > 0$. Note that since $y$ was computed in polynomial time there can be only a polynomial number of such independent sets. Denote by $BT_{dir}(I)$ the set of tasks corresponding to the vertices in $J$.

**Theorem 3.** *Let $I$ be a weighted* MAX-TASK*-instance on a bidirected tree. It holds that $w\left(BT_{dir}(I)\right) \geq \frac{1}{3} w\left(OPT_{dir}(I)\right)$.*

*Proof.* Let $w\left(x^*\right)$ denote the objective value of the optimal solution $x^*$ of $LP_I$. The key idea for the proof is that there is an independent set $J$ with $y_J > 0$ such that $w(J) \geq w\left(x^*\right)/3$. So now assume on the contrary that for all $J\in\mathcal{J}$ with $y_J > 0$ we have that $w(J) < w\left(x^*\right)/3$. Note that any solution $y$ for $LP_w$ satisfies $\sum_{J\in\mathcal{J}} w(J)\cdot y_J \geq w\left(x^*\right)$. We calculate that

$$
w\left(x^*\right) \leq \sum_{J\in\mathcal{J}} w(J)\cdot y_J < \frac{w\left(x^*\right)}{3} \sum_{J\in\mathcal{J}} y_J \leq w\left(x^*\right)\cdot\omega_{\mathcal{C}}\left(G_{MIS}, c\right) \leq w\left(x^*\right)
$$

which is a contradiction. □

Surprisingly, $BT_{dir}(I)$ is also a 3-approximation in comparison with the set of tasks with optimal weight which allows an *indirect* schedule. In the proof of Theorem 3 we showed that $w\left(BT_{dir}(I)\right) \geq w\left(x^*\right)/3$ where $x^*$ is the optimal solution for $LP_I$. Now we prove that $w\left(x^*\right)$ is also an upper bound on the optimal weight obtained by a task selection which allows an indirect schedule. The latter problem can be formulated by an integer program with the LP-relaxation $LP'_I$ defined by

$$(LP'_I)\, \max\{w^T x | x \in Q\}$$

where $Q = \{x \in \mathbb{R}^{|T|} | \sum_{\tau_i \in C} x_i \leq p \; \forall C \in \mathcal{C}'\}$. Here, we derive the set of cliques $\mathcal{C}'$ by considering every arc $e$ and taking the clique consisting of all tasks which use $e$. (The original integer program is obtained by additionally requiring $x_i \in \{0,1\}$ for all $x_i$.)

Even though $LP_I$ and $LP'_I$ differ significantly we show that their optimal solutions have the same value. Let $x'$ denote an optimal solution for $LP'_I$ with value $w\left(x'\right)$. Now we show how to transform $x'$ to a solution $x$ for $LP_I$ with the same weight. For each variable $x_{i,k}$ with $\tau_i \in T$ and $k \in \{0,1,...,p-1\}$ we define $x_{i,k} := x'_i/p$. Hence, for the optimal value $w\left(x^*\right)$ of $LP_I$ we have that $w\left(x^*\right) \geq w\left(x'\right)$. Moreover, every optimal solution $x^*$ for $LP_I$ yields a solution $\bar{x}$ of $LP'_I$ with the same weight: We define $\bar{x}_i := \sum_{k=1}^{p} x^*_{i,k}$ which implies $w(\bar{x}) = w(x^*)$. Hence, $w\left(x^*\right) = w\left(x'\right)$.

Since $LP'_I$ is a relaxation of the original integer program, $w\left(x'\right)$ is an upper bound on the maximum weight of a set of tasks which allow an indirect schedule. We conclude with the following theorem.

**Theorem 4.** *Let $I$ be a weighted* MAX-TASK*-instance on a bidirected tree. It holds that $w\left(BT_{dir}(I)\right) \geq \frac{1}{3} w\left(OPT_{indir}(I)\right)$.*

*Proof.* Follows from $w\left(BT_{dir}(I)\right) \geq \frac{w(x^*)}{3} \geq \frac{w(x')}{3} \geq w\left(OPT_{indir}(I)\right)/3$. □

Now we describe how an optimal (fractional) solution $x'$ of $LP'_I$ can be transformed to an integral solution whose weight is at least $\frac{1}{\eta(p)} w(x')$. We use a novel approach which is based on covering the fractional solution with integral solutions which are not necessarily independent sets. For each task $\tau_i$ we interpret the value $x'_i$ as a cost value and define $c_i := x'_i$. Denote by $\mathcal{J}'$ the set of all valid solutions to our problem. We consider the following linear program $LP'_w$.

$$\begin{aligned}
(LP'_w)\, \min & \sum_{J \in \mathcal{J}'} y'_J & \\
\text{s.t.} & \sum_{J \in \mathcal{J}' | \tau_i \in J} y'_J \geq c_i & \forall \tau_i \in T \\
& 0 \leq y'_J \leq 1 & \forall J \in \mathcal{J}'
\end{aligned}$$

Intuitively, the solutions in $\mathcal{J}'$ – represented by the $y'_J$ variables – cover the $c_i$ (and hence $x'$).

**Lemma 2.** *Assume that the values $c_i$ in $LP'_w$ are constructed from a valid solution $x'$ for $LP'_I$. Then there is a polynomial time algorithm which computes a solution $y'$ for $LP'_w$ such that $\sum_{J \in \mathcal{J}'} y'_J \leq \max\{2, 3 - \frac{2}{p}\} =: \eta(p)$.*

*Proof.* We interpret the problem as assigning each task $\tau_i$ a set of disjoint subintervals of $[0, \eta(p))$ with total length $c_i$. We do this such that for each arc $e$ we have that for each $t \in [0, \eta(p))$ there are at most $p$ tasks using $e$ which are assigned intervals containing $t$. We call this property the *packing property*. We sort the tasks non-descendingly by the height of their peak vertices. The intervals are then assigned greedily such that the packing property is always fulfilled. We can show that within $[0, \eta(p))$ such intervals always exist. For further details cf. [12]. □

Let $y'$ be a solution for $LP'_w$ which is constructed as described in Lemma 2. Our integral solution for $LP'_I$ is obtained by taking the solution $J \in \mathcal{J}'$ with maximum weight such that $y'_J > 0$. Denote by $BT_{indir}(I)$ the resulting tasks.

**Theorem 5.** *Let $I$ be an instance of the weighted* MAX-TASK*-problem on a bidirected tree in the setting of indirect schedules. For the solution $BT_{indir}(I)$ it holds that $w(BT_{indir}(I)) \geq \frac{1}{\eta(p)} w(OPT_{indir}(I))$ with $\eta(p) = \max\{2, 3 - \frac{2}{p}\}$. Moreover, the integrality gap of $LP'_I$ is bounded by $\eta(p)$.*

*Proof.* Can be shown similarly as Theorem 3. □

## 4 Price of Directness on Trees

Now we study the price of directness. For bidirected trees we show an upper bound of 2 and a lower bound of $6/5$. For proving the upper bound we give a new algorithm which splits any set of tasks which allows an indirect schedule into two sets of tasks which both allow a direct schedule. The algorithm computes both the sets and the direct schedules. For the setting of directed trees (i.e., each edge $e$ is equipped with an orientation and tasks use $e$ only in this one direction) we show that the prize of directness is 1.

**Theorem 6.** *Let $I = (G, T, p)$ be a* MAX-TASK*-instance on a bidirected tree such that there is an indirect schedule for the set of all tasks $T$. There is a polynomial time algorithm which computes sets $T^1$ and $T^2$ with $T = T^1 \dot\cup T^2$ and direct schedules for $T^1$ and $T^2$.*

*Proof.* We describe the algorithm. For both sets $T^k$ we maintain a map $task_k : E \times \{0, 1, ..., p-1\} \to T$ which is initially defined by $task_k(e, j) := none$ for all arcs $e$ and all $j \in \{0, 1, ..., p-1\}$. We order the tasks in $T$ *non-descendingly* by the height of their peak vertex. We consider the tasks one by one. In the $i$-th iteration we consider the task $\tau_i \in T$. Let $e_0, ..., e_{|P_i|-1}$ be the edges on $P_i$. We try to assign $\tau_i$ an initial start offset $d$ with which it fits into one of the maps $task_k$. We say a value $d$ is *blocked* in a map $task_k$ if there is an edge $e_j$ such that $task_k(e_j, d + j \bmod p) \neq none$. We want to determine what values for $d$ are blocked for what maps $task_k$. The ordering of the task yields the following useful property: it suffices to check whether a value $d$ is blocked in one of the edges $\bar e$ and $\tilde e$ incident to the peak vertex $v_i$ of $\tau_i$. There can be in total at most $2p - 2$ task different from $\tau_i$ using $\bar e$ and $\tilde e$. Each such task can block at most one value $d$ in one of the maps $task_k$. We conclude that there is always one value $d$ which is *not* blocked in $task_1$ or $task_2$. Let $task_k$ be the map in which a value $d$ is not blocked. We then define $task_k(e_j, d + j \bmod p) := \tau_i$ for all edges $e_j \in P_i$ and we assign $\tau_i$ to the set $T^k$. This procedure defines the sets $T^1$ and $T^2$. □

Since $T = T^1 \cup T^2$ we conclude that either $w(T^1) \geq \frac{1}{2}w(T)$ or $w(T^2) \geq \frac{1}{2}w(T)$. This gives the following corollary.

**Corollary 1.** *The price of directness for the weighted* MAX-TASK*-problem on bidirected trees is upper-bounded by* 2.

Proposition 14 in our technical report [12] describes an instance $\bar{I}$ on a bidirected tree such that $OPT_{indir}(\bar{I})/OPT_{dir}(\bar{I}) = 6/5$. Hence, the price of directness on bidirected trees is at least $6/5$. Finally, we study the price of directness for directed trees.

**Theorem 7.** *Let $I = (G, T, p)$ be an instance of the* MAX-TASK*-problem on a directed tree and let $T' \subseteq T$ denote a set of tasks for which there is an indirect schedule. Then there also exists a direct schedule for the tasks $T'$. Hence, the price of directness for the weighted* MAX-TASK*-problem on directed trees is* 1.

*Proof.* Can be shown using techniques based on path colorings and time-dependent edge-coloring introduced in [10]. For full details see [12]. □

## 5 Grid Graphs

In this section we study the MAX-TASK-problem on bidirected grid graphs. As it is common in the literature [1,5] we assume that the paths of the tasks are row-column-paths. (Note that with arbitrary given paths the problem would contain the INDEPENDENT-SET-problem and hence there would be no non-trivial polynomial time approximation algorithm unless $P = NP$.)

Due to space constraints we only outline the used techniques and state our main results. For further details we refer to our technical report [12]. For our algorithms we employ subroutines which compute an approximative solution $ALG(T_{ru}) \subseteq T_{ru}$ for sets of tasks $T_{ru}$ in which the paths of all tasks move to the right and then up. Lemma 15 in [12] shows that an $\frac{1}{\alpha}$-approximation algorithm for such sets $T_{ru}$ yields a $\frac{1}{2\alpha}$-approximation algorithm for any set of tasks in the bidirected grid.

For the greedy algorithm the tasks are ordered non-descendingly by the grid column of the bend vertex of their path (ties are broken by its grid row, non-descendingly).[2]

**Lemma 3.** *Let $I = (G, T_{ru}, p)$ be an instance of the unweighted* MAX-TASK*-problem on the bidirected grid where the paths of all tasks move to the right and then up. Then it holds that $|GREEDY_{indir}(T_{ru})| \geq \frac{1}{2}|OPT_{indir}(T_{ru})|$.*

*Proof.* Can be shown with a careful charging argument, see Lemma 16 in [12]. □

Lemma 3 and Lemma 15 in [12] yield the following theorem.

**Theorem 8.** *There is a 4-approximation algorithm for the unweighted* MAX-TASK*-problem on the bidirected grid in the setting of indirect schedules.*

With similar arguments as in Section 2 we can show that $|GREEDY_{dir}(T_{ru})| \geq \frac{1}{\eta(p)} \cdot |OPT_{dir}(T_{ru})|$ for sets $T_{ru}$ as described above. This gives the following theorem.

[2] We assume that the grid columns and grid rows increase when moving to the right or down, respectively.

**Theorem 9.** *There is an* $\frac{1}{2\eta(p)}$*-approximation algorithm for the unweighted* MAX-TASK*-problem on the bidirected grid in the setting of direct schedules.*

We can handle the weighted case with similar techniques as in Section 3.

**Theorem 10.** *There is a* $\frac{1}{6}$*-approximation algorithm for the weighted* MAX-TASK*-problem on the bidirected grid in the setting of direct schedules and a* $\frac{1}{2\eta(p)}$*-approximation algorithm for the setting of indirect schedules.*

An adaption of our insights for the price of directness on bidirected trees yield the following theorem.

**Theorem 11.** *The price of directness on the bidirected grid is at most* $4$ *and at least* $6/5$.

## 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. Andrews, M., Fernández, A., Harchol-Balter, M., Leighton, F., Zhang, L.: General dynamic routing with per-packet delay guarantees of O (distance + 1/session rate). SIAM Journal of Computing 30, 1594–1623 (2000)
3. Chekuri, C., Mydlarz, M., Shepherd, F.: Multicommodity demand flow in a tree and packing integer programs. ACM Trans. Algorithms 3, 27 (2007)
4. Erlebach, T., Jansen, K.: The maximum edge-disjoint paths problem in bidirected trees. SIAM Journal of Discrete Mathematics 14, 326–355 (2001)
5. Erlebach, T., Jansen, K.: Conversion of coloring algorithms into maximum weight independent set algorithms. Discrete Applied Mathematics 148, 107–125 (2005)
6. Garg, N., Vazirani, V.V., Yannakakis, M.: Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica 18, 3–20 (1997)
7. Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., Yannakakis, M.: Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. Journal of Computer and System Sciences 67, 473–496 (2003)
8. Kolliopoulos, S.G., Stein, C.: Approximating disjoint-path problems using packing integer programs. Mathematical Programming 99, 63–87 (2004)
9. Könemann, J., Parekh, O., Pritchard, D.: Max-weight integral multicommodity flow in spiders and high-capacity trees. In: Bampis, E., Skutella, M. (eds.) WAOA 2008. LNCS, vol. 5426, pp. 1–14. Springer, Heidelberg (2009)
10. Peis, B., Skutella, M., Wiese, A.: Packet routing: Complexity and algorithms. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 217–228. Springer, Heidelberg (2010)
11. Peis, B., Stiller, S., Wiese, A.: The periodic packet routing problem. Technical Report 008-2010, Technische Universität Berlin (April 2010)
12. Peis, B., Wiese, A.: Throughput maximization for periodic packet scheduling. Technical Report 013-2010, Technische Universität Berlin (June 2010)
13. Schrijver, A.: Theory of Linear and Integer Programming. Wiley-Interscience Series in Discrete Mathematics. J. Wiley & Sons, Chichester (1986)
14. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer, Berlin (2003)
15. Srinivasan, A.: Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS), pp. 416–425 (1997)
16. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory of Computing 3, 103–128 (2007)

# $k$-Edge-Connectivity: Approximation and LP Relaxation

David Pritchard*

École Polytechnique Fédérale de Lausanne

**Abstract.** In the $k$-edge-connected spanning subgraph problem we are given a graph $(V, E)$ and costs for each edge, and want to find a minimum-cost $F \subset E$ such that $(V, F)$ is $k$-edge-connected. We show there is a constant $\epsilon > 0$ so that for all $k > 1$, finding a $(1 + \epsilon)$-approximation for $k$-ECSS is NP-hard, establishing a gap between the unit-cost and general-cost versions. Next, we consider the *multi-subgraph* cousin of $k$-ECSS, in which we purchase a *multi-subset* $F$ of $E$, with unlimited parallel copies available at the same cost as the original edge. We conjecture that a $(1 + \Theta(1/k))$-approximation algorithm exists, and we describe an approach based on graph decompositions applied to its natural linear programming (LP) relaxation. The LP is essentially equivalent to the Held-Karp LP for TSP and the undirected LP for Steiner tree. We give a family of extreme points for the LP which are more complex than those previously known.

**Keywords:** graphs, network design, edge-connectivity, linear programs, Held-Karp relaxation, approximation algorithms, inapproximability.

## 1 Introduction

In the $k$-edge-connected spanning subgraph problem ($k$-ECSS), we are given an input graph $G$ with edge costs, and must select a minimum-cost subset of edges so that the resulting graph has edge-connectivity $k$ between all vertices. This is a natural problem for applications, since it is the same as seeking resilience against $(k-1)$ edge failures, or the ability to route $k$ units of flow between any pair of vertices. A natural variant of $k$-ECSS is to allow each edge to be purchased repeatedly, as many times as desired, with each copy at the same cost. We call this the *$k$-edge-connected spanning multi-subgraph problem* ($k$-ECSM).

When $k = 1$ the $k$-ECSS and $k$-ECSM problems are both equivalent to the *minimum spanning tree* problem, which is well-known to be solvable in polynomial time, but they are non-trivial for $k > 1$. We consider *approximation algorithms* for these problems: an algorithm that approximately solves $k$-ECSS or $k$-ECSM is said to be an *$\alpha$-approximation*, or have *approximation ratio* $\alpha$, if it always outputs a solution with cost at most $\alpha$ times optimal.

* Partially supported by an NSERC post-doctoral fellowship.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 225–236, 2011.

Here we survey the oldest and newest results for $k$-ECSM and $k$-ECSS. Frederickson & Jájá gave a 2-approximation algorithm for 2-ECSS [19], and a 3/2-approximation in the special case of metric costs [20]. A 3/2-approximation is possible for 2-ECSM [9]. For $k$-ECSS/$k$-ECSM in general, Khuller & Vishkin [26] gave a matroid-based 2-approximation, and Jain's iterated LP rounding framework [25] also gives a 2-approximation. Goemans & Bertsimas [23] give an approximation algorithm for $k$-ECSM with ratio $\frac{3}{2}$ when $k$ is even, and $(\frac{3}{2} + \frac{1}{2k})$ when $k$ is odd. Fernandes [18] showed 2-ECSS is APX-hard, even for unit costs.

An important special case is where all edges have unit cost. Then $k$-ECSS gets *easier* to approximate as $k$ gets larger: Gabow et al. [22] gave an elegant $(1+2/k)$-approximation algorithm for $k$-ECSS/$k$-ECSM using iterated LP rounding, and they showed that for some fixed $\epsilon > 0$, for all $k > 1$, it is NP-hard to get a $(1 + \epsilon/k)$-approximation algorithm for unit-cost $k$-ECSS. Together, these establish a $1+\Theta(1/k)$ approximability threshold for unit-cost $k$-ECSS. Improvements to the constant, and improvements in the special case that the input graph is simple, appear in Cheriyan & Thurimella [14] and Gabow & Gallagher [21].

### 1.1 Contributions

**Hardness Results (Section 2).** Our first main result is the following hardness for $k$-ECSS:

**Theorem 1.** *There is a constant $\epsilon > 0$ so that for all $k \geq 2$, it is* NP*-hard to approximate $k$-ECSS within ratio $1 + \epsilon$, even if the costs are* 0-1*.*

Although $\epsilon \approx \frac{1}{300}$ here is small, the qualitative difference is important: whereas the approximability of unit-cost $k$-ECSS tends to 1 as $k$ tends to infinity, we see that the approximability of general-cost $k$-ECSS is bounded away from 1.

Next we establish a relatively straightforward hardness result for $k$-ECSM.

**Proposition 2.** *The 2-ECSM problem is* APX*-hard.*

The key step is to show that 2-ECSM and *metric 2-ECSS* are basically the same problem. First, we use the following well-known fact: in $k$-ECSM, the input is metric without loss of generality [23] (i.e. the graph is complete and its costs satisfy the triangle inequality).[1] Then, simple reduction techniques show that under metric costs, any 2-ECSM can be efficiently converted to a 2-ECSS without increasing the costs. We remark that this approach also yields a simpler 3/2-approximation for 2-ECSM (c.f. [9]), using the 3/2-approximation for metric 2-ECSS [20] as a black box.

What Proposition 2 leaves to be desired is hardness for $k$-ECSM, $k > 2$, and asymptotic dependence on $k$. Why is it hard to show these problems are hard? The hard instances for $k$-ECSS given by Theorem 1 and [22] contain certain

[1] To see this, take the metric closure (i.e. shortest path costs), solve it, and replace each $uv$-edge in the solution with a shortest $u$-$v$ path from the original graph; it is not hard to show this preserves $k$-edge-connectivity. In $k$-ECSS, note metricity is not WOLOG, since the replacement step here can introduce multiple edges.

*mandatory parts* that are "without loss of generality" included in the optimal feasible solution; the argument proceeds to show hardness of the residual problem once the mandatory parts are included. But coming up with suitable mandatory parts for $k$-ECSM, while keeping the residual problem hard, is tricky: e.g. the proof of Theorem 1 will use a spanning tree of zero-cost edges, but in $k$-ECSM this leads to a trivial instance (buy that spanning tree $k$ times). The known hardness for $k$-VCSS (vertex connectivity) by Kortsarz et al. [28] is similar: we take hard instances of 2-VCSS and add $(k-2)$ new vertices, connected to all other vertices by 0-cost (mandatory) edges. A new trick seems to be needed to get a good hardness result for $k$-ECSM.

***k*-ECSM Conjecture (Section 3).** We conjecture that approximation ratio $1 + O(1/k)$ should be possible for $k$-ECSM, using LPs. Obtain the natural LP relaxation of $k$-ECSM by allowing edges to be purchased fractionally: introduce a variable $x_e$ for each edge, and require that there is a fractional value of at least $k$ spanning each cut (see Figure 1, where $\delta(S)$ denotes the set of edges with exactly one end in $S$).

$$\min \Big\{ \sum_{e \in E} c_e x_e : \quad x \in \mathbb{R}^E \qquad (\mathcal{N}_k)$$
$$\sum_{e \in \delta(S)} x_e \geq k, \quad \forall \varnothing \neq S \subsetneq V$$
$$x_e \geq 0, \quad \forall e \in E \Big\}$$

$$\min \Big\{ \sum_{e \in E} c_e x_e : \quad x \in \mathbb{R}^E \qquad (\mathcal{N}'_k)$$
$$\sum_{e \in \delta(v)} x_e = k, \quad \forall v \in V$$
$$\sum_{e \in \delta(S)} x_e \geq k, \quad \forall \varnothing \neq S \subsetneq V$$
$$x_e \geq 0, \quad \forall e \in E \Big\}$$

**Fig. 1.** The undirected relaxation for $k$-edge connected spanning multi-subgraph. The unbounded version $(\mathcal{N}_k)$ is on the left, the bounded version $(\mathcal{N}'_k)$ is on the right. They have the same value for metric costs, including all $k$-ECSM instances.

**Conjecture 3.** *There is a polynomial-time approximation algorithm for $k$-ECSM which produces a solution of value at most $(1 + C/k) \cdot \mathrm{OPT}(\mathcal{N}_k)$ for some universal constant $C$.*

This conjecture implies a $(1 + C/k)$-approximation algorithm, since $\mathrm{OPT}(\mathcal{N}_k)$ is a lower bound on the optimal $k$-ECSM cost. What makes us think Conjecture 3 is true? First, we know it holds for unit costs. Second, the same holds in related high-*width* problems; to explain, say an integer program has width $W$ if in every constraint, the right-hand side is at least $W$ times every coefficient. Multicommodity flow/covering problems in trees are closely related to $(\mathcal{N}_k)$ via *uncrossing* (e.g. [25,22,21]) and they admit an LP-based $1 + O(1/W)$-approximation algorithm [27] (in that setting $W$ is the minimum edge capacity). Similar phenomena are known for LP relaxations of other structured integer

programs [13,12,29,6]. In $k$-ECSM the width is $k$ so one may view our conjecture as seeking integrality gap[2] and approximation ratio $1 + O(1/W)$.

Later, we show an open problem of [5] — can every $k$-edge connected graph be partitioned into two spanning $(\frac{k}{2} - O(1))$-edge-connected subgraphs? — would imply a nonconstructive version of Conjecture 3. Few partial results towards Conjecture 3 are known: the integrality gap of $(\mathcal{N}_1)$ is $2(1-1/n)$ [23], and that of $(\mathcal{N}_2)$ is at most 3/2 [31]. For general $k$, the best integrality gap bounds known for $(\mathcal{N}_k)$ come from the approximation algorithms [25,23,21,22] mentioned earlier.

One further motivation to investigate the conjecture has to do with the *parsimonious property* of Goemans & Bertsimas [23]. Using metricity and *splitting-off*, they showed the constraint $\forall v \in V : x(\delta(v)) = k$ can be added to $(\mathcal{N}_k)$ without affecting the value of the LP (the strengthened LP $(\mathcal{N}'_k)$ is shown in Figure 1). As observed in [23], parsimony implies that Conjecture 3 would give a $(1 + \frac{C}{k})$-approximation algorithm for *subset $k$-ECSM*, where we require edge-connectivity $k$ only amongst a pre-specified set of terminal nodes (generalizing the Steiner tree problem). Thus even if we don't care about LPs *a priori*, they have algorithmic dividends in Conjecture 3.

**Complex Extreme Points (Section 4).** In both of the LPs $(\mathcal{N}_k)$ and $(\mathcal{N}'_k)$, note that $k$ serves only as a scaling factor: $x$ is feasible for $(\mathcal{N}_1)$ iff $kx$ is feasible for $(\mathcal{N}_k)$. In fact, these LPs are well-studied: $(\mathcal{N}_1)$ is equivalent (by the parsimonious property [23]) to the *undirected cut relaxation of the Steiner tree problem* and $(\mathcal{N}'_2)$ is the *Held-Karp relaxation of the Traveling Salesman Problem*. We demonstrate a family of extreme point solutions to these ubiquitous LPs which are more complex than were previously known.

For a solution $x$, the *support* is the edge set $\{e \mid x_e > 0\}$, and the *support graph* is the graph with vertex set $V$ and the support for its edge set. The *fractionality* of $x$ is $\min\{x_e \mid e \in E, x_e > 0\}$.

**Theorem 4.** *There are extreme point solutions for the linear program $(\mathcal{N}'_2)$ with fractionality exponentially small in $|V|$, and whose support graph has maximum degree linear in $|V|$.*

The members of the family are also extreme point solutions for $(\mathcal{N}_2)$, since $(\mathcal{N}'_2)$ is a face of $(\mathcal{N}_2)$. The motivation for this theorem comes from a common design methodology in LP-based approximation algorithms [25,22,24,30]: algorithmically exploit good properties of extreme point solutions. E.g., Jain's algorithm [25] uses the fact that when $(\mathcal{N}_k)$ is generalized to skew-submodular connectivity requirements, every extreme solution $x^*$ has an edge $e$ with $x^*_e \geq \frac{1}{2}$. Hence, complex extreme points give some idea of what properties might or might not exist that can be exploited algorithmically.

Theorem 4 significantly improves previous results in the same vein. (A long-standing conjecture that the Held-Karp relaxation $(\mathcal{N}'_2)$ has integrality gap at most 4/3 has motivated some of the work, e.g. [11,7].) Boyd and Pulleyblank [10]

[2] The integrality gap is the worst-case ratio of the integral optimum to the LP optimum.

showed that for any even $|V| \geq 10$, there is an extreme point of $(\mathcal{N}'_2)$ with fractionality $2/(|V|-4)$. Cheung [15] found extreme points of $(\mathcal{N}'_2)$ whose support graph has maximum degree $\Theta(\sqrt{|V|})$. The construction in Theorem 4 was found with the assistance of computational methods, see the full version [1] for details.

## 2 Hardness Results

In our hardness theorem for $k$-ECSS, we reduce from the following problem. (Here $\uplus$ denotes disjoint union.)

| |
|---|
| PATH-COVER-OF-TREE<br>Input: A tree $T = (V, E)$ and another set $X \subset \binom{V}{2}$ of edges/pairs.<br>Output: A subset of $Y$ of $X$ so that $(V, E \uplus Y)$ is 2-edge-connected.<br>Objective: Minimize $\lvert Y \rvert$. |

PATH-COVER-OF-TREE is sometimes called the *tree augmentation problem* and a 1.8-approximation is published [17]; as an aside, it is basically equivalent to the special case of 2-ECSS where the input graph contains a connected subgraph of cost zero, plus some unit-cost edges. We give it the alternate name PATH-COVER-OF-TREE because it is more natural for us to interpret it as covering a tree's edges with a minimum-size subcollection of a given collection of paths. To make this explicit, for an edge $x = \{u, v\} \in X$ let $P_x$ denote the edges of the unique $u$-$v$ path in $T$. We rehash the proof of the following proposition since we will recycle its methodology.

**Proposition 5** (folklore)**.** *$Y$ is feasible for* PATH-COVER-OF-TREE *if and only if* $\bigcup_{x \in Y} P_x = E$.

*Proof.* For every edge $e$ of $T$, a *fundamental cut* of $e$ and $T$ means the vertex set of either connected component of $T \backslash e$.

Let $\delta_F(U)$ denote $\delta(U)$ in the graph $(V, F)$. First, $Y$ is feasible if $|\delta_{E \uplus Y}(U)| \geq 2$ for every set $U$ with $\varnothing \neq U \subsetneq V$. But $|\delta_E(U)|$ is 1 when $U$ is a fundamental cut and at least 2 otherwise; hence $Y$ is feasible iff $|\delta_Y(U)| \geq 1$ for every fundamental cut $U$.

Second, when $U$ is a fundamental cut, say for an edge $e \in E$, $|\delta_Y(U)| \geq 1$ iff $\bigcup_{x \in Y} P_x$ contains $e$. Taking this together with the previous paragraph, we are done. □

PATH-COVER-OF-TREE is shown NP-hard in [19] and a similar construction implies APX-hardness — see the full version [1]. As an aside, it is even hard for trees of depth 2; compare this with the depth-1 instances which are in P since they can be shown isomorphic to *edge cover*. Now we prove the main hardness result:

**Theorem 1.** *Let it be* NP*-hard to approximate* PATH-COVER-OF-TREE *within ratio* $1 + \epsilon$*. Then for all integers* $k \geq 2$*, it is* NP*-hard to approximate* $k$*-ECSS within ratio* $1 + \epsilon$*, even for* 0-1 *costs.*

*Proof.* Let $(T = (V, E), X)$ denote an instance of Path-Cover-of-Tree. We construct a $k$-ECSS instance on the same vertex set, with edge set $F$. For each $e \in E$, we put $k-1$ zero-cost copies of the edge $e$ into $F$. For each $x \in X$, put one unit-cost copy of the edge $x$ into $F$. These are all the edges of $F$; and although $(V, F)$ is a multigraph, we later show that this can be avoided.

First we show the multigraph instance is hard. Clearly, there is an optimal solution for the $k$-ECSS instance which includes all copies of the 0-cost edges. Let $(k-1)E$ denote these 0-cost edges. The same logic as in the proof of Proposition 5 (analysis using fundamental cuts) shows that $Y$ is a feasible solution for the Path-Cover-of-Tree instance if and only if $(k-1)E \uplus Y$ is a feasible solution for the $k$-ECSS instance. Since costs are preserved between the two problems, it follows that an $\alpha$-approximation algorithm for $k$-ECSS would also give an $\alpha$-approximation algorithm for Path-Cover-of-Tree, and we are done.

Finally, here is how we make $(V, F)$ a simple graph: replace every vertex $v \in V$ of the tree by a $(k+1)$-clique of 0-cost edges; replace every edge $uv \in E$ of the tree by any $k-1$ zero-cost edges between the two cliques for $u$ and $v$; replace each edge $x \in X$ by any unit-cost edge between the cliques for $u$ and $v$. We proceed similarly to before: when $U$ is a vertex set of the newly constructed graph, we see $\delta(U)$ has at least $k$ 0-cost edges unless $U$ is a "blown-up" version of a fundamental cut (i.e., unless there is a fundamental cut $U_0$ of $T$ so that $U$ exactly equals the set of vertices in cliques corresponding to $U_0$). As before, the residual problem assuming these edges are bought is the same as the instance $(T, X)$ (in a cost-preserving way), so we are done. □

### 2.1 Hardness of 2-ECSM (Proof of Proposition 2)

To show that 2-ECSM is APX-hard, we prove that it is "the same" as metric 2-ECSS, i.e. the special case of 2-ECSS on complete metric graphs. Metric 2-ECSS is APX-hard by a general result of [8] (see also a sketch in [1]) and so this gives us what we want. The key observation is the following.

**Proposition 6.** *In a metric instance, given a 2-ECSM $(V, F)$, we can obtain in polynomial time a 2-ECSS $(V, F')$ with $c(F') \leq c(F)$, as long as $|V| \geq 3$.*

In other words, parallel edges can be eliminated without increasing the cost. (A similar observation in [20] turns a 2-ECSS into a 2-VCSS for metric instances.) Because the proof of Proposition 6 is relatively straightforward and not too long, we defer it to the full version [1].

*Proof of Proposition 2.* Since metric 2-ECSS is APX-hard [8], it is enough to show that any $\alpha$-approximation algorithm for 2-ECSM gives an $\alpha$-approximation for metric 2-ECSS. The metric 2-ECSS algorithm is: compute an $\alpha$-approximately-optimal 2-ECSM $F$ and apply Proposition 6 to get a 2-ECSS $F'$ with $c(F') \leq c(F)$. Using Proposition 6 a second time, and using the fact that every 2-ECSS is trivially a 2-ECSM, we see the optimal 2-ECSS and 2-ECSM values are the same. Hence $F'$ is an $\alpha$-approximately-optimal 2-ECSS, as needed. □

## 3 $k$-ECSM Conjecture and Connectivity Decomposition

Here is the conjecture made in the introduction.

**Conjecture 3.** *There is a polynomial-time approximation algorithm for $k$-ECSM which produces a solution of value at most $(1 + C/k) \cdot \mathrm{OPT}(\mathcal{N}_k)$ for some universal constant $C$.*

For positive integers $A$ and $B$, define $f(A, B)$ to be the least integer $f$ so that every $f$-edge-connected multigraph can be partitioned into two spanning subgraphs, one $A$-edge-connected and one $B$-edge-connected. Bang-Jensen and Yeo [5] ask the following question, which we call the *splitting hypothesis*: is there a constant $C$ such that $f(k, k) \leq 2k + C$ for all integers $k$? It has consequences for Conjecture 3:

**Theorem 7.** *If the splitting hypothesis holds, then every $k$-ECSM instance has a solution with cost at most $(1+C/k)\cdot\mathrm{OPT}(\mathcal{N}_k)$, i.e. the integrality gap of $(\mathcal{N}_k)$ is at most $1 + C/k$.*

This would not prove Conjecture 3 due the lack of a polynomial-time algorithm; but one might guess that once the core combinatorial problem is solved, a polynomial-time implementation could be found, as happened in [13].

Before proving Theorem 7 we make some other remarks about $f$. The lower bound $f(A, B) \geq A+B$ is very easy, and it can be raised by 1 or 2 in some cases — see details in [1]. The Nash-Williams/Tutte theorem implies $f(A, B) \leq 2(A+B)$. However, nothing other than these facts seem to be known. M. DeVos asked[3] online whether $\forall A, B : f(A, B) \leq A + B + 2$ holds, which is still open.

*Proof of Theorem 7.* Let $x^*$ be an optimal extreme point solution to $(\mathcal{N}_k)$. Since $x^*$ is rational, there is an integer $t$ such that $tx^*$ is integral. Then, it is easy to see that $tx^*$ (or more precisely, the multigraph obtained by taking $tx^*_e$ copies of each edge $e$) is a $tk$-edge-connected spanning multisubgraph. Likewise, for any positive integer $\alpha$, $\alpha t x^*$ is a $(\alpha tk)$-ECSM.

By induction, the splitting hypothesis easily gives the following.

**Claim 8.** *For all positive integers $k$ and $n$, every $(2^n(k + C) - C)$-ECSM can be decomposed into $2^n$ disjoint $k$-ECSMs.*

Now, for any integer $n$, let us pick $\alpha$ just large enough that $\alpha tk \geq (2^n(k + C) - C)$. Therefore, $\alpha t x^*$ can be decomposed into $2^n$ disjoint $k$-ECSMs. The cheapest one has cost at most

$$\frac{c(\alpha t x^*)}{2^n} = \alpha t 2^{-n} c(x^*) = \left\lceil \frac{2^n(k + C) - C}{tk} \right\rceil t 2^{-n} \mathrm{OPT}(\mathcal{N}_k).$$

Then using $\lceil \frac{2^n(k+C)-C}{tk} \rceil \leq \lceil \frac{2^n(k+C)}{tk} \rceil \leq \frac{2^n(k+C)}{tk} + 1$, we see there is a $k$-ECSM with cost at most

$$\left( \frac{2^n(k + C)}{tk} + 1 \right) t 2^{-n} \mathrm{OPT}(\mathcal{N}_k) = (1 + C/k + t/2^n)\mathrm{OPT}(\mathcal{N}_k).$$

[3] `http://garden.irmacs.sfu.ca/?q=op/partitioning_edge_connectivity`

This establishes that the integrality gap is no more than $1 + C/k + t/2^n$. Taking $n \to \infty$, we are done (since the integrality gap is some fixed real, and since $t$ doesn't depend on $n$). □

We feel strongly that the following holds.

**Conjecture 9.** $f(A, 1) = A + o(A)$.

It is not too hard to see (using repeated splitting and merging) that the splitting hypothesis would imply $f(A, 1) = A + O(C \ln A)$ and hence prove this conjecture.

Variants of $f$ have received some attention. For edge-connectivity in hypergraphs, $f(1, 1)$ is not finite [4]. It is not known whether the analogue of $f(1, 1)$ in *directed graphs* is finite [5,3].

## 4 Complex Extreme Points for $(\mathcal{N}_2')$

Now we give our construction of a new family of extreme points for the TSP subtour relaxation $(\mathcal{N}_2')$; as mentioned earlier, it can be scaled by $k/2$ to give an extreme point for $(\mathcal{N}_k')$ or $(\mathcal{N}_k)$, which is relevant to LP-based approaches for $k$-ECSM.

Let $F_i$ denote the $i$th Fibonacci number, where $F_1 = F_2 = 1$. For a parameter $t \geq 3$, we denote the extreme point by $x^*$. The construction is given in the list below and pictured in Figure 2.

- For $i$ from 1 to $t$, an edge $(2i - 1, 2i)$ of $x^*$-value 1
- For $i$ from 2 to $t - 1$, an edge $(1, 2i)$ of $x^*$-value $F_{t-i}/F_t$
- An edge $(1, 2t)$ of $x^*$-value $1/F_t$
- For $i$ from 3 to $t$, an edge $(2i - 3, 2i - 1)$ of $x^*$-value $F_{t-i+1}/F_t$
- For $i$ from 3 to $t$, an edge $(2i - 4, 2i - 1)$ of $x^*$-value $1 - F_{t-i+2}/F_t$
- An edge $(2, 3)$ of $x^*$-value $F_{t-1}/F_t$
- An edge $(2t - 2, 2t)$ of $x^*$-value $1 - 1/F_t$

The support graph of $x^*$ has $2t$ vertices and $4t - 3$ edges with fractionality $1/F_t$ and maximum degree $t$. Therefore, in order to prove Theorem 4, it suffices to show that $x^*$ is an extreme point solution.

**Proposition 10.** *The solution $x^*$ described above is an extreme point solution for $(\mathcal{N}_2')$.*

*Proof.* With foresight, we write down the following family of $4t - 3$ sets:

$$\mathcal{L} := \{\{i\}_{i=1}^{2t}, \{2i - 1, 2i\}_{i=1}^{t}, \{1, \ldots, 2i\}_{i=2}^{t-2}\}.$$

The plan of our proof is to first show that $x^*$ is the unique solution to $\{x(\delta(T)) = 2 \mid T \in \mathcal{L}\}$. It is easy to verify that $x^*$ indeed satisfies all these conditions, so let us focus on the harder task of showing that $x^*$ is the *only* solution. (Note, we are not assuming that $x^*$ is feasible, so possibly $x^*(\delta(S)) < 2$ for some other sets, but we will deal with this later.)

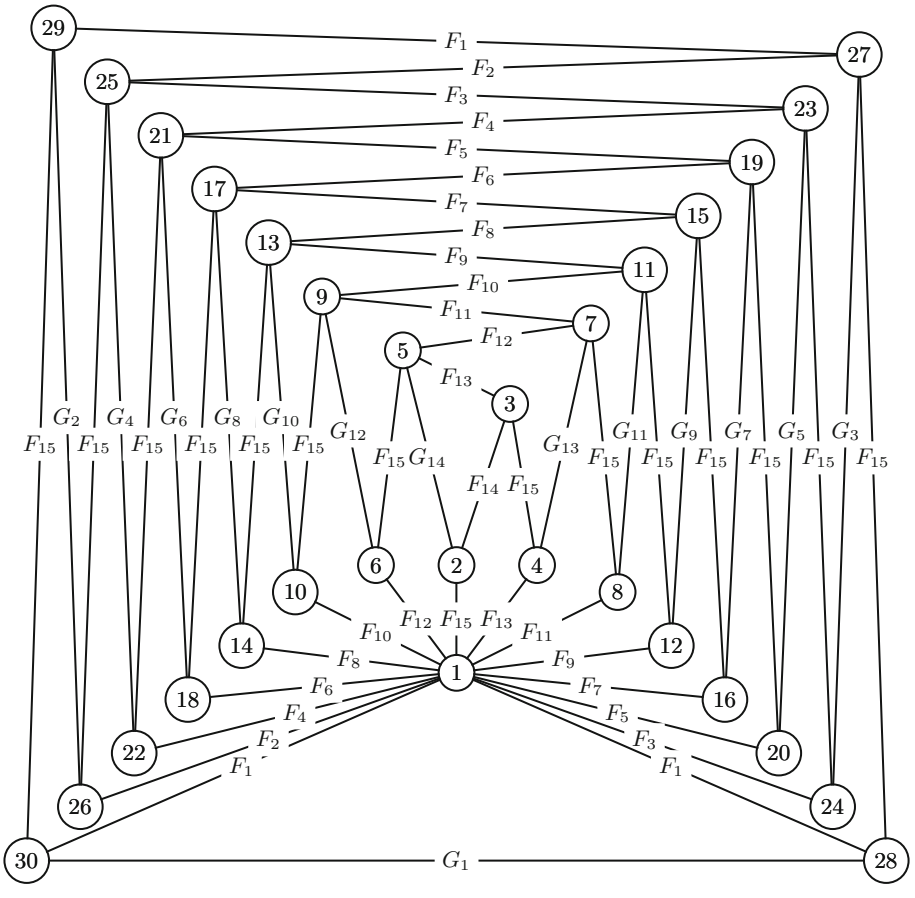



**Fig. 2.** Our new construction of a complex extreme point $x^*$ for the subtour TSP polytope ($\mathcal{N}_2'$), illustrated for $t = 15$. Scaled edge values are shown: the label $F_i$ on an edge $e$ indicates that $x_e^* = F_i/F_t$. The symbol $G_i$ denotes $F_t - F_i$, i.e. an edge $e$ with $x_e^* = 1 - (F_i/F_t)$.

A set $S$ is *tight* for a solution $x$ if $x(\delta(S)) = 2$. Consider any solution which is tight for all sets in $\mathcal{L}$. We first need a simple lemma. For disjoint sets $S, T$, let $\delta(S : T)$ denote the set of edges with one end in $S$ and the other in $T$. The short proof of the following is in [1].

**Lemma 11.** *For some solution $x$, if $S, T$ are disjoint tight sets and $S \cup T$ is also tight, then $x(\delta(S : T)) = 1$.*

Consider a hypothetical solution $x$ with $x(\delta(S)) = 2, \forall x \in \mathcal{L}$. The lemma shows all edges $\{2i-1, 2i\}_{i=1}^{t}$ have $x$-value 1 (take $S = \{2i-1\}, T = \{2i\}$). Define $y_i$ equal to $x_{(2i+1,2i+3)}$ for $i$ from 1 to $t-2$. The degree constraint at 3 (i.e., $x(\delta(3)) = 2$) forces $x_{(2,3)} = 1 - y_1$. The degree constraint at 2 forces $x_{(5,2)} = y_1$. Note $\{1, \ldots, 2t-2\}$ is tight since this set has the same constraint as $\{2t-1, 2t\}$.

For $i$ from 1 to $t-2$, note that the sets $\delta(\{1,\ldots,2i\} : \{2i+1, 2i+2\})$ and $\delta(2i+1)$ differ only in that the former contains the edge $(2i+2, 1)$ and the latter contains the edges $\{(2i+1, 2i+2), (2i+1, 2i+3)\}$. Thus, using the lemma and degree constraint at $2i+1$, we see $x_{(2i+2,1)} + x_{(2i+1,2i+3)} = y_i$. The degree constraint at $2i+2$ then forces $x_{(2i+2,2i+5)} = 1 - y_i$ for $1 \leq i \leq t-3$. The degree constraint at $2t-2$ forces $x_{(1,2t-2)} = 1 - y_{t-2}$; the degree constraint at $2t$ forces $x_{(1,2t)} = y_{t-2}$. The degree constraint at $2t-1$ forces $y_{t-2} = y_{t-3}$, and the degree constraint at $2i+5$ forces $y_i = y_{i+1} + y_{i+2}$ for $i$ from 1 to $t-4$; together this shows $y_i = F_{t-1-i} \cdot y_{t-2}$ for $i$ from $t-4$ to 1 by induction. The degree constraint at 5 forces $2y_1 + y_2 = 1$, so $(2F_{t-2} + F_{t-3})y_{t-2} = 1$ and consequently $y_{t-2} = 1/F_t$. Thus we conclude that $x = x^*$, as desired.

Now, we show $x^*$ is feasible using standard uncrossing arguments, plus the fact that $|\mathcal{L}| = 4t-3$. In $(\mathcal{N}_2')$, the constraints for sets $S$ and $V \backslash S$ are equivalent. Therefore, if we fix any root vertex $r \in V$, we may keep only the constraints for sets $S$ not containing $r$ without changing the LP. Correspondingly, we change $\mathcal{L}$ by complementing the sets that contain $r$, and it is easy to see $\mathcal{L}$ is a laminar family on $V \backslash \{r\}$. (This is along the lines of the standard argument by Cornuéjols et al. [16].) In fact $\mathcal{L}$ is a maximal laminar family, since any laminar family of nonempty subsets of $X$ contains at most $2|X| - 1$ elements, for any set $X$.

Finally, suppose for the sake of contradiction that $x^*$ is not feasible, so there is a set $S$, with $r \notin S$, having $x^*(\delta(S)) < 2$. Clearly $S \notin \mathcal{L}$. Two sets $S, T$, neither containing $r$, *cross* if all three of $S \backslash T$, $T \backslash S$, and $T \cap S$ are non-empty. Take $S$ with $x^*(\delta(S)) < 2$ such that $S$ crosses a minimal number of sets in $\mathcal{L}$. If $S$ crosses zero sets in $\mathcal{L}$, then $\mathcal{L} \cup \{S\}$ is laminar, but this is a contradiction since $S \notin \mathcal{L}$ and, crucially, $\mathcal{L}$ was maximal. Otherwise, set $S$ crosses some tight set $T \in \mathcal{L}$, then since

$$2 + 2 > x^*(\delta(S)) + x^*(\delta(T)) \geq x^*(\delta(S \cup T)) + x^*(\delta(S \cap T)),$$

either $x^*(\delta(S \cup T)) < 2$ or $x^*(\delta(S \cap T)) < 2$. It is easy to verify that both $S \cup T$ and $S \cap T$ cross fewer sets of $\mathcal{L}$ than $S$, contradicting our choice of $S$. □

### 4.1 Relation to Asymmetric TSP

Asymmetric TSP is the analogue of TSP for directed graphs: we are given a metric directed cost function on the complete digraph $(V, A)$, and seek a min-cost directed Hamiltonian cycle. Recently Asadpour et al. [2] obtained a breakthrough $O(\log n / \log \log n)$ approximation for this problem; its analysis uses the fact that extreme points of the natural LP relaxation

$$\{y \in \mathbb{R}_+^A : \forall \varnothing \neq U \subsetneq V, y(\delta^{\text{out}}(U)) \geq 1\} \qquad (\mathcal{A})$$

have denominator bounded by $2^{O(n \ln n)}$. Our undirected construction implies that for this directed variant, the extreme points attain denominator at least $2^{\Omega(n)}$; the proof is straightforward and given in the full version [1].

**Proposition 12.** *For even $n \geq 6$ there are extreme points for $(\mathcal{A})$ on $n$ vertices with fractionality $1/F_{n/2}$ or smaller (and hence denominator at least $F_{n/2}$).*

## References


1. http://arxiv.org/abs/1004.1917
2. Asadpour, A., Goemans, M., Madry, A., Gharan, S.O., Saberi, A.: An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem. In: Proc. 21st SODA (2010)
3. Bang-Jensen, J.: Problems and conjectures concerning connectivity, paths, trees and cycles in tournament-like digraphs. Discrete Mathematics 309(18), 5655–5667 (2009)
4. Bang-Jensen, J., Thomassé, S.: Highly connected hypergraphs containing no two edge-disjoint spanning connected subhypergraphs. Discrete Appl. Math. 131, 555–559 (2003)
5. Bang-Jensen, J., Yeo, A.: Decomposing $k$-arc-strong tournaments into strong spanning subdigraphs. Combinatorica 24(3), 331–349 (2004)
6. Bansal, N., Korula, N., Nagarajan, V., Srinivasan, A.: On $k$-column sparse packing programs. In: Eisenbrand, F., Shepherd, F.B. (eds.) IPCO 2010. LNCS, vol. 6080, pp. 369–382. Springer, Heidelberg (2010); preliminary version at arXiv:0908.2256
7. Benoit, G., Boyd, S.: Finding the exact integrality gap for small traveling salesman problems. Math. Oper. Res. 33(4), 921–931 (2008)
8. Böckenhauer, H.J., Bongartz, D., Hromkovič, J., Klasing, R., Proietti, G., Seibert, S., Unger, W.: On the hardness of constructing minimal 2-connected spanning subgraphs in complete graphs with sharpened triangle inequality. Theor. Comput. Sci. 326(1-3), 137–153 (2004)
9. Boyd, S., Cameron, A.: A 3/2-approximation algorithm for the multi-two-edge connected subgraph problem. Tech. Rep. TR-2008-01, SITE, University of Ottawa (2008)
10. Boyd, S.C., Pulleyblank, W.R.: Optimizing over the subtour polytope of the travelling salesman problem. Math. Program. 49, 163–187 (1991)
11. Carr, R.D., Vempala, S.: On the Held-Karp relaxation for the asymmetric and symmetric traveling salesman problems. Math. Program. 100(3), 569–587 (2004)
12. Chekuri, C., Ene, A., Korula, N.: Unsplittable flow in paths and trees and column-restricted packing integer programs. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 42–55. Springer, Heidelberg (2009)
13. Chekuri, C., Mydlarz, M., Shepherd, F.B.: Multicommodity demand flow in a tree and packing integer programs. ACM Trans. Algorithms 3(3), 27 (2007); preliminary version appeared in Proc. 30th ICALP, pp. 410–425 (2003)
14. Cheriyan, J., Thurimella, R.: Approximating minimum-size $k$-connected spanning subgraphs via matching. SIAM J. Comput. 30(2), 528–560 (2000); preliminary version appeared in Proc. 37th FOCS, pp. 292–301 (1996)
15. Cheung, K.: Subtour Elimination Polytopes and Graphs of Inscribable Type. Ph.D. thesis, University of Waterloo (2003)
16. Cornuéjols, G., Naddef, D., Fonlupt, J.: The traveling salesman problem on a graph and some related integer polyhedra. Math. Programming 33, 1–27 (1985)
17. Even, G., Feldman, J., Kortsarz, G., Nutov, Z.: A 1.8 approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. ACM Trans. Algorithms 5(2), 1–17 (2009)
18. Fernandes, C.G.: A better approximation ratio for the minimum size $k$-edge-connected spanning subgraph problem. J. Algorithms 28(1), 105–124 (1998); preliminary version appeared in Proc. 8th SODA, pp. 629–638 (1997)

19. Frederickson, G.N., JáJá, J.: Approximation algorithms for several graph augmentation problems. SIAM J. Comput. 10(2), 270–283 (1981)
20. Frederickson, G.N., JáJá, J.: On the relationship between biconnectivity augmentation and the traveling salesman problem. Theoretical Computer Science 19, 189–201 (1982)
21. Gabow, H.N., Gallagher, S.: Iterated rounding algorithms for the smallest $k$-edge-connected spanning subgraph. In: Proc. 19th SODA, pp. 550–559 (2008)
22. Gabow, H.N., Goemans, M.X., Tardos, É., Williamson, D.P.: Approximating the smallest $k$-edge connected spanning subgraph by LP-rounding. Networks 53(4), 345–357 (2009); preliminary version appeared in Proc. 16th SODA, pp. 562–571 (2005)
23. Goemans, M.X., Bertsimas, D.: Survivable networks, linear programming relaxations and the parsimonious property. Math. Programming 60, 145–166 (1993); preliminary version appeared in Proc. 1st SODA, pp. 388–396 (1990)
24. Goemans, M.X.: Minimum bounded degree spanning trees. In: Proc. 47th FOCS, pp. 273–282 (2006)
25. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21(1), 39–60 (2001); preliminary version appeared in Proc. 39th FOCS, pp. 448–457 (1998)
26. Khuller, S., Vishkin, U.: Biconnectivity approximations and graph carvings. J. ACM 41(2), 214–235 (1994); preliminary version appeared in Proc. 24th STOC, pp. 59–770 (1992)
27. Könemann, J., Parekh, O., Pritchard, D.: Multicommodity flow in trees: Packing via covering and iterated relaxation, manuscript. Preliminary version appeared in: Bampis, E., Skutella, M. (eds.) WAOA 2008. LNCS, vol. 5426, pp. 1–14. Springer, Heidelberg (2009)
28. Kortsarz, G., Krauthgamer, R., Lee, J.R.: Hardness of approximation for vertex-connectivity network design problems. SIAM J. Comput. 33(3), 704–720 (2004); preliminary version appeared in Proc. 5th APPROX, pp. 185–199 (2002)
29. Pritchard, D., Chakrabarty, D.: Approximability of sparse integer programs. Algorithmica (2010) (in press); Preliminary versions at arXiv:0904.0859 and in: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 83–94. Springer, Heidelberg (2009)
30. Singh, M., Lau, L.C.: Approximating minimum bounded degree spanning trees to within one of optimal. In: Proc. 39th STOC, pp. 661–670 (2007)
31. Wolsey, L.: Heuristic analysis, linear programming and branch and bound. Math. Programming Study 13, 121–134 (1980)

# Minimizing Maximum Flowtime of Jobs with Arbitrary Parallelizability

Kirk Pruhs[1,⋆], Julien Robert[2], and Nicolas Schabanel[3]

[1] Pittsburgh University, USA
[2] Université de Lyon (LIP - ÉNS Lyon), France
[3] CNRS - Université Paris Diderot (LIAFA), France

**Abstract.** We consider the problem of nonclairvoyantly scheduling jobs, which arrive over time and have varying sizes and degrees of parallelizability, with the objective of minimizing the maximum flow. We give essentially tight bounds on the achievable competitiveness. More specifically we show that the competitive ratio of every deterministic nonclairvoyant algorithm is high, namely $\Omega(\sqrt{n})$ for $n$ jobs. But there is a simple batching algorithm that is $(1+\varepsilon)$-processor $O(\log n)$-competitive. And this simple batching algorithm is optimally competitive as no deterministic nonclairvoyant algorithm can be $s$-processor $o(\log n)$-competitive for any constant $s$.

## 1 Introduction

The founder of chip maker Tilera asserts that a corollary to Moore's law will be that the number of cores/processors will double every 18 months [11]. In this paper we consider one of the many resulting technical challenges that arises in such a future: developing algorithms/policies for scheduling jobs on many processors so as to optimize the resulting quality of service. Such a scheduler will be faced with scheduling jobs with highly varying degrees of parallelizability, that is, when allocated many processors some jobs may be considerably sped up, while on the other extreme, some jobs may not be sped up at all.

We will consider the setting where jobs of varying sizes arrive to the system over time, and must be scheduled online on a collection of identical processors. At each point in time the online scheduler needs to partition the processors among the alive jobs. For each portion of each job, there is an inherient speed-up function that specifies the rate at which the job is processed as a function of the number of processor on which it is run. An operating system scheduler generally needs to be *nonclairvoyant*, that is, the algorithm tipically does not have access to the internal knowledge about jobs, such as the size and the speed-up functions. The standard quality of service measure for a job is its flowtime, which is the length of time between when the job arrives to the system and which it is completed. One then normally obtains a quality of service measure

⋆ Supported in part by NSF grants CNS-0325353, IIS-0534531, and CCF-0830558, and an IBM Faculty Award.

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 237–248, 2011.

for a schedule by taking the $\ell_p$ norm of the flowtimes for $1 \leqslant p \leqslant \infty$. The $\ell_p$ norm is the $p^{th}$ root of the sum of the $p^{th}$ powers of the flow times. The $\ell_1$ norm is the total, or equivalently average, flowtime, and the $\ell_\infty$ norm is the maximum flowtime. Intuitively, the higher the value of $p$, the more importance that is being placed on avoiding starvation of jobs.

Formal definitions for all concepts in the introduction can be found in Section 2.

### 1.1 Previous Results

The model of speed-up functions that we adopt here was proposed in [4]. [4] showed that the natural algorithm EQUI, which shares the processors equally among the jobs, is $O(1)$-competitive for the $\ell_1$ norm of flow in the special case that all jobs arrive at the same time. Generalizing to the case of jobs with arbitrary release times, [3] gave a quite involved proof that EQUI is $(2+\varepsilon)$-processor $O(\frac{1}{\varepsilon})$-competitive for the $\ell_1$ norm of flow. Given that a nonclairvoyant algorithm does not know the speed-up functions, it is not clear what reasonable alternative algorithms there are to EQUI, as there is no way for a nonclairvoyant algorithm to avoid the possibility that the jobs that it assigns most processors to may be the least parallelizable. However, [6] showed that, by sharing the processors evenly among most recently arriving constant fraction of the jobs, one obtains an existentially scalable algorithm (see definition p. 242), that is an algorithm that is $O(1)$-competitive with arbitrarily small processor augmentation. [6] also gives a much simpler proof of the competitiveness of EQUI proved in [3].

The model was then extended in [16] to include arbitrary precedence constraints among tasks within each job. [16] showed that the introduction of precedence constraints does not affect the minimum processor augmentation required to be competitive for the $\ell_1$ norm of flow, even if the resulting competitive ratio depends on the internal dependencies of each job.

The $\ell_p$ norm of flow, for $1 < p < \infty$ was recently considered in [7]. [7] showed that a simple algorithm that allocates the processors to the most recent alive jobs proportional to the $(p-1)^{\text{st}}$ power of their age is $(2+\varepsilon)$-processor $O(1)$-competitive. Very recently it was shown that this algorithm is existentially scalable [5].

The only previous work on the $\ell_\infty$ norm of flow was in [15]. [15] showed that if all the jobs are released all together at time 0, EQUI is $O(\log n / \log\log n)$-competitive, and there is a matching general lower bound even allowing constant factor processor augmentation.

*On single processor nonclairvoyant scheduling, or equivalently for multiprocessor scheduling when all the work is parallel,* there has been a fair amount of work done to optimize $\ell_p$ norms of flow. Let us first consider the $\ell_1$ norm. The competitive ratio of every deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized nonclairvoyant algorithm against an oblivious adversary is $\Omega(\log n)$ [12]. There is a randomized algorithm, Randomized Multi-Level Feedback Queues, that is $O(\log n)$-competitive against an oblivious adversary [10,2]. The algorithm Shortest Elapsed Time First, which

shares the processors equally among the jobs that have been processed the least to date, is universally scalable [9]. For the $\ell_p$ norm of flows for $1 < p < \infty$, the competitive ratio of every randomized nonclairvoyant algorithm is $\Omega(n^{(p-1)/3p^2})$, and Shortest Elapsed Time First is universally scalable [1]. The nonclairvoyant algorithm First Come First Served is optimal for maximum flow.

There are many related scheduling problems with other objectives, and/or other assumptions about the machine and job instance. Surveys can be found in [14,13].

### 1.2 Our Results

So essentially what competitiveness is achievable by a nonclairvoyant algorithm for the $\ell_p$ norm of flow is known for finite $p$. In this paper we address the obvious remaining open question: What competitiveness is achievable for the case that $p = \infty$, that is for the objective of maximum flow. We give the following essentially tight results:

- In section 3 we show that the competitive ratio of every deterministic nonclairvoyant algorithm is high, namely $\Omega(\sqrt{n})$.
- In section 4 we show that there is a simple nonclairvoyant batching algorithm OBEQUI that is $(1 + \varepsilon)$-processor $O(\log n)$-competitive. In OBEQUI there are always two active batches, the current batch and the next batch. The processors are shared equally among the current batch. Newly arriving jobs are added to the next batch. When all the jobs in the current batch finish, the next batch becomes the current batch.
- In section 5 we show that this simple batching algorithm is optimally competitive as no deterministic nonclairvoyant algorithm can be $s$-processor $o(\log n)$-competitive for any constant $s$.
- In section 6 we show that the techniques developed in [16] to handle precedence constraints when the objective is the $\ell_1$ norm of flow can be extended to the $\ell_\infty$ norm of flow. Furthermore, we give here a modular presentation of these reduction-based techniques that will allow easy applications of the concepts in [16] to arbitrary non-clairvoyant setting.

We find it surprising that such a simple batching strategy is optimal, it was far from the first algorithm that we tried to analyze. Given the competitiveness results for nonclairvoyantly scheduling jobs with the objective of the $\ell_p$ norm of flow on a single and on multiple processors we make the following observations:

- On a single processor the $\ell_\infty$ norm is the easiest objective for the nonclairvoyant scheduler as First Come First Served produces an optimal schedule, and the best that a nonclairvoyant scheduler can do for the other norms is to be universally scalable. Of course scheduling on multiple processors is harder for a nonclairvoyant scheduler. But on mutiple processors, if jobs can have arbitrary parallelizability, then the $\ell_\infty$ norm is the hardest objective for the nonclairvoyant scheduler as it is the one objective where it is not possible for the scheduler to be at least existentially scalable. This suggests that perhaps

starvation avoidance is a more difficult objective in a multiprocessor setting than in a single processor setting.
- By adding release times, the optimal competitive ratio increases significantly, from $\Theta(\log n/\log\log n)$ to $\Omega(\sqrt{n})$. But adding release dates only raises the competitiveness achievable by a nonclairvoyant algorithm with $(1+\varepsilon)$-processor augmentation from $\Theta(\log n/\log\log n)$ to $\Theta(\log n)$, and a larger constant factor processor augmentation doesn't improve the competitiveness achievable by a nonclairvoyant algorithm.

## 2 Definitions and Notation

We present here the model in its general form with arbitrary speed-up functions and precedences constraints. It turns out that our result proceeds by reduction to a much simpler setting, including only parallel and sequential speed-up functions and with no precedences constraint, which we will present first. Thus some of these definitions will not be needed before section 6.

*The Setting.* We consider a sequence of jobs $\{J_1, J_2, \ldots\}$ with release times $\{r_1, r_2, \ldots\}$. Following the terminology of [8,16], each job $J_i$ consists in a set of tasks $\{J_{i,1}, \ldots, J_{i,m_i}\}$ with precedence constraints that the scheduler has to execute over $p$ processors. Each task goes through different phases, where each phase may have a different speed-up function. The scheduler has to decide online the number of processors to allocate to each alive task. The scheduler is *nonclairvoyant*, *i.e.*, discovers the jobs at the time of their arrivals and the tasks at the time they become available; furthermore, it is unaware of the current speed-up of each task (*i.e.*, how they take advantage of more processing power) nor of the amount of work in each task; it is only informed that a task or a job is completed at the time of its completion. As in [3,16,6], we consider that the processors can be divided fractionally: fractional allocation is usually realized through time multiplexing in real systems.

*Schedules.* A *schedule* $\mathcal{S}_p$ on $p$ processors is a set of piecewise constant functions[1] $\rho_{ij} : t \mapsto \rho_{ij}^t$ where $\rho_{ij}^t$ is the *amount of processors* allotted to the task $J_{ij}$ at time $t$; $(\rho_{ij}^t)$ are arbitrary non-negative real numbers, such that at any time $t$: $\sum_{ij} \rho_{ij}^t \leqslant p$.

*The jobs.* The dependencies are defined as in [16] (extending the definition of [3,4]): each job $J_i$ consists of a directed acyclic graph (*DAG* for short) $(\{J_{i\,1}, \ldots, J_{i\,m_i}\}, \prec)$, where task $J_{ij}$ is released as soon as all tasks $J_{ik}$, such that $J_{ik} \prec J_{ij}$, are completed. Job $J_i$ is completed as soon as all its tasks are completed. Each task goes through a sequence of phases $J_{ij}^1, \ldots, J_{ij}^{q_{ij}}$ with different degrees of parallelism. Each phase $J_{ij}^k$ consists in an amount of work $w_{ij}^k$ and

[1] Requiring the functions $(\rho_{ij})$ to be piecewise constant is not restrictive since any finite set of reasonable (*i.e.*, Riemann integrable) functions can be uniformly approximated from below within an arbitrary precision by piecewise constant functions. In particular, all of our results hold if $\rho_{ij}$ are piecewise continuous functions.

a *speed-up function* $\Gamma_{ij}^k$. At time t, during its $k$-th phase, each task $J_{ij}$ progresses at a *rate* $\Gamma_{ij}^k(\rho_{ij}^t)$ which depends on the amount $\rho_{ij}^t$ of processors allotted to $J_{ij}$ by the scheduler, *i.e.*, the amount of work accomplished between $t$ and $t+dt$ in each task $J_{ij}$ during its $k$-th phase is: $dw = \Gamma_{ij}^k(\rho_{ij}^t)dt$.

Given a schedule $\mathcal{S}_p$ of the jobs $\{J_1, J_2, \ldots\}$. A job or a task is *alive* as soon as it is released and until it is completed. Let $c_{ij}$ denote the *completion time* of task $J_{ij}$. The *release time* $r_{ij}$ of a task $J_{ij}$ is: $r_{ij} = r_i$ (the release time of Job $J_i$) if $J_{i,j}$ does not depend on any other task (*i.e.*, if $J_{ik} \not\prec J_{ij}$ for all $k$); and $r_{ij} = \max\{c_{ik} : J_{ik} \prec J_{ij}\}$, otherwise. Let $c_{ij}^k$ denote the completion time of the $k$-th phase of task $J_{ij}$: $c_{ij}^k$ is the first time $t'$ such that $w_{ij}^k = \int_{c_{ij}^{k-1}}^{t'} \Gamma_{ij}^k(\rho_{ij}^t)\, dt$ (with $c_{ij}^0 = r_{ij}$). Each task $J_{ij}$ completes with its last phase, thus: $c_{ij} = c_{ij}^{q_{ij}}$. Job $J_i$ is thus completed at time $c_i = \max_j c_{ij}$. A schedule is *valid* if all jobs eventually complete, *i.e.*, if $c_i < \infty$ for all $i$.

*Cost of a schedule.* The *flowtime* $F_i$ of a job $J_i$ is the overall time $J_i$ is alive in the system, *i.e.*, $F_i = c_i - r_i$. The *maximum flowtime* of a schedule $\mathcal{S}_p$ is the maximum of the flowtimes of the jobs: $\text{MaxFlowTime}(\mathcal{S}_p) = \max_i F_i$. Our goal is to design a scheduler that minimizes the maximum flowtime, which corresponds to the largest response time of the system, which is a classic measure of quality of service. We denote by $\text{OPT}_p = \inf\{\text{MaxFlowTime}(\mathcal{S}_p) : \text{valid schedule } \mathcal{S}_p\}$, the optimal cost on $p$ processors.

*Speed-up functions.* As in [4,16,6], we assume that each speed-up function is *non-decreasing* and *sub-linear* (*i.e.*, such that for all $i, j, k$, $\rho < \rho' \Rightarrow \frac{\Gamma_{ij}^k(\rho)}{\rho} \geqslant \frac{\Gamma_{ij}^k(\rho')}{\rho'}$). Non-decreasing means that allocating more processors to a job will not slow the processing of that job, and sub-linear means that efficiency decreases as the number of processors increase. As in [4,16,6], two types of speed-up functions will be of particular interest here:

- the *sequential* phases ($\mathsf{Seq}$) where $\Gamma(\rho) = 1$, for all $\rho \geqslant 0$ (the task progresses at a constant rate even if *no* processor is allotted to it, similarly to an idle period); and
- the *parallel* phases ($\mathsf{Par}$) where $\Gamma(\rho) = \rho$, for all $\rho \geqslant 0$.

We say that a job $J_i$ is $\mathsf{SeqPar}$ if each of the phases of its tasks is either sequential or parallel. An instance is $\mathsf{SeqPar}$ if all of its jobs are $\mathsf{SeqPar}$. For any task $J_{ij}$ of a $\mathsf{SeqPar}$ job $J_i$, we define $\mathsf{Seq}(J_{ij})$ and $\mathsf{Par}(J_{ij})$ as the overall sequential and parallel works in the task respectively:

$$\mathsf{Seq}(J_{ij}) = \sum_{k\text{: } k\text{th phase of } J_{ij} \text{ is sequential}} w_{ij}^k \quad \text{and} \quad \mathsf{Par}(J_{ij}) = \sum_{k\text{: } k\text{th phase of } J_{ij} \text{ is fully parallel}} w_{ij}^k.$$

We denote by $\mathsf{Par}(J_i) = \sum_{J_{ij} \in J_i} \mathsf{Par}(J_{ij})$ the total amount of parallel work in the tasks of a $\mathsf{SeqPar}$ job $J_i$. We denote by $\mathsf{Seq}(J_i) = \max_{J_{ij_1} \prec \cdots \prec J_{ij_k}} \sum_{\ell=1}^{k} \mathsf{Seq}(J_{ij_\ell})$, the maximum amount of sequential work along a chain of tasks in job $J_i$. We denote by $\mathsf{Par}(J) = \max_i \mathsf{Par}(J_i)$ and $\mathsf{Seq}(J) = \max_j \mathsf{Seq}(J_i)$ for any set of jobs $J = \{J_1, \ldots, J_n\}$.

**Lemma 1 (Trivial lower bound, [3,16]).** *For any* SeqPar *instance* $J_1, \ldots, J_n$*, we have:* $\mathrm{OPT}_1(J) \geqslant \max_i(\mathsf{Par}(J_i), \mathsf{Seq}(J_i))$.

*Non-clairvoyant scheduling with precedence constraints.* As in [4,16,6], we consider that the scheduler knows nothing about the progress of each tasks and is only informed that a job or a task is completed *at the time of its completion*; in particular, it is not aware of the different phases that each task goes through (neither of the amount of work nor of the speed-up function). Furthermore, tasks are released as soon as they become available without noticing the scheduler of the precedence constraints: if two tasks complete as other tasks are released, the scheduler is unable to guess which spawns which. In particular, as in [16], the order in which the tasks of a given job are released depends heavily on the computed schedule, and the scheduler cannot even reconstruct the DAG *a posteriori* in general. It is only aware at all time of the IDs of the current alive jobs and of their alive tasks.

*Competitiveness and resource augmentation.* We say that a given scheduler $A_p$ is *c-competitive* if it computes a schedule $A_p(S)$ whose maximum flowtime is at most $c$ times the optimal *clairvoyant* maximum flowtime (that is aware of the characteristics of the phases of each task and of the DAG of each job), *i.e.*, such that $\mathrm{MaxFlowTime}(A_p(J)) \leqslant c \cdot \mathrm{OPT}_p(J)$ for all instances $J$. A scheduler $A_p$ is *s-processor c-competitive* if it computes a schedule $A_{sp}(J)$ on $sp$ processors whose maximum flowtime is at most $c$ times the optimal maximum flowtime on $p$ processors only, *i.e.*, such that $\mathrm{MaxFlowTime}(A_{sp}(J)) \leqslant c \cdot \mathrm{OPT}_p(J)$ for all instances $J$ [9]. A scheduler $A$ is *universally scalable* if for every $\varepsilon > 0$, there is a constant $c_\varepsilon$ such $A_p$ is $(1+\varepsilon)$-processor $c_\varepsilon$-competitive. A family of algorithms $A_{p,\varepsilon}$ is *existentially scalable* if for every $\varepsilon > 0$, there is a constant $c_\varepsilon$ such algorithm $A_{p,\varepsilon}$ is $(1+\varepsilon)$-processor $c_\varepsilon$-competitive.

*Par→Seq instances.* A special case of SeqPar job will be of particular interest here. A job is said to be Par→Seq if it consists in one single task consisting of only two phases: *one single parallel phase followed by one final sequential phase.* An instance is said to be Par→Seq if all its jobs are Par→Seq. As we will be show in Section 6, proving the competitiveness of our algorithm on Par→Seq instances (proved in Section 4) will be enough to conclude its competitiveness on instances with arbitrary speed-up functions and precedence constraints.

## 3 The Lower Bound on the Competitive Ratio

**Theorem 2.** *There is no deterministic non-clairvoyant* 1*-processor c-competitive algorithm for any* $c < \sqrt{n}/4$ *for maximum flowtime even in the case that each consists of a single* SeqPar *task.*

*Proof.* Consider a deterministic non-clairvoyant algorithm $A$ on 1 processor. Let $n$ be the square of an even integer: $n = 4m^2$. The adversary releases 2 jobs $J_i$ and $J_i'$ at each time $t = i \in \{0, 1, \ldots, n/2-1\}$. Each job is composed of a parallel

phase followed by a sequential phase of length 1. The amount of parallel work in each job is determined on-the-fly by the adversary according to the schedule computed by $A$ so far (since $A$ is non-clairvoyant, the adversary can set the phases afterwards). The total amount of parallel work within each pair of jobs $J_i, J_i'$ will always be equal to 1. This unit of parallel work is split between $J_i$ and $J_i'$ as follows. Consider each time slot $[t, t+1]$ with $t \in \{i, i+1, \ldots\}$. Both jobs remain in a parallel phase as long as $A$ allots at most $1/\sqrt{n}$ processors to each of them during each time slot $[t, t+1]$ and $t \leqslant i + \sqrt{n}/2 - 1$. Then, either we reach $t = i + \sqrt{n}/2$ and then the adversary sets the amount of parallel work in each job to $\frac{1}{2}$; since none of the jobs is allotted more than $1/\sqrt{n}$ processors on average, none of their parallel phases can be completed at time $i + \sqrt{n}/2$ and the instance is correctly defined. Otherwise, one the of job, say $J_i$, is allotted more than $1/\sqrt{n}$ processors during time slot $[t, t+1]$. Then, the adversary sets the amount of parallel work in $J_i$ and $J_i'$ to the total amount of processors each received from $A$ between $i$ and $t$ (which sums to some $w < 1$), and gives the remaining parallel work $1 - w$ to $J_i'$.

The optimum can complete both parallel phases in each pair of jobs during the time unit after their release and thus guarantees a maximum flowtime of 2 for each job.

We claim that $\text{MaxFlowTime}(A) \geqslant \sqrt{n}/2$. Indeed, either there is one pair of jobs $J_i, J_i'$ that were never allotted more than $1/\sqrt{n}$ processors each in each time slot $[t, t+1]$ for $t \in \{i, \ldots, i + \sqrt{n}/2 - 1\}$. Then, both of their parallel phases could not be completed at time $t = i + \sqrt{n}/2 - 1$ and their flowtime is $\geqslant \sqrt{n}/2$. Otherwise, one job in each pair was allotted at least $1/\sqrt{n}$ processors for its final sequential phase. Let $n + T$ be the completion time of the last completed job. $n + T$ has to be at least $n/2 \cdot 1/\sqrt{n} + n$, the total amount of processors wasted on sequential phases plus the total amount, n, of parallel work in the instance. It follows that $\text{MaxFlowTime}(A) \geqslant T \geqslant \sqrt{n}/2 \geqslant \sqrt{n}/4 \cdot \text{OPT}$.

## 4 Analysis of the Batching Algorithm on Par→Seq Instances

We consider here only Par→Seq instances. Section 6 will show that one can reduce the general case to this simpler case. Recall our batching algorithm for job without precedence constraints, named OBEQUI (for Online Batching EQUI): it maintains at all time two batches; at the beginning, one batch contains the first released jobs, and the other one is empty; then repetitively, the jobs contained in the older batch are all scheduled together using the EQUI algorithm (each alive job in the batch receives an equal share of the processors) until all of them completes, while OBEQUI collects all the jobs released in between in the other batch; OBEQUI then switches the batches and restarts. We denote by $\mathcal{B}_k$ the $k$th batch of jobs scheduled by OBEQUI. This section is dedicated to proving the following theorem.

**Theorem 3.** *The simple non-clairvoyant batching algorithm* OBEQUI *is* $(1+\varepsilon)$*-processor* $O(\frac{\log n}{\varepsilon^2})$*-competitive for maximum flowtime on* $\mathsf{Par}\rightarrow\mathsf{Seq}$ *instances, for all* $\varepsilon > 0$.

First, we give a lower bound on the optimal cost that we will use extensively.

*Excess.* The *maximum parallel work in excess* of a SeqPar instance $\{J_1, \ldots, J_n\}$ is defined as:

$$\mathsf{Exc}(J) = \min\big\{W : \forall(t \leqslant t') \textstyle\sum_{r_i \in [t,t']} \mathsf{Par}(J_i) \leqslant t' - t + W\big\},$$

*i.e.* the maximum quantity of parallel work received during any time interval that cannot be scheduled on 1 processor within this interval. By minimality of Exc, consider a time interval $[t, t']$ such that $\sum_{r_i \in [t,t']} \mathsf{Par}(J_i) = t' - t + \mathsf{Exc}(J)$, no schedule can complete the parallel work received during $[t, t']$ before time $t' + \mathsf{Exc}(J)$, it follows that the flowtime of some job released in $[t, t']$ is at least $\mathsf{Exc}(J)$. Together with Lemma 1, we obtain the following lower bound that we will use to analyze our algorithms.

**Lemma 4 (Lower bound).** *For all SeqPar instance* $J_1, \ldots, J_n$,
$$\mathrm{OPT}_1(J) \geqslant \max(\mathit{Exc}(J), \mathit{Seq}(J)).$$

We use the following notations: $\gamma_k$ denotes the completion time of the $k$th batch, $\mathcal{B}_k$, ($\gamma_0 = -\infty$ by convention); $\rho_k$ denotes the release time of the first job in Batch $\mathcal{B}_k$; $n_k$ denotes the number of jobs in Batch $\mathcal{B}_k$. Note that for all $k \geqslant 2$, $\max(\rho_{k-1}, \gamma_{k-2}) < \rho_k \leqslant \gamma_k$. Note also that the processing of batch $\mathcal{B}_k$ begins exactly at time $\beta_k =_{\text{def}} \max(\rho_k, \gamma_{k-1})$.

Let $T(n, \varepsilon) = \big(\frac{8}{\varepsilon} + \frac{32}{\varepsilon^2}\big) \log n \cdot \max(\mathsf{Exc}(J), \mathsf{Seq}(J))$.

The main idea of the analysis is to show that if the previous batch lasts at most $T(n, \varepsilon)$, then the next one will be completed in time at most $T(n, \varepsilon)$ as well. The batch $\mathcal{B}_k$ contains all the jobs released between time $\rho_k$ and time $\beta_k = \max(\rho_k, \gamma_{k-1})$. Its processing starts at time $\beta_k$ and ends at time $\gamma_k$. We will show the following:

**Lemma 5.** *For all* $\tau \geqslant T(n, \varepsilon)$*, if* $\beta_k - \rho_k \leqslant \tau$ *then* $\gamma_k - \beta_k \leqslant \tau$.

*Proof.* Since all the jobs in $\mathcal{B}_k$ are released in $[\rho_k, \beta_k]$, the total amount of parallel work of this batch is at most $\beta_k - \rho_k + \mathsf{Exc}(J)$ by definition of Exc. Assume that OBEQUI is run on $1 + \varepsilon$ processors with $\varepsilon \leqslant \frac{1}{2}$ (this assumption is not necessary but simplifies the calculations bellow). We now follow the lines of [15]. We partition the processing period of the batch $\mathcal{B}_k$ in two sets: $A$ is the set of all instant $t \in [\beta_k, \gamma_k]$ such that a fraction at least $1 - \frac{\varepsilon}{8}$ of the alive jobs of $\mathcal{B}_k$ are in a parallel phase; $\bar{A}$ is its complementary set, *i.e.* the set of all instant $t \in [\beta_k, \gamma_k]$ such that a fraction more than $\frac{\varepsilon}{8}$ of the active jobs of $\mathcal{B}_k$ are in their final sequential phase. By construction,

$$\gamma_k - \beta_k = \textstyle\int_A dt + \int_{\bar{A}} dt.$$

At each instant $t \in A$, the amount of parallel work decreases at a rate at least $(1+\varepsilon)(1-\frac{\varepsilon}{8})$. It follows that:

$$\int_A dt \leqslant \frac{\mathsf{Par}(\mathcal{B}_k)}{(1+\varepsilon)(1-\frac{\varepsilon}{8})} \leqslant (\tau + \mathsf{Exc}(J))(1-\tfrac{\varepsilon}{4}),$$

since $\varepsilon \leqslant \frac{1}{2}$. Let $\mathsf{Seq}(\mathcal{B}_k) = \max_{J_i \in \mathcal{B}_k} \mathsf{Seq}(J_i)$. Clearly, $\mathsf{Seq}(\mathcal{B}_k) \leqslant \mathsf{Seq}(J)$. As in [15], we cover $\bar{A}$ with a minimum number $q$ of non-overlapping intervals of length $\mathsf{Seq}(\mathcal{B}_k)$. At the beginning of each of these $q$ intervals, a fraction larger than $\frac{\varepsilon}{8}$ of the alive jobs in $\mathcal{B}_k$ are in their final sequential phase and will then be completed at the end of the interval. It follows that the number of alive jobs decreases by at least a factor $1-\frac{\varepsilon}{8}$ after each of these intervals. Thus, $q \leqslant -\log_{(1-\frac{\varepsilon}{8})} n_k$. Then

$$\int_{\bar{A}} dt \leqslant q \cdot \mathsf{Seq}(\mathcal{B}_k) \leqslant -\frac{\log n_k}{\log(1-\frac{\varepsilon}{8})}\mathsf{Seq}(J) \leqslant \frac{8}{\varepsilon} \cdot \log n \cdot \mathsf{Seq}(J).$$

It follows that:

$$\gamma_k - \beta_k \leqslant \left(1-\tfrac{\varepsilon}{4}\right)(\tau + \mathsf{Exc}(J)) + \tfrac{8}{\varepsilon} \cdot \log n \cdot \mathsf{Seq}(J).$$

We are now left with proving that:

$$\left(1-\tfrac{\varepsilon}{4}\right)(\tau + \max(\mathsf{Exc}(J), \mathsf{Seq}(J))) + \tfrac{8}{\varepsilon} \cdot \log n \cdot \max(\mathsf{Exc}(J), \mathsf{Seq}(J)) \leqslant \tau$$

whenever $\tau \geqslant T(n,\varepsilon) = \left(\frac{8}{\varepsilon} + \frac{32}{\varepsilon^2}\right) \log n \cdot \max(\mathsf{Exc}(J), \mathsf{Seq}(J))$. This holds since by subtracting $\tau$ from both sides of this inequation, we get:

$$\begin{aligned}
&-\tfrac{\varepsilon}{4}\tau + \left(1 - \tfrac{\varepsilon}{4} + \tfrac{8\log n}{\varepsilon}\right) \max(\mathsf{Exc}(J), \mathsf{Seq}(J)) \\
&\leqslant \left(-\tfrac{\varepsilon}{4}\left(\tfrac{8}{\varepsilon} + \tfrac{32}{\varepsilon^2}\right) \log n + 1 - \tfrac{\varepsilon}{4} + \tfrac{8}{\varepsilon}\log n\right) \max(\mathsf{Exc}(J), \mathsf{Seq}(J)) \\
&\leqslant 0. \qquad \square
\end{aligned}$$

By immediate induction, the flowtime of every job is at most $2T(n,\varepsilon)$: $T(n,\varepsilon)$ for waiting to be scheduled, plus $T(n,\varepsilon)$ for its batch to be completed. Thus, Theorem 3 follows by the lower bound for $OPT$ given in Lemma 4.

## 5 The General Lower Bound

This section will be devoted to proving the following theorem.

**Theorem 6.** *For all $\varepsilon > 0$, there is no deterministic non-clairvoyant $(1+\varepsilon)$-processor $c$-competitive algorithm for $c < \frac{1}{2} \cdot \log n$. This holds even if instances are restricted such that each job consists of a single $\mathsf{Par} \rightarrow \mathsf{Seq}$ task.*

Consider a deterministic non-clairvoyant algorithm $A$ on $1+\varepsilon$ processors. Let $n = b \cdot m$ be the product of two integers such that: $m \sim n^{1-1/\sqrt{\log n}}$ and $b \sim n^{1/\sqrt{\log n}} = e^{\sqrt{\log n}}$. Let $F = \log n$. The adversary releases $m$ jobs $J_1^i, \ldots, J_m^i$ at each integer time $t = i \in \{0, 1, \ldots, b-1\}$; the set $J_1^i, \ldots, J_m^i$ is referred as the

$i$th batch with $0 \leqslant i < b$. Each job is composed of a parallel phase followed by a sequential phase of length 1. The adversary will ensure that the total amount of parallel work in each batch is at most 1, so that the optimum can schedule all the parallel work between $t$ and $t+1$ on one processor and thus complete every job within 2 time units, i.e. $\mathrm{OPT}_1 \leqslant 2$.

The adversary sets the parallel work of the jobs in each batch as follows. Let $j_t^i$ denote the number of alive jobs in the $i$-th batch at time $i+t$ ($j_0^i = m$). At each time $i+t$, with $t \in \{1, \ldots, F\}$, and as long as $j_{t-1}^i > 0$, the adversary sorts the $j_{t-1}^i$ surviving jobs of the batch by non-decreasing average number of processors allotted by $A$ during $[i+t-1, i+t]$. The adversary selects the maximum $k$ such that the amount of processors allotted by $A$ to the $k$ first jobs in that order is at most $1/F$. The adversary sets these $k$ first jobs in a parallel phase during $[i+t-1, i+t]$ and sets the $j_t^i - k$ others in their final sequential phase (they are thus completed at time $t$). Note that after that $j_t^i = k$. All the surviving jobs (if any) are forced to enter their final sequential phase at time $t = i+F$.

Note that at most $1/F$ parallel work is injected into the jobs of the batch in each time slot; since the lifetime of each batch is at most $F$, each batch contains at most one unit of parallel work in total, which ensures that for this instance $\mathrm{OPT}_1 \leqslant 2$ as claimed earlier. We will now prove that $\mathrm{MaxFlowTime}(A) = \Omega(\log n)$.

Let $s_t^i$ denote the total amount of processors allotted by $A$ during $[i+t, i+t+1]$ to the surviving jobs of the $i$-th batch.

**Lemma 7.** *For all $i$ and $t < F$ such that $j_t^i > 0$, we have: $j_{t+1}^i = j_t^i$ if $s_t^i \leqslant \frac{1}{F}$; and $j_{t+1}^i > \frac{j_t^i}{F \cdot s_t^i} - 1$, otherwise.*

*Proof.* During $[i+t, i+t+1]$, if $s_t^i \leqslant \frac{1}{F}$, all the jobs are set in a parallel phase and are still alive at time $i+t+1$, thus $j_{t+1}^i = j_t^i$. Otherwise, each alive job in the $i$-th batch is allotted on average $s_t^i / j_t^i$ processors. It follows, by the maximality of $k$, that $(k+1)s_t^i / j_t^i > \frac{1}{F}$ and thus $j_{t+1}^i = k > \frac{j_t^i}{F \cdot s_t^i} - 1$.

Simple algebraic manipulation yields the following corollary:

**Corollary 8.** *For all $i$ and $t < F$ such that $j_t^i \geqslant (1+\varepsilon)F^3$ and $s_t^i > \frac{1}{F}$, we have:*

$$j_{t+1}^i > \left(1 - \tfrac{1}{F^2}\right) \cdot \tfrac{j_t^i}{F \cdot s_t^i}.$$

Let $\Delta^i = \{t : 0 \leqslant t < F \text{ and } j_t^i \geqslant (1+\varepsilon)F^3 \text{ and } s_t^i > \frac{1}{F}\}$ denote the set of the time slots in the lifetime of the $i$th batch, where it receives from $A$ at least $\frac{1}{F}$ processors, and where the number of alive jobs is at least $(1+\varepsilon)F^3$, so that the lower bound of the corollary above applies. Let $T^i = \#\Delta^i$ denote the size of $\Delta^i$ and $t_+^i = \max \Delta^i$. Note that the maximum flowtime of the jobs in the $i$-th batch is at least $T^i$.

Note that as long as $j_t^i \geqslant (1+\varepsilon)F^3$, we have $j_{t+1}^i < j_t^i$ only for $t \in \Delta^i$, and $j_{t+1}^i = j_t^i$ otherwise. If the lifetime of the $i$th batch is $F$, then the maximum flow time of $A$ is $F$ and we are done. Let us now assume that the flowtime of all

batches is less than $F$. It follows that for all i, $j^i_t < (1+\varepsilon)F^3$ for some $t < F$, in particular: $\Delta^i \neq \varnothing$, $t^i_+ \geqslant T^i > 0$, and $j^i_{1+t^i_+} < (1+\varepsilon)F^3$ (indeed, since $j^i_{t+1} < j^i_t$ only for $t \in \Delta^i$ as long as $j^i_t \geqslant (1+\varepsilon)F^3$, this threshold is crossed exactly between $t^i_+$ and $1+t^i_+$).

Let $\overline{s}^i = \frac{1}{T^i}\sum_{t\in\Delta^i} s^i_t$ denote the average amount of processors allotted to the batch during the time slots in $\Delta^i$. Note that $\overline{s}^i > \frac{1}{F}$ by construction. By Corollary 8,

$$(1+\varepsilon)F^3 > j^i_{1+t^i_+} > j^i_0 \prod_{t\in\Delta^i} \tfrac{1-1/F^2}{F\cdot s^i_t} \geqslant \frac{m}{\left(\frac{F\cdot\overline{s}^i}{1-1/F^2}\right)^{T^i}},$$

by log-concavity of the product. Now, by taking the log of both ends of the inequality above,

$$T^i > \frac{\log m - \log((1+\varepsilon)F^3)}{\log\left(\frac{F\cdot\overline{s}^i}{1-1/F^2}\right)}, \tag{1}$$

since $\overline{s}^i > \frac{1}{F}$ and thus $\log\left(\frac{F\cdot\overline{s}^i}{1-1/F^2}\right) > 0$.

To get the $\Omega(\log n)$ lower bound on some $T^i$, it suffices now to show that some batch gets a small enough average amount of processors $\overline{s}^i$ compared to $1/F$.

**Lemma 9.** *For large enough $n$, there exists a batch $i$ such that $\overline{s}^i \leqslant \dfrac{e}{F}$*

*Proof.* Consider some constant $K$ to be chosen later. We proceed by contradiction and assume that for all batches, $\overline{s}^i > K/F$. Since the lifetime of every of the $b$ batches is at most $F$ by construction, the total amount of processors used by Algorithm $A$ is at most $(b+F)(1+\varepsilon)$. But, each batch $i$ receives $\overline{s}^i$ processors on average during $T^i$ time. It follows that:

$$(1+\varepsilon)(b+F) \geqslant \sum_{i=0}^{b-1} T^i\cdot\overline{s}^i \geqslant \sum_{i=0}^{b-1} \frac{\overline{s}^i}{\log\left(\frac{F\cdot\overline{s}^i}{1-1/F^2}\right)}\cdot(\log m - \log((1+\varepsilon)F^3)), \quad \text{by (1).}$$

But $s \mapsto s/\log(a\cdot s)$ is increasing for $s \geqslant e/a$. Assume $K \geqslant e\cdot(1-1/F^2)$. Now $\overline{s}^i > K/F$ for all $i$ by hypothesis, so:

$$\begin{aligned}(1+\varepsilon)\cdot n^{1/\sqrt{\log n}} \sim (1+\varepsilon)(b+F) &\geqslant b\cdot\frac{K/F}{\log(F\cdot(K/F)\,/\,(1-1/F^2))}\cdot(\log m - \log((1+\varepsilon)F^3))\\ &= b\cdot\frac{K}{\log(K/(1-1/F^2))}\cdot\frac{(1-\frac{1}{\sqrt{\log n}})\log n - O(\log\log n)}{\frac{1}{1+\varepsilon}\log n}\\ &\sim (1+\varepsilon)\cdot\frac{K}{\log K}\cdot n^{1/\sqrt{\log n}}\end{aligned}$$

We obtain thus a contradiction for large enough $n$ when $K$ is chosen so that $K \geqslant e$ and $\frac{K}{\log K} > 1$, which is true for $K = e$ for all $\varepsilon > 0$. □

To conclude, consider now a batch $i$ such that $\overline{s}^i \leqslant \frac{e}{F}$. By (1),

$$\text{MaxFlowTime}(A) \geqslant T^i \geqslant \frac{(1-\frac{1}{\sqrt{\log n}})\log n - O(\log\log n)}{\log(e/(1-1/F^2))} \geqslant (1-o(1))\cdot\log n\cdot\frac{\text{OPT}_1}{2}.$$

## 6 Reduction from the General Setting to Par → Seq Instances

Using reductions from [3,15,16](omitted due to space constraints), we are able to show that Theorem 3 extends to the general setting, as follows:[2]

**Theorem 10.** *For all* $\varepsilon > 0$*, for all instance* $J_1, \ldots, J_n$ *with arbitrary speed-up functions and precedence constraints, there exists a simple algorithm* OBEQUI ∘ EQUI *that is* $(1+\varepsilon)$*-processor* $\frac{(\kappa(J)+1)c_\varepsilon}{2} \cdot \log n$*-competitive, where* $\kappa(J)$ *denotes the maximum number of independent tasks in a job of the instance, and* $c_\varepsilon \leqslant 16(\frac{1}{\varepsilon} + \frac{4}{\varepsilon^2})$ *for* $\varepsilon \leqslant \frac{1}{2}$.

## References

1. Bansal, N., Pruhs, K.: Server scheduling in the $\ell_p$ norm: a rising tide lifts all boat. In: STOC, pp. 242–250 (2003)
2. Becchetti, L., Leonardi, S.: Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. J. ACM 51(4), 517–539 (2004)
3. Edmonds, J.: Scheduling in the dark. TCS 235(1), 109–141 (2000)
4. Edmonds, J., Chinn, D.D., Brecht, T., Deng, X.: Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. J. of Scheduling 6(3), 231–250 (2003)
5. Edmonds, J., Im, S., Moseley, B.: Online scalable scheduling for the lk-norms of flow time without conservation of work, personal communication
6. Edmonds, J., Pruhs, K.: Scalably scheduling processes with arbitrary speedup curves. In: SODA, pp. 685–692 (2009)
7. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling jobs with varying parallelizability to reduce variance. In: SPAA (2010)
8. He, Y., Hsu, W.J., Leiserson, C.E.: Provably efficient online non-clairvoyant adaptive scheduling. In: IPDPS, pp. 1–10 (2007)
9. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM 47, 214–221 (2000)
10. Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. J. ACM 50(4), 551–567 (2003)
11. Merritt, R.: CPU designers debate multi-core future. EETimes (2008)
12. Motwani, R., Phillips, S., Torng, E.: Nonclairvoyant scheduling. TCS 130(1), 17–47 (1994)
13. Pruhs, K.: Competitive online scheduling for server systems. SIGMETRICS Perf. Eval. Rev. 34(4), 52–58 (2007)
14. Pruhs, K., Sgall, J., Torng, E.: Online Scheduling. In: Leung, J.Y.-T. (ed.) Handbook of Scheduling: Algorithms, Models and Performance Analysis, ch. 15. Chapman & Hall/CRC, Boca Raton (2004)
15. Robert, J., Schabanel, N.: Non-clairvoyant batch sets scheduling: Fairness is fair enough. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 742–753. Springer, Heidelberg (2007)
16. Robert, J., Schabanel, N.: Non-clairvoyant scheduling with precedence constraints. In: SODA, pp. 491–500 (2008)

[2] Section 6 can be downloaded in the full version of the present article from the web pages of the authors and will be available in the upcoming journal version of this article.

# An Improved Algorithm for Online Rectangle Filling*

Rob van Stee

Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany
vanstee@mpi-inf.mpg.de

**Abstract.** We consider the problem of scheduling resource allocation where a change in allocation results in a changeover penalty of one time slot. We assume that we are sending packets over a wireless channel of uncertain and varying capacity. In each time slot, a bandwidth of at most the current capacity can be allocated, but changing the capacity has a cost, which is modeled as an empty time slot. Only the current bandwidth and the bandwidth of the immediately following slot are known. We give an online algorithm with competitive ratio 1.753 for this problem, improving over the previous upper bound of 1.848. The main new idea of our algorithm is that it attempts to avoid cases where a single time slot with a nonzero allocation is immediately followed by an empty time slot. Additionally, we improve the lower bound for this problem to 1.6959. Our results significantly narrow the gap between the best known upper and lower bound.

## 1 Introduction

In wireless networks, channel conditions can change frequently, which affects the bit error rate and therefore the channel transmission capacity [5]. We consider the problem of setting data transmission rates over such a channel in order to maximize the throughput. Naturally, at any time the transmission rate cannot be higher than the current transmission capacity, but there is also typically a nonzero cost involved in changing the transmission rate, because the transmitter and receiver will have to coordinate and reset to a new transmission rate. We model this cost as the loss of a single time slot. That is, whenever we want to change the transmission rate (or if we are forced to change it, because the current capacity is below the rate that we set earlier), we will have one time slot in which nothing can be transmitted.

Formally, we are given an online sequence of nonnegative real numbers $h(1)$, $h(2), \ldots$, which represent the maximum transmission capacities of the wireless channel at each time step, and we need to determine the transmission rate $u(i)$ at each time step. Our goal is to maximize $\sum_i u(i)$, and due to the changeover cost we have for any $i$ that $u(i) = u(i+1)$, $u(i) = 0$, or $u(i+1) = 0$. It can be seen that if only the current bandwidth is known, no competitive online algorithm

* Research supported by the German Research Foundation (DFG).

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 249–260, 2011.

exists [2]. We therefore focus on the case where some information about future bandwidth is given; in particular, for our results we assume that we have a lookahead of a single time slot.

The name rectangle filling comes from a geometrical interpretation of the problem, where each time slot $i$ is represented by a rectangle of unit width and height $h(i)$ (also called a *column*). An algorithm needs to decide how much of each rectangle to fill, i.e., what the transmission rate $\in [0, h(i)]$ should be. Any feasible solution for this problem is a set of rectangles (of varying width) where the transmission rate is constant; all these rectangles are separated by one or more zero columns.

Probabilistic analysis for this problem was given by Tsibonis et al. [6] and Borst [3]. Arora and Choi were the first ones to study this problem from a worst-case perspective [1]. They gave a dynamic program for the offline version with a running time of $O(n^3)$, and a 4-competitive online algorithm called Wait Dominate Hold (using a lookahead of 1). Arora et al. [2] soon afterwards showed that this algorithm is actually 8/3-competitive, and gave a lower bound of 8/5. For the version of the problem with $k$-lookahead, they gave an online algorithm with competitive ratio 2 for any $k$, and a lower bound of $(k+2)/(k+1)$.

Wang et al. [7] presented a faster offline algorithm with a running time of $O(n^2)$, and new lower and upper bounds of 1.6358 and 1.848 respectively. In the following year, the same authors [8] considered the version with $k$-lookahead. They gave a deterministic algorithm with a competitive ratio of $1 + 2/(k-1)$, as well as a randomized algorithm with competitive ratio $1 + 1/(k+1)$. They also gave a randomized lower bound of $1 + 1/(\sqrt{k+2} + \sqrt{k+1})^2$, which is more than $1 + 1/(4k+8)$ and tends to $1 + 1/(4k+4)$ for large $k$. The randomized lower bound was recently improved to $(k+2)\ln\frac{k+2}{k+1} > 1 + 1/(2k+3)$ by Epstein and Levin [4].

Generally, despite the seeming simplicity of the model, the gap between the upper and lower bounds has so far remained relatively large, particularly for the most basic version with a lookahead of 1, and it appears to be very hard to give tight bounds for this problem. See also the conclusions.

*Our results.* We give an improved algorithm which achieves a competitive ratio of less than 1.753. The main new idea of our algorithm is to try and limit the amount of changes in the used bandwidth. Therefore, as soon as a nonzero transmission rate is allocated, we temporarily relax the condition for changing the rate, and allow a *single* time slot with relatively high capacity to appear without resetting the transmission rate. The idea behind this is that as long as there is only a single slot with high capacity, we do not lose too much compared to the optimal solution, because the optimal solution needs to allocate zero columns before and after the slot in order to be able to use the full capacity of the slot. That is, it is not worth the trouble of paying a penalty to serve a slot completely, especially if we have very recently paid another penalty to start transmitting.

The analysis uses the natural block partitioning from Wang et al. [7] as a starting point, but is significantly more involved. Apart from their partitioning

rules, we will need a number of additional assignment rules to deal with the columns that are left unassigned in their scheme. These assignment rules depend on the optimal solution, and sometimes assign part of the optimal profit to preceding blocks and part to following blocks in order to allow us to analyze these blocks independently.

Moreover, in some cases we analyze the competitive ratio by splitting the instance into two parts, replacing one column by a sequence of columns, and showing that our algorithm gives exactly the same allocation as before to all columns where it does not allocate 0, and the optimal profit is split according to another set of rules between the first part and the second part of the input.

Finally, we improve the lower bound from 1.6358 to 1.6959. The construction is similar to the one from Wang et al. [7], but we use an extra threat in every step of the input.

## 2 Lower Bound

We let $\varepsilon > 0$ be some very small value. The lower bound of $r = 1.6959$ is constructed according to the following rules. See Figure 1. We have $p < 1$ and $q > 2$. Each rectangle represents a state. The states consist of three components: the current online allocation, the current column, and the next column. Hence, if we move from one state to another, the second column in the starting state is identical to the first column in the destination state. However, we sometimes (conceptually) rescale column heights, for instance when we move from state $B$ to state $A$.

Additionally, the number $x-\varepsilon$ in the ellipse indicates that one final column of height $x-\varepsilon$ arrives, where $x$ is the (nonzero) allocation that the online algorithm just used. Hence, this column has no value for the online algorithm (it can only use the column by forfeiting a profit of $x$ on the current column).

The idea behind this construction is that no 1.6959-competitive algorithm can stay in state $A$ indefinitely, or move back and forth between state $A$ and $B$ indefinitely. The threat that the allocation $x$ is deemed 'too high' and one final column of height $x - \varepsilon$ arrives eventually forces a competitive ratio of 1.6959. What too high means will depend on what has gone before.

Denote the online algorithm by ALG. Our proof consists of the following statements, which hold for all $r < 1.695$.

1. If ALG stays in state $A$ long enough, we can let it move to state $C$ and force a ratio of at least $r$.
2. If ALG stays in state $B$ long enough, the competitive ratio tends to at least $q > r$.
3. If ALG keeps moving between state $A$ and $B$, the implied competitive ratio on the sequence seen so far is at least $q$.
4. The upper limit of what ALG can assign in state $A$ while maintaining a competitive ratio of $r$ decreases monotonically over time.
5. Eventually, ALG must assign such a low value that the competitive ratio is at least $r$ after all.

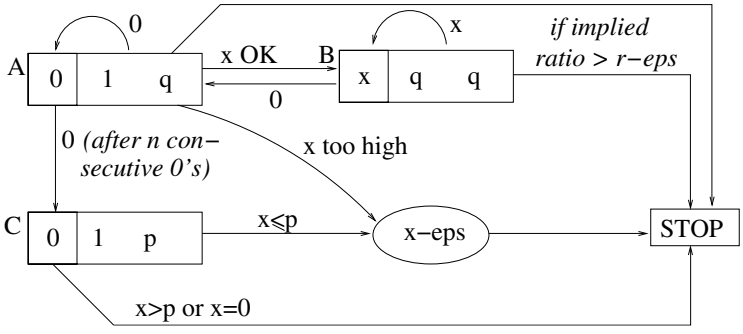



**Fig. 1.** Lower bound. The variable $x$ always indicates the allocation chosen by the online algorithm.

1) Suppose the algorithm spends $n$ steps in state $A$ for some large $n$, and the last time that it assigns 0, it moves to state $C$. We rescale column heights such that the last columns seen so far have heights 1 and $p < 1$. On the preceding $n-1$ columns, it is possible to earn $1/q^2 + 1/q^4 + \ldots$ which tends to $1/(q^2-1)$ and is hence at least $(1-\varepsilon)/(q^2-1)$ for large enough $n_\varepsilon$ for any fixed $\varepsilon > 0$.

In state $C$, ALG must choose an allocation $x$. If $x > p$, the input stops. Else, the next and final column has height $(1-\varepsilon)x$. If $x > p$, the online profit is at most 1, and the optimal profit is at least $2p+(1-\varepsilon)/(q^2-1)-\varepsilon$. If $x \leq p$, the online profit is at most $2x$, whereas the optimal profit is at least $3(1-\varepsilon)x+(1-\varepsilon)/(q^2-1)-\varepsilon$. It can be seen that the online algorithm should not set $x$ below $p$, i.e., it should choose either $p$ or 1. For $\varepsilon \to 0$, the implied competitive ratios tend to

$$\mathcal{R}_1 = \frac{3p + 1/(q^2-1)}{2p} \quad \text{and} \quad \mathcal{R}_2 = 2p + 1/(q^2-1).$$

These ratios are equal for

$$p = \frac{3q^2 - 5 + \sqrt{9q^4 - 14q^2 + 9}}{8q^2 - 8}\ , \tag{1}$$

and this is the value that we use.

2) This is clear because ALG earns only $x \leq 1$ on each column, whereas it is possible to earn $q$ each time.

In our construction, we will set $q = 2.14447$. From (1) we then get $p = 0.709039$. The two statements that we proved above imply that if ALG stays in state $A$ or $B$ sufficiently long, a competitive ratio arbitrarily close to $\mathcal{R}_1 = \mathcal{R}_2 = 1.69595$ follows, since $q > \mathcal{R}_1$. The only option remaining for ALG is to keep moving back and forth. For points 3)–5), we first consider a simple case where ALG always acts in the same way. We partition the input into phases. A new phase starts whenever ALG moves to state $A$ from state $B$.

**Observation 1.** *If* ALG *allocates a nonzero amount (not necessarily the full column height!) whenever it is in state A, and zero whenever it is in state B, it*

*has earned at most* $\sum_{i=0}^{n-1} q^{-i}$ *after* $n$ *phases (if the first column height in phase* $n$ *is 1), while it is possible to earn* $q\sum_{i=0}^{n-1} q^{-i}$ *in these phases.*

Consider an algorithm which acts as in Observation 1 for $n$ consecutive phases. Scale column heights such that the first column height in phase $n$ is 1. Let the first column of the $n$ phases be column $n_0$. Let the profit of ALG on columns $1, \ldots, n_0 - 1$ be $P_0$. Since the column heights are geometrically increasing during the $n$ phases (the lowest column height seen in phase $i$ is $q^{i-n}$), for any $\delta > 0$ and $n$ large enough, we have $P_0 < \frac{\delta}{3q-3} < \frac{\delta}{q}$ after the scaling (possibly $P_0 = 0$).

Write the maximum possible online profit on all columns before the last one as $\alpha_n \sum_{i=0}^{n-1} q^{-i}$. Then by Observation 1 and since $P_0 < \frac{\delta}{q}$ for sufficiently large $n$, we have $\alpha_n < 1 + \delta$ for such $n$. Denote the allocation of ALG in phase $i$ by $x_i$. We will use repeatedly that for $k \geq 1$,

$$\sum_{i=0}^{n-1} q^{-i} = \frac{1 - q^{-n}}{1 - q^{-1}} = \frac{q - q^{1-n}}{q - 1} \ . \tag{2}$$

**Lemma 1.** *Assume that* ALG *acts as in Observation 1 for* $n$ *consecutive phases and has a competitive ratio less than* $(1 - \delta)\mathcal{R}_1$*. Then for* $0 < \delta < 0.2$ *and sufficiently large* $n$*, we have:*

- $1/2 < \alpha_n < 1 + \delta$,
- $1/2 < x_n < (1 - \delta)\alpha_{n-1}$, *and*
- $\alpha_n < (1 - \frac{\delta}{2})\alpha_{n-1}$.

Since these three statements eventually lead to a contradiction (we get $x_{n+1} < (1 - \delta)(1 - \frac{\delta}{2})\alpha_n$ etc.), this proves the lower bound for this type of algorithms.

*Proof.* For the first statement, if $\alpha_n \leq 1/2$ for large enough $n$ the input just stops. ALG can earn at most 1 on the final column. By Observation 1 and (2), the implied competitive ratio is

$$\frac{q\sum_{i=0}^{n-1} q^{-i}}{1 + \alpha_n \sum_{i=0}^{n-1} q^{-i}} = \frac{q \cdot \frac{q - q^{1-n}}{q-1}}{1 + \alpha_n \frac{q - q^{1-n}}{q-1}} \geq \frac{q\left(q - q^{1-n}\right)}{q - 1 + \frac{1}{2}(q - q^{1-n})} \to \frac{q^2}{\frac{3}{2}q - 1} > 2 > \mathcal{R}_1$$

for $n \to \infty$. Similarly, if $x_n \leq 1/2$, the input stops and the implied competitive ratio is at least

$$\frac{q\sum_{i=0}^{n-1} q^{-i}}{2x_n + (1+\delta)\sum_{i=1}^{n-1} q^{-i}} \geq \frac{q(q - q^{1-n})}{q - 1 + (1+\delta)(1 - q^{1-n})} \to \frac{q^2}{q + \delta} > \mathcal{R}_1$$

for $n \to \infty, \delta < 0.2$. (Here we use that already $\alpha_{n-1} < 1 + \delta$.) If $x_n > 1/2$, we consider the input where a final column of height $(1-\varepsilon)x_n$ arrives (after the last column of phase $n$, which has height $q$). Now it is possible to earn $4(1-\varepsilon)x_n$ on the last four columns of the input. Letting $\varepsilon \to 0$, the implied competitive ratio when allocating $y$ after $n$ phases is at least

$$\frac{4x_n + q\sum_{i=2}^{n-1} q^{-i}}{2x_n + \alpha_{n-1}\sum_{i=1}^{n-1} q^{-i}} = \frac{4(q-1)x_n + 1 - q^{2-n}}{2(q-1)x_n + \alpha_{n-1}(1 - q^{1-n})} \ .$$

(Note the indices of the summations.) Let $n$ be large enough so that $q^{2-n} < \delta$. Then it can be verified that

$$\frac{4(q-1)(1-\delta)\alpha_{n-1} + q - \delta}{2(q-1)(1-\delta)\alpha_{n-1} + \alpha_{n-1}(1-\frac{\delta}{q})} > (1-\delta)\mathcal{R}_1.$$

(We get equality for $\alpha_{n-1} = 4/(4-18.69\delta)$, but we have $0 < \alpha_{n-1} < 1+\delta$ so this cannot happen. Note that $4 - 18.69\delta > 0$ since $\delta < 0.2$.) The third statement can be checked by using that $\lim_{n\to\infty} \sum_{i=1}^{n-1} q^{-i} = \frac{q}{q-1}$ and verifying that

$$\frac{\alpha_{n-1}}{q} \cdot \frac{q}{q-1} + \frac{(1-\delta)\alpha_{n-1}}{q} < \left(1 - \frac{\delta}{2}\right) \alpha_{n-1} \cdot \frac{q}{q-1}$$

for $q = 2.144472$, independently of $\alpha_n$ and $\delta$ (the difference is about 0.04). Here we use that due to the scaling which we always do, the profit in phase $n+1$ is (profit until phase $n$ plus whatever the algorithm allocates on the next column)$/q$. Therefore the third statement holds for sufficiently large $n$. □

We give an example that shows a lower bound of 1.69. Here we assume that ALG allocates the highest possible amount in each time that avoids a competitive ratio of 1.69. Slightly deviating from the above, we set $q = 2.169$ and $p = 0.710$.

| Input | 1 | $q$ | $q$ | $q^2$ | $q^2$ | $q^3$ | $q^3$ | $q^4$ | $q^4$ | $q^5$ | $q^5$ | $q^6$ | $q^6$ | $q^7$ | $q^7$ | $q^8$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ALG | 1 | 0 | $q$ | 0 | $q^2$ | 0 | $q^3$ | 0 | $0.98q^4$ | 0 | $0.94q^5$ | 0 | $0.88q^6$ | 0 | $0.76q^7$ | $0.76q^7$ |
| OPT | 0 | $q$ | 0 | $q^2$ | 0 | $q^3$ | 0 | $q^4$ | 0 | $q^5$ | 0 | $q^6$ | 0 | $q^7$ | 0 | $q^8$ |

It remains to be shown that ALG cannot do better by acting differently than in Observation 1. First of all, any time that ALG allocates a nonzero amount in state $B$, this simply gives an additional column on which OPT can earn $q$ times the online profit. Hence the overall competitive ratio does not decrease.

This shows that we only need to deal with phases of the form $A^kB$ for some $k \geq 1$. We defer this part of the proof to the full version.

## 3 Algorithm MoreFilling

We first give an informal description of our algorithm. MoreFilling creates blocks between zero columns. The nonzero part of a block starts by trying to guess a good allocation for the first two nonzero columns. If the first height is much smaller than the second one (by a factor of at least $\gamma > 2$, then the nonzero part of the block is simply postponed till later. Otherwise, if the first height is much larger (by a factor of at least $\frac{1}{\beta} > 1.4$, then the first column is allocated its full height, and the block will contain a single nonzero column. If the two heights are relatively close, the minimum between the two heights is allocated.

The main idea of our algorithm is to try and avoid having zero columns if the nonzero part of a block has just started. Hence there is one case where the first

The first column arrives at time $t = 1$. We take $u(0) = 0$.

1. If $u(t-1) > h(t)$, set $u(t) = 0$.
2. If $u(t-1) = 0$, set $u(t)$ depending on $h(t+1)/h(t)$, as follows:

| $h(t+1)/h(t)$ | $[0, \beta)$ | $[\beta, 1)$ | $[1, \gamma)$ | $[\gamma, \infty)$ |
|---|---|---|---|---|
| $u(t)$ | $h(t)$ | $h(t+1)$ | $h(t)$ | 0 |

3. If $u(t-1) > 0$, let $t_1 = 1 + \max\{t \geq 0 | u(t) = 0\}$, and set $H = \min(h(t_1), h(t_1 + 1))$.
   If $t_1 = t - 1$ and $h(t)/h(t-1) \in [1, \delta]$, set $q_3 := \gamma$, else $q_3 := \delta$.
   If $h(t+1) \geq q_3 H$, set $u(t) = 0$, else $u(t) = u(t-1)$.

**Fig. 2.** The algorithm MoreFilling

nonzero column is fully used, where we allow the column height to grow by a factor of $\gamma > 2$ for just one step, while still keeping other (later) column heights bounded by a smaller factor. This ensures that there is only a zero column if the new column height is significantly larger, or if the nonzero column is allocated its full height.

The third column in a block is almost always limited to a maximum allowed height of $\delta < 1.6$. The only exception is if the second column has height between 1 and $\delta$ times the first column, i.e., the block does not contain a column of height (almost) $\gamma$ yet. We never let the heights grow by a factor of $\gamma$ if the block already contains at least two nonzero columns.

We now formally define our algorithm. Define the following values.

$$\begin{aligned} \mathcal{R} &= &&= 1.75214 & \qquad \beta &= \mathcal{R}/(2\mathcal{R}-1) &&= 0.69966 \\ \varepsilon &= (2\mathcal{R}-3)\beta &&= 0.35282 & \qquad \gamma &= \mathcal{R}/(4-2\mathcal{R}+\varepsilon) &&= 2.06489 \\ \delta &= (2\mathcal{R}\beta - \varepsilon - 1)/\beta &&= 1.57074 & \qquad \eta &= 1 - \mathcal{R}/\gamma + \varepsilon &&= 0.50414 \end{aligned}$$

We have $(5 - \mathcal{R}/\gamma + \varepsilon)/(1 + \delta) = \mathcal{R}$. Our algorithm is defined in Figure 2.

**Theorem 1.** *The competitive ratio of MoreFilling is at most* $\mathcal{R} = 1.75214$.

It should be pointed out that $\delta$ could be set to any value between roughly 1.55 and 5/3, and MoreFilling would still be 1.753-competitive. The other variables, $\beta, \varepsilon$ and $\gamma$ are decisive.

## 4 Analysis

We begin our analysis by introducing the block partitioning and giving some properties of the optimal solution in Section 4.1. We use this to analyze the most basic type of block in Theorem 2. The remainder of the analysis is omitted in this extended abstract.

### 4.1 Block Partitioning

We classify the columns with zero allocation (also called zero columns) by the rule in which they are set to 0: a column is of type $i$ if it is set to 0 by Rule $i$. The zero columns partition the input into blocks in a natural way. Note that type 2 columns only occur after other zero columns. We follow the partitioning scheme from Wang et al. [7], which we describe next.

A type 1 column ends a block, and is a part of that block. If column $i$ is a type 3 column, one block ends at column $i-1$ and the next starts at column $i+1$. (In this case, we will decide later what to do with column $i$.) This partitioning scheme ignores the type 2 columns (which only occur after other zero columns). Each block hence consists of zero or more type 2 columns, followed by one or more nonzero columns and possibly one final type 1 column. For each block $B$, the number of *nonzero* columns in $B$, also called the *length* of $B$, is denoted by $|B|$.

Let us consider the possible optimal profit on a block. We normalize the height of the columns in this block such that the height of the first column is exactly 1.

**Definition 1.** *We define the* BaseHeight *of a block with at least two nonzero columns as the minimum height among its first two columns.*

The BaseHeight is abbreviated by $H$ in the algorithm.

**Definition 2.** *A block is called long if it has at least one (nonzero) column after its first fully-used column, else it is called short.*

**Observation 2.** *Consider a block of which the first nonzero column is column $i$. We have $h(i) = 1$. If $h(i) > h(i+1)$, then $h(i+1) \geq \beta h(i)$, or $u(i+1) = 0$. If $u(i+1) > 0$ and the last column $k$ of this block is of type 1, we have $h(k) < \beta$.*

*If $h(i) \leq h(i+1)$, then $h(i+1) < \gamma$. Moreover, almost always we have $h(j) < \delta$ for all $j > i+1$ such that $j$ and $i$ belong to the same block. The only exception to this is if $h(i+1) \leq \delta$, in which case we have $h(i+2) < \gamma$ if $i+2$ is part of the same block. In addition, if the last column $k$ of the block of $i$ is of type 1, we have $h(k) < \alpha$.*

We now consider the type 2 zero columns at the start of the block.

**Lemma 2 (Wang et al.).** *Given a sequence of columns $S$, if each column's height is at least $\gamma$ times the height of the previous one (where $\gamma \geq 2$), then the value of the optimal solution is at most $\frac{\gamma^2}{\gamma^2-1}h$ where $h$ is the height of the last column in $S$. This value is achieved by using every other column completely starting from the right.*

In order to efficiently deal with all the cases, we will in fact use the following estimates, which are all higher than the bound from Lemma 2. Consider a column $i$ of height $h$ which is preceded by type 2 columns. We make two distinctions:

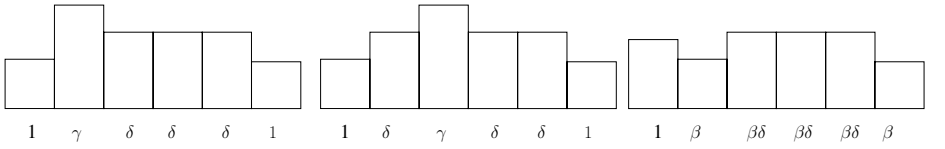



**Fig. 3.** This figure shows the maximum possible heights of nonzero columns in a block (plus a final type 1 column). There are three cases. Let the first nonzero column of the block be $t$, then we have the cases $\delta h(t) < h(t+1) \leq \gamma h(t)$ (left), $h(t) \leq h(t+1) \leq \delta h(t)$ (middle), and $\beta h(t) \leq h(t+1) < h(t)$ (right). If $h(t+1) < \beta h(t)$, the block contains only a single nonzero column; if $h(t+1) > \gamma h(t)$, no block starts at time $t$. The third column may have height $\gamma$ only if the heights are ascending and the second column has height at most $\delta$. Shown is the case where the sixth column is of type 1; if there are additional nonzero columns instead, their maximum height is the same as that of the fifth column.

one based on whether $a(i) > h/\gamma$ or not (if $a(i) > h/\gamma$, then $a(i-1) = 0$), and one based on whether the block containing column $i$ is long or short (Definition 2). The bounds used are as follows.

| | $a(i) > h/\gamma$ | $a(i) \leq h/\gamma$ |
|---|---|---|
| Short block | $\varepsilon \approx 0.353$ | $\varepsilon\gamma \approx 0.729$ |
| Long block | $\eta \approx 0.505$ | $\varepsilon\gamma \approx 0.729$ |

Naturally the profit on type 2 columns does not really depend on the type of the following block, but this assumption simplifies the analysis later.

**Observation 3.** *If the input contains a sequence of columns $i, \ldots, j$ such that $h(k) \geq \gamma h(k-1)$ for $k = i+1, \ldots, j$, then MoreFilling earns 0 on columns $i, \ldots, j-1$.*

To determine the maximum optimal profit on a block, we need to consider how the type 2 blocks at the start of such a block are serviced. Depending on the exact heights of the nonzero columns, it may not be optimal to service them as described in Lemma 2. However, Observation 3 allows us to make the following assumption.

**Assumption 1.** *If columns $i, \ldots, j$ form a sequence of type 2 columns followed by a nonzero column $j$, then $h(k) = \gamma h(k-1)$ for $k = i+1, \ldots, j+1$.*

Here we simply round up column heights, which can only improve the total optimal value. Regarding MoreFilling, if the previous block ended with a type 1 column, its behavior is unaffected. If it ended with a type 3 column, then the column following that was too high for the block to continue, and it is now not less high than before.

Fix an optimal solution and denote its allocation to column $i$ by $a(i)$.

**Lemma 3.** *For each column $i$, we have $a(i) = 0$ or $a(i) > h(i)/3$.*

*Proof.* Suppose the optimal allocation for columns $i-1, i, i+1$ is $x, a(i), z$ (if one of these columns does not exist, assume that its height is zero), then we have $x = 0$ or $x = a(i)$ and we also have $z = 0$ or $z = a(i)$, so if $a(i) < h(i)/3$ we can replace $x, a(i), z$ by $0, h(i), 0$, which is always feasible. □

**Theorem 2.** *MoreFilling is $\mathcal{R}$-competitive on a block which is **not** immediately followed by a type 3 column.*

*Proof.* Let FIRST be the number of the first nonzero column in block $b$, and normalize $h(\text{FIRST}) = 1$. Then, the profit of our algorithm on this block is $|b|$ or $\beta|b|$, depending on whether $h(\text{FIRST}) \leq h(\text{FIRST}+1)$. By the text below Lemma 2, the optimal profit on the type 2 columns in $b$ is at most $\varepsilon\gamma h(\text{FIRST})$ (the height of the last type 2 column is at most $h(\text{FIRST})/\gamma$, and if the last type 2 column is not used, the optimal profit on the other type 2 columns is at most $\varepsilon h(\text{FIRST})$, or at most $\eta h(\text{FIRST})$ if the block is long). For the calculations, we make the worst-case assumption that there is a type 1 column at the end of this block, and it has height 1 (it actually must have smaller height). The exception to this is a block of length 1; in that case, the type 1 column must have height at most $\beta h(\text{FIRST})$ by our algorithm.

Note that all nonzero columns in the block have height at least 1. To derive upper bounds for the optimal profit on a block, we may assume that each of these columns has the maximum height as bounded by Observation 2 (Figure 3). The only allocations that we need to consider to find the optimal profit are values that are the height of at least one column. In particular, for column FIRST, by Lemma 3 only the following allocations need to be considered: $h(\text{FIRST})$, $\frac{1}{\gamma}h(\text{FIRST})$, and possibly $\beta h(\text{FIRST})$.

This gives us the results shown in Figure 4. The figure shows the optimal profit for each case and compares it to the profit of MoreFilling. Regarding a block of length 1 for instance, we find that it is optimal to set $a(\text{FIRST}) = a(\text{FIRST}+1) = h(\text{FIRST}+1) \leq \beta h(\text{FIRST})$, $a(\text{FIRST}-1) = 0$, and then it is possible to earn at most $\varepsilon h(\text{FIRST}) < 0.353 h(\text{FIRST})$ on the type 2 columns up to column $\text{FIRST}-2$.

For a block that contains three or more nonzero columns, note that adding a column of height $\delta$ (resp. $\delta\beta$ in case $h(\text{FIRST}+1) < h(\text{FIRST})$) adds at most $\delta$ (resp. $\delta\beta$) to the optimal profit and exactly 1 (resp. $\beta$) to the profit of MoreFilling. Since $\delta < \mathcal{R}$, this does not increase the competitive ratio above $\mathcal{R}$. □

To complete our analysis, we still need to eliminate the type 3 columns. We will not do this here, but merely indicate the difficulties we encounter in this part of the proof (which is by far the largest part).

Let column $r$ be a type 3 zero column. These columns are the most difficult to handle for the following reason. For type 1 columns, MoreFilling already decided in the previous time step (or earlier) to allocate zero to the type 1 column, because its height is too small compared to some previous height. Similarly, type 2 columns are allocated zero because their height is small compared to the immediately following height. For these cases, it is clear how to group the columns into blocks as described above (i.e., how to compensate for the missed

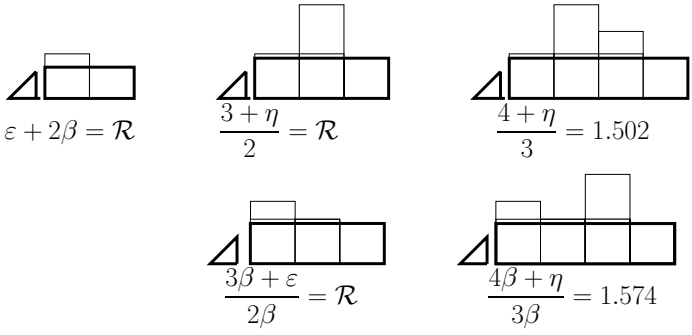



**Fig. 4.** Possible profiles of blocks that are followed by type 1 columns. At the top are blocks which start with a fully-used column, below are blocks where the second column is fully-used. The triangles indicate preceding type 2 columns (that have geometrically increasing heights), of value at most $\varepsilon < 0.353$, or $\eta < 0.505$ for long blocks. The dashed blocks indicate blocks created by MoreFilling. Bold lines indicate the optimal solution for a block.

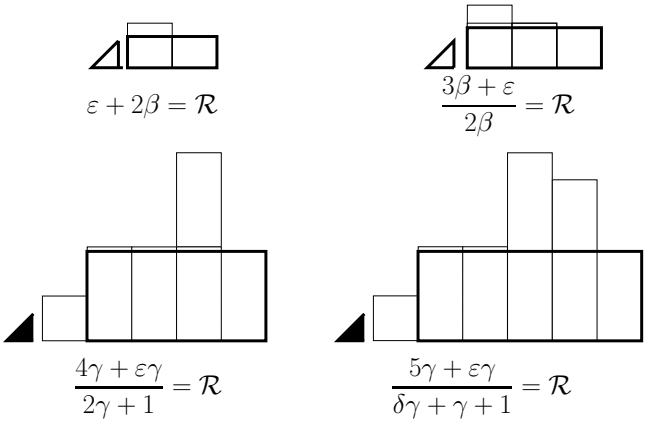



**Fig. 5.** The tight cases. The white triangles represent profits on preceding type two columns of value $\varepsilon$; the black triangles represent profits of value $\varepsilon\gamma$ (in this case, the optimal solution uses the last of the type 2 columns). For an explanation of the other symbols, see Figure 4.

profit on the zero column), and we can analyze these cases in a straightforward way as shown in Theorem 2.

In contrast, type 3 columns are allocated zero "at the last minute", and this decision does *not* depend on its own height. This is in particular troublesome in the case where column $r-1$ was allocated $h(r) < h(r-1)$. In normal cases, column $r$ would be allocated $h(r)$ to compensate for the fact that less than $h(r-1)$ was earned on column $r-1$. But now, column $r$ is allocated zero, and in many cases we will have to consider the block following column $r$ to complete the analysis. We show the tight cases in Figure 5.

## 5 Conclusions

We have narrowed the gap for this problem to 0.056. We believe that both our lower bound and upper bound could potentially be improved, but we conjecture that the lower bound is closer to the true competitive ratio of the problem. However, it is not easy to see how to narrow the gap further. There are four cases where the analysis for our algorithm is tight; additionally, there are various cases where the analysis is nearly tight. An improved algorithm would have to achieve a better ratio in all of the very different tight cases without losing too much in other cases.

*Acknowledgment.* The author would like to thank Leah Epstein and Asaf Levin for interesting discussions, and Marek Chrobak for bringing a gap in the proof of the lower bound to our attention.

## References

1. Arora, A., Choi, H.-A.: Channel aware scheduling in wireless networks. Technical Report 002, George Washington University (2006)
2. Arora, A., Jin, F., Choi, H.-A.: Scheduling resource allocation with timeslot penalty for changeover. Theor. Comput. Sci. 369(1-3), 323–337 (2006)
3. Borst, S.C.: User-level performance of channel-aware scheduling algorithms in wireless data networks. In: INFOCOM (2003)
4. Epstein, L., Levin, A.: Private communication (2009)
5. Stallings, W.: Wireless Communication & Networks. Prentice Hall, Englewood Cliffs (2001)
6. Tsibonis, V., Georgiadis, L., Tassiulas, L.: Exploiting wireless channel state information for throughput maximization. In: INFOCOM (2003)
7. Wang, H., Chaudhary, A., Chen, D.Z.: Online rectangle filling. In: Kaklamanis, C., Skutella, M. (eds.) WAOA 2007. LNCS, vol. 4927, pp. 274–287. Springer, Heidelberg (2008)
8. Wang, H., Chaudhary, A., Chen, D.Z.: New algorithms for online rectangle filling with $k$-lookahead. In: Hu, X., Wang, J. (eds.) COCOON 2008. LNCS, vol. 5092, pp. 385–394. Springer, Heidelberg (2008)

# Approximate Counting for Complex-Weighted Boolean Constraint Satisfaction Problems

Tomoyuki Yamakami

Department of Information Science, University of Fukui
3-9-1 Bunkyo, Fukui 910-8507, Japan

**Abstract.** Constraint satisfaction problems (or CSPs) have been extensively studied in, e.g., artificial intelligence, database theory, graph theory, and statistical physics. A practical application often requires only an approximate value of the total number of assignments that satisfy all given Boolean constraints. There is a known trichotomy theorem for such approximate counting for (non-weighted) Boolean CSPs; namely, all such counting problems are neatly classified into three categories under polynomial-time approximation-preserving reductions. We extend this result to approximate counting for complex-weighted Boolean CSPs, provided that all unary constraints are freely available to use. This marks a significant progress in the quest for the approximation classification of all counting Boolean CSPs. To deal with complex weights, we employ proof techniques along the line of solving Holant problems. Our result also gives an approximation version of the known dichotomy theorem of the complexity of exact counting for such complex-weighted Boolean CSPs.

## 1 Background and Challenges

*Constraint satisfaction problems* (or CSPs) have appeared in many different contexts, such as graph theory, database theory, type inferences, scheduling, and notably artificial intelligence, from which the notion of CSPs was originated. The importance of CSPs comes partly from the fact that the framework of the CSPs is broad enough to capture numerous problems that arise in real applications. A CSP instance is composed of a set of "variables" (over a specified domain) and a set of "constraints" (such a set of constraints is sometimes called a *constraint language*) among these variables. As a decision problem, a CSP asks whether there exists a variable assignment, which satisfies all the given constraints. Typical examples of CSPs include the satisfiability problem (or SAT) and the colorability problem, both of which are known to be NP-complete. On the contrary, other CSPs, such as the perfect matching problem on planar graphs, fall into P. Toward a better understanding of the characteristics of CSPs, one naturally asks what kind of constraints make them NP-complete. Given a set $\mathcal{F}$ of constraints, we restrict our attention on CSP instances that depend only on constraints chosen from $\mathcal{F}$. Such a restricted CSP is conventionally denoted CSP($\mathcal{F}$). A classic *dichotomy theorem* of Schaefer [8] states that if $\mathcal{F}$ is included

K. Jansen and R. Solis-Oba (Eds.): WAOA 2010, LNCS 6534, pp. 261–272, 2011.

in a certain clearly specified class, CSP($\mathcal{F}$) belongs to P; otherwise, it is indeed NP-complete. There are no intermediate Boolean CSPs between P and the class of NP-complete problems.

To count the number of all satisfying assignments for a given CSP instance also has been a challenging question. The counting satisfiability problem, #SAT, is such a counting CSP (or succinctly, #CSP) and it is proven to be complete for Valiant's class #P of counting functions. When restricted to a set $\mathcal{F}$ of Boolean constraints, Creignou and Hermann [3] gave the first dichotomy theorem concerning the complexity of the restricted counting problem #CSP($\mathcal{F}$).

> If all constraints in $\mathcal{F}$ are affine,$^\star$ then #CSP($\mathcal{F}$) is in FP. Otherwise, #CSP($\mathcal{F}$) is #P-complete.

Dyer, Goldberg, and Jerrum [6] extended their result to nonnegative-weighted Boolean #CSPs. Eventually, Cai, Lu, and Xia [2] further pushed the scope to complex-weighted Boolean #CSPs.

However, when we turn our attention from exact counting to *approximate counting*, a situation looks quite different. Instead of the aforementioned dichotomy theorems, Dyer et al. [7] presented a *trichotomy theorem* for the complexity of approximately counting the number of satisfying assignments for each Boolean CSP instance. What they actually proved is that, depending on the choice of a set $\mathcal{F}$ of Boolean constraints, the complexity of approximating #CSP($\mathcal{F}$) can be classified into three categories.

> If all constraints in $\mathcal{F}$ are affine, then #CSP($\mathcal{F}$) is in FP. Otherwise, if all constraints in $\mathcal{F}$ are in a well-defined class, known as $IM_2$, then #CSP($\mathcal{F}$) is equivalent to #BIS. Otherwise, #CSP($\mathcal{F}$) is equivalent to #SAT. The equivalence is defined in terms of polynomial-time approximation-preserving reductions (or AP-reductions).

Here, #BIS is the problem of counting the number of independent sets in a given bipartite graph.

There still remains an unsolved question on the approximation complexity of a "weighted" version of #CSPs: what happens if we expand the scope of #CSPs from non-weighted ones to complex-weighted ones? When we deal with complex-weighted constraints, a significant complication occurs as a result of massive cancellations of weights on the process of summing all weights of the constraints. This situation demands a quite different approach toward the complex-weighted #CSPs. Do we still have a trichotomy theorem, similar to that of Dyer et al.? In this paper, we answer this question affirmatively under a reasonable assumption that all unary (i.e., arity 1) constraints are freely available to use. Let the notation #CSP$^*$($\mathcal{F}$) denote the counting problem #CSP($\mathcal{F}$) with this extra assumption. Such a free use of unary constraints has appeared in the past literature. In case of bounded-degree #CSPs, Dyer et al. [5] assumed free unary unweighted Boolean constraints. Although it is reasonable, this extra condition makes the complexity of #CSP$^*$($\mathcal{F}$) look quite different from the complexity of #CSP($\mathcal{F}$), except for the case of Boolean constraints. When restricted to

$^\star$ An affine relation is a set of solutions of a certain set of linear equations over GF(2).

Boolean constraints, then the only nontrivial unitary constraints are $\Delta_0$ and $\Delta_1$ (which will be explained in Section 2) and thus, as shown in [7], we can eliminate them from the definition of $\#\mathrm{CSP}^*(\mathcal{F})$ using randomized approximation algorithms. In the case of complex-weighted constraints, however, the elimination of all unitary constraints is seemingly impossible.

As the main theorem, we will prove a trichotomy theorem (as in Theorem 1) for the approximation complexity of $\#\mathrm{CSP}^*(\mathcal{F})$'s. This theorem marks a significant progress in the quest of determining the approximation complexity of all $\#\mathrm{CSP}(\mathcal{F})$'s. Our proof heavily relies on the previous works of Dyer et al. [6,7] and, particularly, of Cai et al. [2], which is based on a theory of *signatures* (see, e.g., [1]) that formulate underlying concepts of *holographic algorithms*. A challenging issue is that core arguments of Dyer et al. [7] exploited Boolean natures of constraints but they are not designed to lead to a trichotomy theorem for complex-weighted constraints. Cai's theory of signature, on the contrary, deals with such constraints; however, the theory has been developed over polynomial-time Turing reductions, not meant for AP-reductions. Therefore, our first task is to re-examine the well-known results in this theory and salvage its key arguments that are still valid for our AP-reductions. From that point on, we need to find our own way to establish the desired approximation theory for #CSPs.

All proofs omitted due to page limitation can be found in [10].

## 2 Basic Definitions

We briefly present fundamental notions and notations, which will be used in later sections. Let $\mathbb{N}$ and $\mathbb{C}$ denote respectively the set of all natural numbers (i.e., non-negative integers) and the set of all complex numbers. For convenience, the notation $\mathbb{N}^+$ denotes $\mathbb{N}-\{0\}$. For each number $n \in \mathbb{N}$, $[n]$ denotes the integer set $\{1, 2, \ldots, n\}$. Note that we always treat vectors as *row vectors*, unless stated otherwise.

When we refer to nodes in a given undirected graph, unless there is any ambiguity, we call such nodes by their labels instead of their original node names. For instance, if a node $v$ is labeled by a variable $x$, then we often call it "node $x$," although there are many other nodes labeled $x$, as far as it is clear from the context that which node is referred to.

### 2.1 Signatures, Holant Problems, and #CSPs

For any undirected graph $G = (V, E)$ (where $V$ is a vertex set and $E$ is an edge set) and a vertex $v \in V$, an incident set $E(v)$ of $v$ is the set of all edges *incident* to $v$, and $deg(v)$ is the *degree* of $v$. For any matrix $A$, the notation $A^T$ denotes the *transposed matrix* of $A$.

We follow the terminology developed in [1]. Let $\Sigma$ be a finite set with $|\Sigma| \geq 1$. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two sets of functions $f : \Sigma^m \to \mathbb{C}$ with $m \geq 1$. A *bipartite Holant problem* $\mathrm{Holant}(\mathcal{F}_1|\mathcal{F}_2)$ is a counting problem defined as follows. The problem takes an instance, called a *(bipartite) signature grid* $\Omega = (G, \mathcal{F}'_1|\mathcal{F}'_2, \pi)$, which consists of a finite undirected bipartite graph $G = (V_1|V_2, E)$ (assuming

$V_1 \cap V_2 = \emptyset$ for simplicity), two finite subsets $\mathcal{F}_1' \subseteq \mathcal{F}_1$ and $\mathcal{F}_2' \subseteq \mathcal{F}_2$, and a labeling function $\pi$ such that each vertex $v \in V_i$ is labeled by a function $\pi(v) : \{0,1\}^{deg(v)} \to \mathbb{C}$ in $\mathcal{F}_i'$, where $i \in \{1,2\}$. For convenience, we sometimes write $f_v$ for $\pi(v)$. Assuming the standard lexicographic order on $\Sigma^{deg(v)}$, we express $f_v$ as a series of its output values, which is identified with an element in the space $\mathbb{C}^{\otimes|\Sigma|^{deg(v)}}$. For instance, if $\Sigma = \{0,1\}$ and $deg(v) = 2$, then $f_v = (f_v(00), f_v(01), f_v(10), f_v(11))$. Each function $f$ in $\mathcal{F}_1 \cup \mathcal{F}_2$ is called a *signature*. When all entries in a signature $f$ sit in $\{0,1\}$, we particularly call $f$ a *Boolean signature*. A signature $f$ is *symmetric* iff $f$'s values depend only on the Hamming weight of inputs. For any symmetric function $f$ of arity $k$, we use another notation $f = [f_0, f_1, \ldots, f_k]$, where each $f_i$ is the value of $f$ on inputs of Hamming weight $i$. For example, if $f$ is the equality function ($EQ_k$) of arity $k$, then it is expressed as $[1, 0, \ldots, 0, 1]$ ($k-1$ zeros). We use two special signatures: $\Delta_0 = [1, 0]$ and $\Delta_1 = [0, 1]$. Generally, let $\mathcal{U}$ denote the set of all *unary* (i.e., arity 1) signatures.

Let $Asn(E)$ be the set of all edge assignments $\sigma : E \to \{0,1\}$. The *bipartite Holant problem* is to compute $\text{Holant}_\Omega =_{def} \sum_{\sigma \in Asn(E)} \prod_{v \in V} f_v(\sigma|E(v))$, where $\sigma|E(v)$ denotes the binary string $(\sigma(w_1), \sigma(w_2), \cdots, \sigma(w_k))$ if $E(v) = \{w_1, w_2, \ldots, w_k\}$, sorted in a certain pre-fixed order.

Here, we need to address a technical issue concerning complex-valued functions. Recall that each instance to a Holant problem involves a finite set of signatures. How can we compute those signatures? More importantly, how can we receive them as a part of input instance in the first place? For our core subject on the (approximate) computability of a Holant problem, it is quite convenient to treat such a signature $f$ of arity $k$ as a "black box," which answers the complex value $f(x)$ instantly whenever one makes a query $x \in \{0,1\}^k$. In this way, we do not need to include the entire description of $f$ (e.g., bit sequences representing complex numbers) as a part of the instance for the Holant problem. See Section 2.2 for a further discussion on the running time of an algorithm that takes complex-valued signatures.

Let us define complex-weighted Boolean #CSP problems. Associated with a set $\mathcal{F}$ of signatures, a complex-weighted Boolean #CSP problem, denoted $\#\text{CSP}(\mathcal{F})$, takes a finite set $G$ of *constraints* (that is, signatures) of the form $h(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$ on Boolean variables $x_1, x_2, \ldots, x_n$, where $i_1, \ldots, i_k \in [n]$, $h \in \mathcal{F}$, and it outputs the value: $\sum_{x_1, x_2, \ldots, x_n \in \{0,1\}} \prod_{h \in G} f(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$. In a connection to Holant problems, we can view $\#\text{CSP}(\mathcal{F})$ as a special case of bipartite Holant problem of the following form: an instance of $\#\text{CSP}(\mathcal{F})$ is a bipartite graph $G$, where all vertices on the left-hand side are labeled by variables with the equality functions ($EQ_k$) and all vertices on the right-hand side are labeled by constraints. In short, $\#\text{CSP}(\mathcal{F})$ is just another name for $\text{Holant}(\{EQ_k\}_{k \geq 1} | \mathcal{F})$. Throughout this paper, we interchangeably use these two different ways to view the complex-weighted Boolean #CSP problems.

To improve readability, we often omit the set notation and express, e.g., $\#\text{CSP}(f, \mathcal{F}, \mathcal{G})$ to mean $\#\text{CSP}(\{f\} \cup \mathcal{F} \cup \mathcal{G})$. When we allow any unary

signature to use for free of charge, we briefly write $\#\mathrm{CSP}^*(\mathcal{F})$ instead of $\#\mathrm{CSP}(\mathcal{U}, \mathcal{F})$. In the rest of this paper, we will target the counting problems $\#\mathrm{CSP}^*(\mathcal{F})$.

### 2.2 Randomized Approximation Schemes

We give a general treatment to randomized approximation schemes. Let $F$ be any counting function mapping from $\{0,1\}^*$ to $\mathbb{C}$. To treat complex numbers, we need to modify the standard notions of computability and randomized approximation schemes that are based on the binary alphabet. Now, let us define the function class $\mathrm{FP}_{\mathbb{C}}$ as the set of all complex-valued functions that can be computed deterministically in polynomial time. It is important to note that, as we have stated before, we do not treat complex numbers as bit sequences; rather, we treat the complex numbers as basic "objects" and then perform "natural" operations (such as, multiplications, addition, division, etc.) on them as basic operations, each of which requires only constant time to execute. To given complex numbers, we apply such natural operations only in a very simple and clear fashion; therefore, our assumption on the constant execution time of the operations causes no harm in a later discussion on the computability of $\#\mathrm{CSP}(\mathcal{F})$. (See [1] for further justification.)

The notation $\mathrm{Re}(\alpha)$ ($\mathrm{Im}(\alpha)$, resp.) denotes the real part (imaginary part, resp.) of a complex number $\alpha$. A *randomized approximation scheme* for $F$ is a randomized algorithm that takes a standard input $x \in \Sigma^*$ together with an error tolerance parameter $\varepsilon \in (0,1)$, and outputs values $w$ with probability at least $3/4$ for which

1. $\min\{\gamma\mathrm{Re}(F(x)), \gamma^{-1}\mathrm{Re}(F(x))\} \leq \mathrm{Re}(w) \leq \max\{\gamma\mathrm{Re}(F(x)), \gamma^{-1}\mathrm{Re}(F(x))\}$ and
2. $\min\{\gamma\mathrm{Im}(F(x)), \gamma^{-1}\mathrm{Im}(F(x))\} \leq \mathrm{Im}(w) \leq \max\{\gamma\mathrm{Im}(F(x)), \gamma^{-1}\mathrm{Im}(F(x))\}$,

where $\gamma = e^{-\varepsilon}$ and $e$ is the base of natural logarithms. Moreover, a *fully polynomial-time randomized approximation scheme* (or simply, *FPRAS*) for $F$ is a randomized approximation scheme for $F$ that runs in time polynomial in $(|x|, 1/\varepsilon)$.

Given two functions $F$ and $G$, a *polynomial-time approximation-preserving reduction* (or *AP-reduction*) from $F$ to $G$ is a randomized algorithm $M$ that takes a pair $(x, \varepsilon) \in \Sigma^* \times (0,1)$ as input, uses an arbitrary randomized approximation scheme $N$ for $G$ as oracle, and satisfies the following conditions: (i) $M$ is a randomized approximation scheme for $F$; (ii) every oracle call made by $M$ is of the form $(w, \delta) \in \Sigma^* \times (0,1)$ with $\delta^{-1} \leq \mathit{poly}(|x|, 1/\varepsilon)$ and its answer is the outcome of $N$ on $(w, \delta)$; and (iii) the running time of $M$ is bounded from above by a certain polynomial in $(|x|, 1/\varepsilon)$, not depending on the choice of $N$. In this case, we say that $F$ is *AP-reducible* to $G$ via $M$ and we also write $F \leq_{\mathrm{AP}} G$. If $F \leq_{\mathrm{AP}} G$ and $G \leq_{\mathrm{AP}} F$, then $F$ and $G$ are *AP-interreducible* and we write $F \equiv_{\mathrm{AP}} G$. The following lemma is straightforward.

**Lemma 1.** *If* $\mathcal{F} \subseteq \mathcal{G}$, *then* $\#\mathrm{CSP}^*(\mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(\mathcal{G})$.

## 3 T-Constructibility

We will present a key technique of constructing various signatures from a given set of signatures by applying seven simple operations, while maintaining the AP-reducibility between them.

To pursue notational succinctness, we use the following succinct notations throughout this paper. For any index $i \in [k]$ and any bit $c \in \{0,1\}$, let $f^{x_i=c}$ denote the function $g$ satisfying that $g(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_k) = f(x_1,\ldots,x_{i-1},c,x_{i+1},\ldots,x_k)$ for every tuple $(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_k) \in \{0,1\}^{k-1}$. For any two indices $i,j \in [k]$ with $i < j$, we denote by $f^{x_i=x_j}$ the function $g$ defined as $g(x_1,\ldots,x_i,\ldots,x_{j_1},x_{j+1},\ldots,x_k) = f(x_1,\ldots,x_i,\ldots,x_{j-1},x_i,x_{j+1},\ldots,x_k)$. Moreover, let $f^{x_i=*}$ be the function $g$ defined as $g(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_k) = \sum_{x_i\in\{0,1\}} f(x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_k)$.

We say that an arity-$k$ signature $f$ is *T-constructible* (or *T-constructed*) from a set $\mathcal{G}$ of signatures if $f$ can be obtained, initially from signatures in $\mathcal{G}$, by applying a finite number (possibly zero) of operations described below.

1. *Permutation*: for two indices $i,j \in [k]$ with $i < j$, by exchanging two columns $x_i$ and $x_j$ with $i < j$ in $(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_k)$, transform $g$ into $g'$, which is defined by $g'(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_k) = g(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_k)$.
2. *Pinning*: for an index $i \in [k]$ and a bit $c \in \{0,1\}$, build $g^{x_i=c}$ from $g$.
3. *Projection*: for an index $i \in [k]$, build $g^{x_i=*}$ from $g$.
4. *Linking*: for two distinct indices $i,j \in [k]$, construct $g^{x_i=x_j}$ from $g$.
5. *Expansion*: for an index $i \in [k]$, introduce a new "free" variable, say, $y$ and transform $g$ into $g'$, which is defined by $g'(x_1,\ldots,x_i,y,x_{i+1},\ldots,x_k) = g(x_1,\ldots,x_k)$.
6. *Multiplication*: from two signatures $f$ and $g$ of arity $k$, build $g_1 \cdot g_2$, which is defined as $(g_1 \cdot g_2)(x_1,\ldots,x_k) = g_1(x_1,\ldots,x_k)g_2(x_1,\ldots,x_k)$.
7. *Normalization*: for a constant $\lambda \in \mathbb{C} - \{0\}$, build $\lambda \cdot g$ from $g$, where $\lambda \cdot g$ is defined as $(\lambda \cdot g)(x_1,\ldots,x_k) = \lambda \cdot g(x_1,\ldots,x_k)$.

When $f$ is T-constructible from $\mathcal{G}$, we briefly write $f \leq_{con} \mathcal{G}$. In particular, when $\mathcal{G}$ is a singleton $\{g\}$, we write $f \leq_{con} g$ instead of $f \leq_{con} \{g\}$.

The usefulness of T-constructibility comes from the following lemma, which indicates the invariance of the T-constructibility under AP-reductions.

**Lemma 2.** *If $f \leq_{con} \mathcal{G}$, then $\#\mathrm{CSP}^*(f,\mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(\mathcal{G},\mathcal{F})$ for any set $\mathcal{F}$ of signatures.*

## 4 Relations and Signature Sets

A *relation* of arity $k$ is a subset of $\{0,1\}^k$. Such a relation can be also viewed as a function mapping Boolean variables to $\{0,1\}$ (i.e., $x \in R$ iff $R(x) = 1$, for every $x \in \{0,1\}^k$) and it can be treated as a Boolean signature as well. For instance, logical relations $OR$, $NAND$, $XOR$, and $Implies$ are all expressed

as appropriate Boolean signatures in the following manner: $OR = [0,1,1]$, $NAND = [1,1,0]$, $XOR = [0,1,0]$, and $Implies = (1,1,0,1)$.

For each signature $f$ of arity $k$, its *underlying relation* is the set $R_f = \{x \in \{0,1\}^k \mid f(x) \neq 0\}$. A relation $R$ is said to be *affine* if it is expressed as a set of solutions to a certain system of linear equations over $GF(2)$. Let $AFFINE$ be the set of all such affine relations. Moreover, a non-empty set $\mathcal{F}$ of relations is also called *affine* if $\mathcal{F}$ is a subset of $AFFINE$. A relation $R$ is in $IMP$ (slightly different from $IM_2$ in [7]) if it is logically equivalent to a conjunction of a certain "positive" number of relations of the form $\Delta_0(x)$, $\Delta_1(x)$, and $Implies(x,y)$. It is worth mentioning that $EQ_2 \in IMP$ but $EQ_1 \notin IMP$.

The purpose of this paper is to extend the trichotomy theorem of Dyer et al. [7] for relations, stated in Section 1, to another trichotomy theorem for complex signatures. Recall that $\mathcal{U}$ denotes the set of all unary signatures. To simplify our further descriptions, it is better to introduce the following five special sets of signatures.

1. Let $\mathcal{NZ}$ denote the set of all non-zero signatures.
2. Let $\mathcal{DG}$ denote the set of all signatures $f$ of arity $k$ that are expressed by products of $k$ unary functions, which are applied respectively to $k$ variables. A signature in $\mathcal{DG}$ is called *degenerate*. Obviously, $\mathcal{U} \subseteq \mathcal{DG}$. See [2] for its basic property.
3. Let $\mathcal{ED}$ denote the set of functions expressed as products of unary signatures, the equality $EQ_2$, and the disequality $XOR$ (which are possibly multiplied by complex constants). Clearly, $\mathcal{DG} \subseteq \mathcal{ED}$. The name $\mathcal{ED}$ refers to its key components, "equality" and "disequality." See [2] for its definition.
4. Let $\mathcal{IM}$ be the set of all signatures $f$ such that $R_f$ is in $IMP$ and $f$ equals $R_f \cdot g$ for a certain non-zero signature $g$. Here, $R_f$ is viewed as a Boolean function (i.e., $R(x) = 1$ iff $x \in R$, for all $x$'s). It is important to note that $\mathcal{IM} \cap \mathcal{NZ} = \emptyset$, because $IMP \cap \mathcal{NZ} = \emptyset$.
5. Let $\mathcal{AF}$ denote the set of all signatures of the form $g(x_1, \ldots, x_k) \prod_{j:j\neq i} R_j(x_i, x_j)$ for a certain fixed index $i \in [k]$, where $g$ is in $\mathcal{DG}$ and each $R_j$ is an affine relation. Note that $f \in \mathcal{AF}$ implies $R_f \in AFFINE$. The name $\mathcal{AF}$ comes from its "affine"-like nature. (Compare this with $\mathcal{A}$ in [2].)

## 5 Counting Complex-Weighted Solutions

We will discuss two important counting problems used in the literature. The *counting satisfiability problem* #SAT is a problem of counting the number of truth assignments that make each given propositional formula true. This problem was proven to be complete for #P under AP-reductions [4]. Another crucial problem in this paper is the *counting downset problem*, which is proven to be AP-interreducible to #BIS (i.e., the problem of counting the number of independent sets in a given bipartite graph). For any partially ordered set $(X, \preceq)$, a *downset* in $(X, \preceq)$ is a subset $D$ of $X$ that is closed downward under $\preceq$: namely, for any $x, y \in X$, if $x \preceq y$ and $y \in D$ then $x \in D$. The counting downset problem,

denoted #DOWNSET, takes an instance of a partially ordered set $(X, \preceq)$, and it outputs the number of all downsets in $(X, \preceq)$.

Dyer et al. [7] showed that those two counting problems, #SAT and #DOWNSET, posses the computational power equivalent to #CSP($OR$) and #CSP($Implies$), respectively, under AP-reductions. Nevertheless, to deal with complex-weighted counting problems rather than unweighted ones, we need to introduce complex-weighted versions of #SAT and #DOWNSET.

In a quite straightforward way, we can define $\#\mathrm{SAT}^*_{\mathbb{C}}$, a complex-weighted version of #SAT. Let $\phi$ be any propositional formula and let $V(\phi)$ be the set of all variables appearing in $\phi$. Let $\{w_x\}_{x\in V(\phi)}$ be any series of *node-weight functions* $w_x : \{0,1\} \to \mathbb{C} \setminus \{0\}$. Given such a pair $(\phi, \{w_x\}_{x\in V(\phi)})$, $\#\mathrm{SAT}^*_{\mathbb{C}}$ outputs the sum of all weights $w(\sigma)$ over every truth assignment $\sigma$ satisfying $\phi$, where $w(\sigma)$ denotes the product of all $w_x(\sigma(x))$'s over every $x \in V(\sigma)$. If $w_x(\sigma(x))$ always equals 1 for every pair of $\sigma$ and $x \in V(\sigma)$, then we immediately obtain #SAT.

**Lemma 3.** $\#\mathrm{SAT}^*_{\mathbb{C}} \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(OR)$.

Similar to $\#\mathrm{SAT}^*_{\mathbb{C}}$, we introduce a complex-weighted version of #DOWNSET, denoted $\#\mathrm{DOWNSET}^*_{\mathbb{C}}$. Let $(X, \preceq)$ be any partially ordered set and let $\{w_x\}_{x\in X}$ be any series of node-weight functions $w_x : \{0,1\} \to \mathbb{C} - \{0\}$. The counting problem $\#\mathrm{DOWNSET}^*_{\mathbb{C}}$ for complex-weighted downsets takes an instance $((X, \preceq), \{w_x\}_{x\in X})$ and outputs the sum of all weights $w(D)$ over any downset $D$ of $(X, \preceq)$, where $w(D)$ means $\prod_{x\in D} w_x(1)$. Moreover, we introduce another variant of #DOWNSET, denoted $\#\mathrm{DOWNSET}_{\mathbb{C}}$. Unlike $\#\mathrm{DOWNSET}^*_{\mathbb{C}}$, $\#\mathrm{DOWNSET}_{\mathbb{C}}$ takes a pair $((X, \preceq), w)$, where $w : P(X) \to \mathbb{C} - \{0\}$, as an instance and it outputs the sum of all weights $w(D)$ of any downset $D$ in $(X, \preceq)$, where $P(X)$ is the power set of $X$.

**Proposition 1.** *1.* $\#\mathrm{DOWNSET}^*_{\mathbb{C}} \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(Implies)$.
*2. For any signature set* $\mathcal{F}$*, if* $\mathcal{F} \subseteq \mathcal{IM}$*, then* $\#\mathrm{CSP}^*(\mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{DOWNSET}_{\mathbb{C}}$.

## 6 Elementary AP-Reductions

In the rest of this paper, we intend to prove our trichotomy theorem (Theorem 1). Its proof is comprised of several crucial ingredients. A starting point of the proof is the computability result, proven by Cai et al. [2, Section 3], that, for any set $\mathcal{F}$ of signatures, if $\mathcal{F} \subseteq \mathcal{ED}$ then $\#\mathrm{CSP}(\mathcal{F})$ belongs to $\mathrm{FP}_{\mathbb{C}}$. From this, we can prove:

**Lemma 4.** *For any signature set* $\mathcal{F}$*, if either* $\mathcal{F} \subseteq \mathcal{AF}$ *or* $\mathcal{F} \subseteq \mathcal{ED}$*, then* $\#\mathrm{CSP}^*(\mathcal{F})$ *is in* $\mathrm{FP}_{\mathbb{C}}$.

In the remainder of this paper, we will focus our attention on the remaining case where $\mathcal{F} \nsubseteq \mathcal{AF} \cup \mathcal{ED}$. Now, we begin exploring useful properties of binary (i.e., arity 2) signatures. We remark that, since all unary signatures are free to use, it holds that $\#\mathrm{CSP}^*(\Delta_0, \Delta_1, \mathcal{F}) \equiv_{\mathrm{AP}} \#\mathrm{CSP}^*(\mathcal{F})$ for any set $\mathcal{F}$ of signatures.

As one can easily see, two relations, $OR$ and $NAND$, are similar in nature. T-constructibility helps establish the AP-interreducibility between $OR$ and $NAND$ in terms of #CSP*s.

**Lemma 5.** *For any signature set $\mathcal{F}$, it holds that $\#\mathrm{CSP}^*(OR, \mathcal{F}) \equiv_{\mathrm{AP}} \#\mathrm{CSP}^*(NAND, \mathcal{F})$.*

The following lemma shows various lower bounds of #CSP*s restricted to binary signatures. These lower bounds will serve a basis to more general lower bounds of #CSP*s in Section 8.

**Lemma 6.** *Each of the following statements holds. Let $\mathcal{F}$ be any signature set.*

1. *If $a \in \mathbb{C}$ with $a \neq 0$, then $\#\mathrm{CSP}^*(OR, \mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*([0, a, 1], \mathcal{F})$ and $\#\mathrm{CSP}^*(OR, \mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*([1, a, 0], \mathcal{F})$.*
2. *If $a, b \in \mathbb{C}$ with $ab \neq 0$, then $\#\mathrm{CSP}^*(XOR, \mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*((0, a, b, 0), \mathcal{F})$ and $\#\mathrm{CSP}^*(OR, \mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*((0, a, b, 1), \mathcal{F})$. The same statement holds for $(1, a, b, 0)$.*
3. *Let $f = (1, a, 0, b)$ with $a, b \in \mathbb{C}$. If $ab \neq 0$, then $\#\mathrm{CSP}^*(Implies, \mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(f, \mathcal{F})$. By permutation, $(1, 0, a, b)$ also yields the same consequence.*
4. *Let $x, y, z \in \mathbb{C}$. If $xyz \neq 0$ and $xy \neq z$, then $\#\mathrm{CSP}^*(OR, \mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*((1, x, y, z), \mathcal{F})$.*

## 7 Affine Support and Imp Support

Underlying relations of complex-valued signatures $f$ play a distinguishing role in our analysis of the behaviors of $\#\mathrm{CSP}^*(f)$. In particular, relations in $AFFINE \cup IMP$ are crucial part of the proof of our trichotomy theorem.

To handle the properties of $AFFINE$ and $IMP$, it is convenient to introduce a notion of affine support [6] and "imp" support. A signature $f$ is said to have *affine support* if its underlying relation $R_f$ is affine. Clearly, every signature in $\mathcal{AF}$ has affine support; moreover, all unary signatures have affine support. Signatures that have affine support will be characterized in Lemma 7. A signature $f$ has *imp support* if its underlying relation $R_f$ is in $IMP$. All signatures in $\mathcal{IM}$ obviously have imp support.

In this section, we will present three technical lemmas, which capture useful properties of affine support and imp support. The first lemma gives a complete characterization of non-zero signatures that have affine support. Notice that if either $f \in \mathcal{NZ}$ or $f$ has arity 1 then $f$ has affine support. For any Boolean matrix $A$, the notation $\xi_A$ denotes the function defined as follows: if $AX^T = O$ then $\xi_A(x_1, \ldots, x_k) = 1$; otherwise, $\xi_A(x_1, \ldots, x_k) = 0$, where $O$ is an all-0 column vector, $X = (x_1, \ldots, x_k, 1)$, and $AX^T$ is calculated over $GF(2)$.

**Lemma 7.** *For any signature $f \notin \mathcal{NZ}$ of arity $k \geq 2$, $f$ has affine support iff there exist a $(k+1)$-by-$(k+1)$ Boolean matrix $A$, a signature $g$, a number $m$ with $1 \leq m < k$, and (after properly permuting variable indices) variables $x_1, x_2, \ldots, x_m$, which are free in the equation $AX^T = O$, and variables $x_{m+1}, x_{m+2}, \ldots, x_k$, which are depending on these free variables, such*

*that* $f(x_1,\ldots,x_m,\ldots,x_k) = \xi_A(x_1,\ldots,x_m,\ldots,x_k)g(x_1,\ldots,x_m)$ *and* $R_g \supseteq \xi_A^{x_{m+1}=*,\ldots,x_k=*}$ *(seen as sets), where* $X = (x_1,x_2,\ldots,x_k,1)$. *Moreover, in this case, if* $g \in \mathcal{AF}$, *then* $f \in \mathcal{AF}$.

The above lemma gives a certain canonical form to signatures when they have affine support. As a corollary of the lemma, we obtain:

**Corollary 1.** *Let* $f$ *be any signature of arity* $k \geq 3$ *with* $f \notin \mathcal{AF} \cup \mathcal{NZ}$. *Let* $\mathcal{F}$ *be any set of signatures. If* $f$ *has affine support, then there exists a signature* $g$ *of arity* $m$ *such that* $2 \leq m < k$, $g \notin \mathcal{AF}$, $g \leq_{con} f$, *and either* $g$ *is a non-zero signature or* $g$ *has no affine support. In particular,* $g$ *is of the form* $f^{x_{m+1}=*,\ldots,x_k=*}$ *after an appropriate permutation of variable indices.*

From any given signature $f$, it is possible to extract factors of the form $\Delta_0(x)$, $\Delta_1(x)$, and $EQ_2(x,y)$ from $f$. After such an extraction, the remaining portion of the signature can be expressed by a notion of simple form. For every arity-$k$ signature $f$, we consider its *representing Boolean matrix* $M_f$, whose rows are indexed by all instances $a = (a_1,a_2,\ldots,a_k)$ in $R_f$ (in the standard lexicographical order), columns are indexed by then numbers in $[k]$, and each $(a,i)$-entry is a Boolean value $a_i$. We say that a signature is in *simple form* if its representing Boolean matrix does not contain all-0 columns, all-1 columns, or any pair of identical columns.

As is stated in the lemma below, we can always factorize a given signature into two factors, at least one of which is in simple form.

**Lemma 8.** *For any arity-k signature* $f$, *there exist two indices* $m$ *and* $m'$ *with* $1 \leq m \leq m' \leq k$, *a relation* $R \in (IMP \cap AFFINE \cap \mathcal{ED}) \cup \{[1,1]\}$, *and a signature* $g$ *such that (after properly permuting variable indices)* $f(x_1,\ldots,x_k) = R(x_1,\ldots,x_{m'})g(x_m,\ldots,x_k)$, $g \leq_{con} f$, *and* $g$ *is in simple form. Moreover,* $f$ *has affine support iff* $g$ *has affine support.*

It is useful to stretch the aforementioned notion of simple form by further excluding any pair of *complementary* columns from representing Boolean matrices. To be more precise, a signature $f$ is said to be in *clean form* if there is no column, specified below, in the representing Boolean matrix $M_f$: (i) all-0 columns, (ii) all-1 columns, (iii) two identical columns, and (iv) two columns which are complementary (i.e., the component-wise XOR of the two columns becomes an all-1 column).

**Lemma 9.** *For each signature* $f$ *of arity* $k \geq 1$, *there exist two indices* $m$ *and* $m'$ *with* $1 \leq m \leq m' \leq k$, *a relation* $p \in (AFFINE \cap \mathcal{ED}) \cup \{[1,1]\}$, *and a signature* $g$ *(or it might possibly be the constant* 1*) such that (after permuting variable indices)* $f(x_1,\ldots,x_k) = R(x_1,\ldots,x_{m'})g(x_m,\ldots,x_k)$ *and* $g$ *is in clean form. Moreover, (i) if* $f \notin \mathcal{ED}$ *then* $g \notin \mathcal{ED}$ *and (ii) if* $f$ *has affine support iff* $g$ *has affine support.*

Before closing this section, we present a useful property of signatures in clean form. Let $S_3$ denote the symmetric group over $\{1,2,3\}$. The proof of this property is part of the proof of [2, Lemma 4.4].

**Lemma 10.** *Let $f \notin \mathcal{ED}$ of arity $k \geq 2$ having affine support. If $f$ is in clean form, then the following two statements hold.*

1. *If $f \notin \mathcal{NZ}$ and $k \geq 3$, then there exists a signature $h \in \{[a,0,1,0],[0,1,0,a]\}$ not in $\mathcal{ED}$ with $a \neq 0$ such that $h \leq_{con} f$. In particular, $h(x_1,x_2,x_3) = \prod_{\sigma \in S_3} g(x_{\sigma(1)},x_{\sigma(2)},x_{\sigma(3)})$ after an appropriate index re-ordering, where $g$ (before normalizing) is of the form $h^{x_4=0,\ldots,x_m=0,x_{m+1}=*,\ldots,x_k=*}$ for a certain index $m \leq k$.*
2. *If $f \in \mathcal{NZ}$, then there exists a signature $g = (1,x,y,z) \notin \mathcal{ED}$ with $xyz \neq 0$ and $z \neq xy$ such that $g \leq_{con} f$. In particular, $g$ (before normalizing) is of the form $f^{x_3=c_3,\ldots,x_k=c_k}$ for certain constants $(c_3,\ldots,c_k) \in \{0,1\}^{k-2}$, after an appropriate permutation of variables.*

## 8 Lower Bounds of #CSP*s

We have shown in Section 5 three specific upper bounds on the approximation complexity of $\#\mathrm{CSP}^*(f)$. To complement those bounds, we will present four lower bounds for $\#\mathrm{CSP}^*(f)$ by building appropriate AP-reductions to one of the following counting problems: $\#\mathrm{CSP}^*(OR)$, $\#\mathrm{CSP}^*(XOR)$, and $\#\mathrm{CSP}^*(Implies)$.

We first discuss the case where $f$ is a signature that lacks affine support. The proof of this proposition follows from Lemmas 6(1) and 6(3) using a part of the proof of [2, Lemma 4.2].

**Proposition 2.** *Let $f$ be any signature. If $f$ has no affine support, then there exists a signature $g \in \{OR, Implies\}$ such that $\#\mathrm{CSP}^*(g,\mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(f,\mathcal{F})$ for any signature set $\mathcal{F}$.*

In the next proposition, we target signatures that have no imp support. The proof of the proposition relies on Lemmas 6(2) as well as the following simple fact: a binary signature $f$ is not in $\mathcal{IM} \cup \mathcal{NZ}$ iff $f$ is of the form $(a,b,0,c)$ with $ad = 0$ and $bc \neq 0$.

**Proposition 3.** *Let $f$ be any signature not in $\mathcal{NZ}$. If $f$ has no imp support, then there exists a signature $g \in \{OR, XOR\}$ such that $\#\mathrm{CSP}^*(g,\mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(f,\mathcal{F})$ for any signature set $\mathcal{F}$.*

In what follows, we are focused on signatures in $\mathcal{IM}$. The proof of the statement below requires Lemmas 6(3) and 8 as well as a structural property of $\mathcal{IM}$ signatures that either they have no affine support or they are not in $\mathcal{ED}$. Recall that $\mathcal{IM} \cap \mathcal{NZ} = \emptyset$.

**Proposition 4.** *Let $f$ be any signature having imp support and let $\mathcal{F}$ be any signature set. If either $f$ has no affine support or $f \notin \mathcal{ED}$, then $\#\mathrm{CSP}^*(Implies,\mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(f,\mathcal{F})$.*

Finally, we discuss signatures that lack both affine support and imp support. The proof of the following proposition uses Propositions 2 and 3.

**Proposition 5.** *Let $f$ be any signature not in $\mathcal{NZ}$ and let $\mathcal{F}$ be any signature set. If $f$ has neither affine support nor imp support, then* $\#\mathrm{CSP}^*(OR, \mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(f, \mathcal{F})$.

## 9 The Trichotomy Theorem

Our trichotomy theorem states that all counting problems of the form $\#\mathrm{CSP}^*(\mathcal{F})$ can be classified into three categories. This theorem extends an earlier work of Dyer et al. [7] on unweighted Boolean constraints and also gives an approximation version of the dichotomy theorem of Cai et al. [2].

**Theorem 1.** *Let $\mathcal{F}$ be any set of signatures. If either $\mathcal{F} \subseteq \mathcal{AF}$ or $\mathcal{F} \subseteq \mathcal{ED}$, then* $\#\mathrm{CSP}^*(\mathcal{F})$ *is in* $\mathrm{FP}_{\mathbb{C}}$. *Otherwise, if $\mathcal{F} \subseteq \mathcal{IM}$, then* $\#\mathrm{DOWNSET}^*_{\mathbb{C}} \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(\mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{DOWNSET}_{\mathbb{C}}$. *Otherwise,* $\#\mathrm{SAT}^*_{\mathbb{C}} \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(\mathcal{F})$.

Theorem 1 immediately follows from the next proposition, which can be proven by the fundamental properties shown in Sections 5–8.

**Proposition 6.** *Let $f$ be any signature and let $\mathcal{F}$ be any set of signatures. Assume that $f \notin \mathcal{AF} \cup \mathcal{ED}$.*

1. *If $f \in \mathcal{IM}$, then* $\#\mathrm{CSP}^*(Implies, \mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(f, \mathcal{F})$.
2. *If $f \notin \mathcal{IM}$, then* $\#\mathrm{CSP}^*(OR, \mathcal{F}) \leq_{\mathrm{AP}} \#\mathrm{CSP}^*(f, \mathcal{F})$.

## References

1. Cai, J., Lu, P.: Holographic algorithms: from arts to science. In: Proceedings of STOC 2007, pp. 401–410 (2007)
2. Cai, J., Lu, P., Xia, M.: The complexity of complex weighted Boolean #CSP (2009); An early version appeared in Proceedings of STOC 2009, pp. 715–724 (2009)
3. Creignou, N., Hermann, M.: Complexity of generalized satisfiability counting problems. Inform. and Comput. 125, 1–12 (1996)
4. Dyer, M., Goldberg, L.A., Greenhill, C., Jerrum, M.: The relative complexity of approximating counting problems. Algorithmica 38, 471–500 (2004)
5. Dyer, M., Goldberg, L.A., Jalsenius, M., Richerby, D.: The complexity of approximating bounded-degree Boolean #CSP. Available on line at arXiv:0907.2663v1 (2009)
6. Dyer, M., Goldberg, L.A., Jerrum, M.: The complexity of weighted Boolean #CSP. SIAM J. Comput. 38, 1970–1986 (2009)
7. Dyer, M., Goldberg, L.A., Jerrum, M.: An approximation trichotomy for Boolean #CSP. J. Comput. System Sci. 76, 267–277 (2010)
8. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of FOCS 1978, pp. 216–226 (1978)
9. Vadhan, S.P.: The complexity of counting in sparse, regular, and planar graphs. SIAM J. Comput. 31, 398–427 (2001)
10. Yamakami, T.: Approximate counting for complex-weighted Boolean constraint satisfaction problems. Available on line at arXiv:1007.0391 (2010)

# Author Index

# Author Index