Roberto Solis-Oba
Giuseppe Persiano (Eds.)

LNCS 7164

Approximation and Online Algorithms

9th International Workshop, WAOA 2011
Saarbrücken, Germany, September 2011
Revised Selected Papers

Lecture Notes in Computer Science 7164

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

Roberto Solis-Oba Giuseppe Persiano (Eds.)

Approximation and Online Algorithms

9th International Workshop, WAOA 2011
Saarbrücken, Germany, September 8-9, 2011
Revised Selected Papers

Springer

Volume Editors

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

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

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-29115-9 e-ISBN 978-3-642-29116-6
DOI 10.1007/978-3-642-29116-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012934372

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

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

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

Printed on acid-free paper

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

Preface

The 9th Workshop on Approximation and Online Algorithms (WAOA 2011) took place in Saarbrücken, Germany, September 8–9, 2011. The workshop was part of the ALGO 2011 event that also hosted ESA 2011, WABI 2011, IPEC 2011, ALGOSENSORS 2011, and ATMOS 2011. The previous WAOA workshops were held in Budapest (2003), Rome (2004), Palma de Mallorca (2005), Zurich (2006), Eilat (2007), Karlsruhe (2008), Copenhagen (2009), and Liverpool (2010). The proceedings of these previous WAOA workshops have appeared as LNCS volumes 2909, 3351, 3879, 4368, 4927, 5426, 5893, and 6534, respectively.

The Workshop on Approximation and Online Algorithms focuses on the design and analysis of algorithms for online and computationally hard problems. Both kinds of problems have a large number of applications in a wide variety of fields. Topics of interest for WAOA 2011 were: algorithmic game theory, approximation classes, coloring and partitioning, competitive analysis, computational finance, cuts and connectivity, geometric problems, inapproximability results, mechanism design, network design, packing and covering, paradigms for design and analysis of approximation and online algorithms, parameterized complexity, randomization techniques and scheduling problems.

In response to the call for papers, we received 48 submissions. Each submission was reviewed by at least three referees. The submissions were mainly judged on originality, technical quality, and relevance to the topics of the conference. Based on the reviews, the Program Committee selected 21 papers. In addition to the presentations of the 21 accepted papers, Klaus Jansen from the University of Kiel gave an invited talk on "Approximation Algorithms for Scheduling and Packing Problems."

We are grateful to Andrei Voronkov for providing the EasyChair conference system, which was used to manage the electronic submissions and the review process. It made our task much easier. We would also like to thank all the authors who submitted papers to WAOA 2011 as well as the local organizers of ALGO 2011.

November 2011

Roberto Solis-Oba
Giuseppe Persiano

Organization

Program Co-chairs

Roberto Solis-Oba	University of Western Ontario, Canada
Giuseppe Persiano	Università di Salerno, Italy

Program Committee

Vincenzo Auletta	Università di Salerno, Italy
Evripidis Bampis	University of Evry, France
Ioannis Caragiannis	University of Patras, Greece
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
Klaus Jansen	University of Kiel, Germany
Christos Kaklamanis	University of Patras, Greece
Jochen Könemann	University of Waterloo, Canada
Alejandro López-Ortiz	University of Waterloo, Canada
Monaldo Mastrolilli	IDSIA Lugano, Switzerland
Julian Mestre	University of Sydney, Australia
Giuseppe Persiano	(Co-chair), Università di Salerno, Italy
Hadas Shachnai	Technion, Israel
Roberto Solis-Oba	(Co-chair), University of Western Ontario, Canada
Clifford Stein	Columbia University, USA
Denis Trystram	Grenoble Institute of Technology, France
Carmine Ventre	University of Liverpool, UK

Additional Referees

Markus Bläser
Marin Bougeret
Stefan Canzar
Johanne Cohen
Reza Dorrigiv
Ioannis Emiris
Leah Epstein
Cristina Fernandes
Diodato Ferraioli
Robert Fraser
Konstantinos Georgiou
Masud Hasan
Chien-Chung Huang
Sungjin Im
Shahin Kamali
Panagiotis Kanellopoulos
Nikos Karanikolas
Kim Klein
Ephraim Korach
Stefan Kraft
Ariel Kulik
Maria Kyropoulou

Bundit Laekhanukit
Dimitris Letsios
Giorgio Lucarelli
Hamid Mahini
Bodo Manthey
Nicole Megow
Nikolaus Mutsanas
Rajiv Raman
Aris Pagourtzis
Konstantinos Panagiotou
Paolo Penna
Matthias Poloczek
Lars Prädel
Kirk Pruhs
Claude-Guy Quimper
Dror Rawitz
David Rizzuto
Christina Robenek
Alejandro Salinger
Guido Schaefer
Ilka Schnoor
Martin Skutella
Gwen Spencer
Ola Svensson
Chaitanya Swamy
Tami Tamir
Marc Uetz
Anke Van Zuylen
Jose Verschae
Haifeng Xu Lisa Zhang

Table of Contents

Approximation Algorithms for Scheduling and Packing Problems

Klaus Jansen*

Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
kj@informatik.uni-kiel.de

Abstract. In this paper we present an overview about new approximation results for several optimization problems. During the last years we have worked on the design of approximation algorithms with a smaller approximation ratio and on the design of efficient polynomial time approximation schemes with a faster running time. We presented approximation algorithms with a smaller ratio for scheduling with fixed jobs and for two dimensional strip packing. On the other hand, we developed efficient approximation schemes with an improved running time for multiple knapsack and scheduling independent jobs on uniform processors.

1 Introduction

In the first part of the paper we focus on approximation algorithms with good performance guarantees. Let $A(I)$ be the objective value (e.g. the schedule length or total profit) generated by a polynomial time algorithm A, and $OPT(I)$ be the optimal value for an instance I. The approximation ratio R_A of A is $\sup_I \frac{A(I)}{OPT(I)}$ and $sup_I \frac{OPT(I)}{A(I)}$ for a minimization problem and maximization problem, respectively. The goal in both cases is to find an algorithm with minimum ratio R_A.

In the second part we focus on the running time of approximation schemes. A problem admits a polynomial-time approximation scheme (PTAS) if there is a family of algorithms $\{A_\varepsilon \mid \varepsilon > 0\}$ such that for any $\varepsilon > 0$ and any instance I, A_ε produces a $(1+\varepsilon)$-approximate solution in time polynomial in the size of the input. Two important restricted classes of approximation schemes were defined to reduce the running time. An efficient polynomial-time approximation scheme (EPTAS) is a PTAS with running time of the form $f(1/\epsilon)poly(|I|)$, while a fully time polynomial time approximation scheme (FPTAS) runs in time $poly(|I|, 1/\epsilon)$.

There is an interesting connection to parameterized complexity, i.e. to fixed parameter tractable (FPT) algorithms and to the complexity class W[1]. In fact, if the standard parametrization of an optimization problem is W[1]-hard, then the optimization problem does not have an EPTAS, unless FPT=W[1] [2,4]. For a survey on the connection between approximation algorithms and parameterized complexity we refer to [31].

* Research supported by the Deutsche Forschungsgemeinschaft (DFG).

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 1–8, 2012.

2 Scheduling with Fixed Jobs

In parallel machine scheduling, an important issue is the scenario where either some jobs are already fixed in the system [33] or intervals of non-availability of some machines must be taken into account [21]. The first problem occurs when high-priority jobs are already scheduled in the system while the latter problem is due to regular maintenance of machines. Both models are relevant for turnaround scheduling [32] and distributed computing where machines are donated on a volunteer basis.

The problem can be defined as follows: an instance consists of a set $M = \{M_1, \ldots, M_m\}$ of m identical machines and a set $J = \{J_1, \ldots, J_n\}$ of n jobs with non-negative processing times $p_1, \ldots, p_n \in \mathbb{N}$. The first k jobs are fixed via a list $(m_1, s_1), \ldots, (m_k, s_k)$ giving a machine index $m_j \in M$ and starting time $s_j \geq 0$ for the corresponding job J_j, for $j = 1, \ldots, k$. We suppose that these fixed jobs do not overlap. A schedule is a non-preemptive assignment of the jobs to machines and starting times such that the first k jobs are assigned as encoded in the instance and all jobs do not intersect.

For the problem with *fixed jobs*, the objective is to minimize the makespan (the maximum completion time $C_j = s_j + p_j$) among all jobs including the fixed ones; i.e. $C_{max} = \max_{j=1,\ldots,n} C_j$. In the setting with *non-availability*, our goal is to find a schedule with minimum makespan among all non-fixed jobs; i.e. $C_{max} = \max_{j=k+1,\ldots,n} C_j$. Both problems generalize the well-known problem $P||C_{max}$ (scheduling jobs on parallel identical machines to minimize makespan) [18] and hence are strongly NP-hard. Interestingly, the second variant is harder to approximate.

2.1 Related Results

Scheduling with fixed jobs was studied by Scharbrodt, Steger and Weisser [34,33]. They mainly studied the problem for a constant number m of processors. For this strongly NP-hard problem (which consequently does not admit an FPTAS) they presented a polynomial time approximation scheme (PTAS). They also found approximation algorithms for an arbitrary number m of processors with ratios 3 [34] and $2 + \epsilon$ [33]. Furthermore, they [33] proved that there is no approximation algorithm with ratio $3/2 - \epsilon$ for scheduling with fixed jobs for any $\epsilon \in (0, 1/2]$, unless P = NP.

2.2 New Results

Diedrich and Jansen [9] presented a $3/2 + \epsilon$-approximation for both variants. However, the algorithm used on a large number of enumeration steps and involved up to $m^{1/\epsilon^{1/\epsilon^2}}$ calls to a subroutine that approximately solves a maximization problem, the Multiple Subset Sum Problem (MSSP), for a fixed accuracy $\epsilon > 0$. Let $T_{MSSP}(n, \epsilon)$ be the running time of this subroutine.

Recently, we presented improved algorithms for scheduling with fixed jobs and scheduling with non-availability constraints. These algorithms achieve exactly

the bound of 3/2 and are both faster and conceptually simpler than the previous algorithms in [9]. Formally stated our results are the following:

Theorem 1. *[25] Scheduling with fixed jobs admits an approximation algorithm with ratio* 3/2 *and running time*

$$O(n \log n + \log((n/m) \max_{j=1,\ldots,n} p_j)(n + T_{MSSP}(n, 1/8))).$$

For scheduling with non-availability, the result is slightly weaker for technical reasons:

Theorem 2. *[25] Scheduling with non-availability, as long as a constant fraction* ρm *of machines is always available with* $\rho \in (0, 1)$*, admits an approximation algorithm with ratio* 3/2 *and running time*

$$O(n \log n + \log((n/\rho m) \max_{j=1,\ldots,n} p_j)(n + T_{MSSP}(n, \rho/8))).$$

3 2D Strip Packing

Two-dimensional packing problems are classical combinatorial optimization problems. One of the most important ones is the 2D strip packing problem: Given a set of n rectangles $I = \{R_1, \ldots, R_n\}$ of specified widths $w_i \leq 1$ and heights $h_i \leq 1$, the problem is to find a feasible packing for I (i.e. an orthogonal arrangement where rectangles do not overlap and are not rotated) into a strip of width 1 and minimum height. The 2D strip packing problem has many practical applications in manufacturing, logistics, VLSI design, and parallel computing. In many manufacturing settings, rectangular pieces need to be cut out of some sheet of raw material (e.g. textile, glass, or wood), while minimizing the usage or the waste. Scheduling independent tasks on parallel processors, each requiring a certain number of contiguous processors or memory allocation during a certain time interval, can also be modeled as a 2D strip packing problem. Since 2D strip packing includes bin packing as a special case (where each rectangle has height $h_i = 1$), the problem is strongly NP-hard. Therefore, there is no efficient algorithm for constructing an optimal packing, unless P = NP.

3.1 Related Work

Coffman et al. [8] provided the first algorithms Next Fit Decreasing Height (NFDH) and First Fit Decreasing Height (FFDH) with absolute approximation ratios of 3 and 2.7, respectively. The approximation algorithm found by Sleator [37] generates a strip packing of height at most $2.5OPT(I)$. This was independently improved by Schiermeyer [36] and Steinberg [38]. They both presented approximation algorithms with ratio 2. Furthermore, there is no approximation algorithm with absolute ratio better than 1.5, unless P = NP; otherwise we could solve the PARTITION problem in polynomial time.

3.2 New Results

After the work by Steinberg and Schiermeyer, there was no improvement on the best known approximation ratio for more than 14 years. Jansen and Thöle [27] presented an approximation algorithm with approximation ratio $1.5+\varepsilon$ for restricted instances where the widths have the form $\frac{i}{m}$ for $i \in \{1,\ldots,m\}$ and m is polynomially bounded in the number of rectangles or the input size. Notice that the general version appears to be considerably more difficult. Recently, Harren and van Stee [16] broke the barrier of 2 for the general 2D strip packing problem and presented an approximation algorithm with a ratio of 1.9396. Our main new result is the following significant improvement.

Theorem 3. *[15] For any $\varepsilon > 0$, there is an approximation algorithm A which produces a packing of a list I of n rectangles in a strip of width 1 and height $A(I)$ such that*

$$A(I) \leq \left(\frac{5}{3}+\varepsilon\right) OPT(I).$$

The running time of A is polynomial in n for any fixed ε.

4 Multiple Knapsack Problem

The knapsack problem is a fundamental problem in combinatorial optimization. One interesting generalization is the multiple knapsack problem (MKP), in which an instance I is given by a set $\mathcal{A}$ of n items and a set $\mathcal{B}$ of m bins or knapsacks. Each item $a \in \mathcal{A}$ has a size $size(a) \in \mathbb{Q}^+$ and a profit $profit(a) \in \mathbb{Q}^+$, and each bin $b \in \mathcal{B}$ has a capacity or size $c(b) \in \mathbb{Q}^+$. The goal of MKP is to find a subset $S \subseteq \mathcal{A}$ that can be packed into B without exceeding the capacities of the bins and has maximum total profit $profit(S) = \sum_{a \in S} profit(a)$. The maximum total profit among all feasible subsets $S \subseteq \mathcal{A}$ that can be packed into $\mathcal{B}$ is denoted by $OPT(I)$. MKP has many applications in computer science, operations research, and related disciplines; see also the book by Kellerer, Pferschy, and Pisinger [28]. An interesting application is the scheduling problem with fixed jobs described above.

4.1 Known Results

In contrast to the classical knapsack problem, MKP even with two bins with the same capacity does not have a fully polynomial time approximation scheme (FPTAS), unless P=NP [3,5]. Chekuri and Khanna [5] gave a polynomial-time approximation scheme (PTAS) for MKP with general capacities. The running time of their PTAS is $n^{O(1/\epsilon^8 \log(1/\epsilon))}$. Furthermore, they [5] posed the question of whether there is a PTAS with an improved running time and conjectured that an efficient polynomial time approximation scheme (EPTAS) with running time $f(1/\epsilon)poly(n)$ for some function f might be possible. Fellows [11] considered it as a significant open problem to determine whether MKP admits a fixed parameter tractable (FPT) algorithm or it is W[1]-hard.

4.2 New Results

We [22] found an EPTAS for MKP with running time $2^{O(1/\epsilon^5 \log(1/\epsilon))} poly(n)$ (that can be bounded also by $2^{O(1/\epsilon^5 \log(1/\epsilon))} + poly(n)$) answering the open question posed by Chekuri and Khanna in the affirmative. Recently we improved the running time above and obtained the following main result:

Theorem 4. *[24] There is an efficient polynomial-time approximation scheme (EPTAS) for the multiple knapsack problem with running time*

$$2^{O(1/\epsilon \log^4(1/\epsilon))} + poly(n).$$

If the integrality gap between the ILP and LP objective values for the bin packing problem with different bin sizes is bounded by a constant C, similar to the modified round-up conjecture by Scheithauer and Terno [35] (i.e. that $ILP(I) \leq \lceil LP(I) \rceil + 1$ for the ILP and LP formulations for each instance I of the classical bin packing problem), then we can reduce the above running time to

$$2^{O(1/\epsilon \log^2(1/\epsilon))} + poly(n).$$

5 Scheduling on Uniform Processors

Another fundamental problem in scheduling theory is the following. Suppose that we are given a set $\mathcal{J}$ of n independent jobs J_j with processing time p_j and a set $\mathcal{P}$ of m non-identical processors P_i that run at different speeds s_i. If job J_j is executed on processor P_i, the machine needs p_j/s_i time units to complete the job. The problem is to find an assignment $a : \mathcal{J} \to \mathcal{P}$ for the jobs to the processors that minimizes the total execution time $\max_{i=1,\ldots,m} \sum_{J_j : a(J_j)=P_i} p_j/s_i$. This is the minimum time needed to complete the execution of all jobs on the processors. The problem is denoted by $Q||C_{max}$ and it is also called the minimum makespan problem on uniform parallel processors. The problem for uniform (and also identical) processors has been demonstrated to be NP-hard [13], and the existence of a polynomial time algorithm for it would imply $P = NP$.

5.1 Known Results

Most of the work on this fundamental scheduling problem has been done already more than 20 years ago. Horowitz and Sahni [20] proposed an approximation scheme for scheduling on a fixed number m of uniform processors with running time $(n/\epsilon)^{O(m)}$. Gonzales, Ibara, and Sahni [14] analyzed list schedules for uniform processors based on the LPT (longest processing time) rule. They proved that LPT produces a schedule of length between 1.5 and 2 times the optimum. Friesen and Langston [12] analyzed a variant of the MULTIFIT algorithm derived from bin packing and proved that its worst-case performance bound is within 1.4 of the optimum value. The upper bound has been improved to 1.38 by Chen [6].

Hochbaum and Shmoys [18] introduced the dual approximation approach for identical and uniform processors and used the relationship between the scheduling and the bin packing problem. This relationship for scheduling on identical processors had been exploited already by Coffman, Garey, and Johnson [7]. Using the dual approximation approach, Hochbaum and Shmoys [18] proposed a PTAS for scheduling jobs on identical processors with running time $(n/\epsilon)^{O(1/\epsilon^2)}$.

Leung [30] found a PTAS for scheduling on identical processors with improved running time $(n/\epsilon)^{O(1/\epsilon \log(1/\epsilon))}$. Hochbaum and Shmoys (see [17]) and Alon et al. [1] both achieved an improvement by using an integer linear program (ILP) formulation of the bin packing problem for the large items and a result on integer linear programming with a fixed number of variables by Lenstra [29]. This gives an EPTAS for identical processors with running time $f(1/\epsilon) + O(n)$ where f is doubly exponential in $1/\epsilon$.

For uniform processors, the decision problem for the scheduling problem with makespan at most T can be interpreted as a bin packing problem with different bin sizes. Using an ϵ-relaxed version of this bin packing problem, Hochbaum and Shmoys [19] were able to obtain a PTAS for scheduling jobs on uniform processors with running time $(n/\epsilon)^{O(1/\epsilon^2)}$. The existence of an EPTAS for uniform processors was mentioned as an open problem by Epstein and Sgall [10].

5.2 New Results

Our first result, which uses an ILP and MILP formulation with a constant number of integral variables, is the following:

Theorem 5. *[23] There is an EPTAS which, given an instance I of $Q||C_{max}$ with n jobs and m processors and a positive number $\epsilon > 0$, produces a schedule for the jobs of length $A_\epsilon(I) \leq (1+\epsilon)OPT(I)$. The running time of A_ϵ is*

$$2^{O(1/\epsilon^2 \log(1/\epsilon)^3)} + poly(n).$$

Interestingly, the running time of the EPTAS is only single exponential in $1/\epsilon$. Recently, we found an EPTAS for scheduling on uniform processors that avoids the use of an ILP or MILP solver. Furthermore, under the assumption that the distance between an arbitrary LP and closest ILP solution is bounded, we are also able to improve the running time.

Theorem 6. *[26] There is an EPTAS which, given an instance of $Q||C_{max}$ with n jobs and m processors and a positive number $\epsilon > 0$, produces a schedule for the jobs of length $A_\epsilon(I) \leq (1+\epsilon)OPT(I)$. If the maximum distance between an arbitrary LP and closest ILP solution is bounded by $poly(1/\epsilon)$, then the running time of A_ϵ is bounded by*

$$2^{O(1/\epsilon \log^2(1/\epsilon))} + poly(n).$$

References

1. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation schemes for scheduling on parallel machines. Journal on Scheduling 1, 55–66 (1998)
2. Bazgan, C.: Schémas d'approximation et complexité paramétrée. Technical Report, Université Paris-Sud (1995)
3. Caprara, A., Kellerer, H., Pferschy, U.: The multiple subset sum problem. SIAM Journal of Optimization 11, 308–319 (2000)
4. Cesati, M., Trevisan, L.: On the efficiency of polynomial time approximation schemes. Information Processing Letters 64, 165–171 (1997)
5. Chekuri, C., Khanna, S.: A PTAS for the multiple knapsack problem. SIAM Journal on Computing 35, 713–728 (2006)
6. Chen, B.: Tighter bounds for MULTIFIT scheduling on uniform processors. Discrete Applied Mathematics 31, 227–260 (1991)
7. Coffman, E.G., Garey, M.R., Johnson, D.S.: An application of bin packing to multiprocessor scheduling. SIAM Journal on Computing 7, 1–17 (1978)
8. Coffman, E.G., Garey, M.R., Johnson, D.S., Tarjan, R.E.: Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms. SIAM Journal on Computing 4, 808–826 (1980)
9. Diedrich, F., Jansen, K.: Improved approximation algorithms for scheduling with fixed jobs. In: Proceedings of Symposium on Discrete Algorithms, SODA, pp. 675–684 (2009)
10. Epstein, L., Sgall, J.: Approximation schemes for scheduling on uniformly related and identical parallel machines. Algorithmica 39, 43–57 (2004)
11. Fellows, M.R.: Blow-Ups, Win/Win's, and Crown Rules: Some New Directions in *FPT*. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 1–12. Springer, Heidelberg (2003)
12. Friesen, D.K., Langston, M.A.: Bounds for multifit scheduling on uniform processors. SIAM Journal on Computing 12, 60–70 (1983)
13. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman, San Francisco (1979)
14. Gonzales, T., Ibarra, O.H., Sahni, S.: Bounds for LPT schedules on uniform processors. SIAM Journal on Computing 6, 155–166 (1977)
15. Harren, R., Jansen, K., Prädel, L., van Stee, R.: A $(5/3 + \epsilon)$-Approximation for Strip Packing. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 475–487. Springer, Heidelberg (2011)
16. Harren, R., van Stee, R.: Improved Absolute Approximation Ratios for Two-Dimensional Packing Problems. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 177–189. Springer, Heidelberg (2009)
17. Hochbaum, D.S.: Various notions of approximations: good, better, best, and more. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-Hard Problems, Ch. 9, pp. 346–398. Prentice Hall (1997)
18. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: practical and theoretical results. Journal of the ACM 34, 144–162 (1987)
19. Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM Journal on Computing 17, 539–551 (1988)
20. Horowitz, R., Sahni, S.: Exact and approximate algorithms for scheduling nonidentical processors. Journal of the ACM 23, 317–327 (1976)

21. Hwang, H.-C., Lee, K., Chang, S.Y.: The effect of machine availability on the worst-case performance of LPT. Discrete Applied Mathematics 148, 49–61 (2005)
22. Jansen, K.: Parameterized approximation scheme for the multiple knapsack problem. SIAM Journal on Computing 39, 1392–1412 (2009)
23. Jansen, K.: An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. SIAM Journal on Discrete Mathematics 24, 457–485 (2010)
24. Jansen, K.: A fast approximation scheme for the multiple knapsack problem. In: Proceedings of Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2012 (to appear, 2012)
25. Jansen, K., Prädel, L., Schwarz, U.M., Svensson, O.: Faster approximation algorithms for scheduling with fixed jobs. In: Proceedings of Conference of Computing: the Australasian Theory Symposium, CATS 2011, pp. 3–9 (2011)
26. Jansen, K., Robenek, C.: Scheduling Jobs on Identical and Uniform Processors Revisited. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 109–122. Springer, Heidelberg (2012)
27. Jansen, K., Thöle, R.: Approximation algorithms for scheduling parallel jobs. SIAM Journal on Computing 39, 3571–3615 (2010)
28. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin (2004)
29. Lenstra, H.W.: Integer programming with a fixed number of variables. Mathematics of Operations Research 8, 538–548 (1983)
30. Leung, J.: Bin packing with restricted piece sizes. Information Processing Letters 31, 145–149 (1989)
31. Marx, D.: Parameterized complexity and approximation algorithms. The Computer Journal 51, 60–78 (2008)
32. Megow, N., Möhring, R.H., Schulz, J.: Decision Support and Optimization in Shutdown and Turnaround Scheduling. Informs Journal on Computing 23, 189–204 (2011)
33. Scharbrodt, M., Steger, A., Weisser, H.: Approximability of scheduling with fixed jobs. Journal of Scheduling 2, 267–284 (1999)
34. Scharbrodt, M.: Produktionsplanung in der Prozessindustrie: Modelle, effiziente Algorithmen und Umsetzung, Dissertation, Fakultät für Informatik, Technische Universität München (2000)
35. Scheithauer, G., Terno, J.: Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. European Journal of Operational Research 20, 93–100 (1997)
36. Schiermeyer, I.: Reverse-Fit: A 2-Optimal Algorithm for Packing Rectangles. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 290–299. Springer, Heidelberg (1994)
37. Sleator, D.D.: A 2.5 times optimal algorithm for packing in two dimensions. Information Processing Letters 10, 37–40 (1980)
38. Steinberg, A.: A Strip-Packing Algorithm with Absolute Performance Bound 2. SIAM Journal on Computing 2, 401–409 (1997)

Approximating Subset k-Connectivity Problems

Zeev Nutov

The Open University of Israel
nutov@openu.ac.il

Abstract. A subset $T \subseteq V$ of terminals is k-connected to a root s in a directed/undirected graph J if J has k internally-disjoint vs-paths for every $v \in T$; T is k-connected in J if T is k-connected to every $s \in T$. We consider the Subset k-Connectivity Augmentation problem: given a graph $G = (V, E)$ with edge/node-costs, a node subset $T \subseteq V$, and a subgraph $J = (V, E_J)$ of G such that T is $(k-1)$-connected in J, find a minimum-cost augmenting edge-set $F \subseteq E \setminus E_J$ such that T is k-connected in $J \cup F$. The problem admits trivial ratio $O(|T|^2)$. We consider the case $|T| > k$ and prove that for directed/undirected graphs and edge/node-costs, a ρ-approximation algorithm for Rooted Subset k-Connectivity Augmentation implies the following approximation ratios for Subset k-Connectivity Augmentation:

(i) $b(\rho + k) + \left(\frac{|T|}{|T|-k}\right)^2 O\left(\log \frac{|T|}{|T|-k}\right)$ and
(ii) $\rho \cdot O\left(\frac{|T|}{|T|-k} \log k\right)$,

where $b = 1$ for undirected graphs and $b = 2$ for directed graphs. The best known values of ρ on undirected graphs are $\min\{|T|, O(k)\}$ for edge-costs and $\min\{|T|, O(k \log |T|)\}$ for node-costs; for directed graphs $\rho = |T|$ for both versions. Our results imply that unless $k = |T| - o(|T|)$, Subset k-Connectivity Augmentation admits the same ratios as the best known ones for the rooted version. This improves the ratios in [19,14].

1 Introduction

In the Survivable Network problem we are given a graph $G = (V, E)$ with edge/node-costs and pairwise connectivity requirements $\{r(u, v) : u, v \in T \subseteq V\}$ on a node subset T of terminals. The goal is to find a minimum-cost subgraph of G that contains $r(u, v)$ internally-disjoint uv-paths for all $u, v \in T$. In the node-costs version of Survivable Network, we seek a min-cost node subset $U \subseteq V \setminus T$ such that the graph induced by $T \cup U$ in G satisfies the connectivity requirements.

In the Rooted Subset k-Connectivity problem there is $s \in T$ such that $r(s, t) = k$ for all $t \in T \setminus \{s\}$ and $r(u, v) = 0$ otherwise. In the Subset k-Connectivity problem $r(u, v) = k$ for all $u, v \in T$ and $r(u, v) = 0$ otherwise. In the *augmentation versions*, G contains a subgraph J of cost zero with $r(u, v) - 1$ internally disjoint paths for all $u, v \in T$. A subset $T \subseteq V$ of terminals is k-connected to a root s in a directed/undirected graph J if J has k internally-disjoint vs-paths for every $v \in T$; T is k-connected in J if T is k-connected to every $s \in T$. Formally, the versions of Survivable Network we consider are as follows.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 9–20, 2012.

Rooted Subset k-Connectivity Augmentation
Instance: A graph $G=(V,E)$ with edge/node-costs, a set $T \subseteq V$ of terminals, a root $s \in T$, and a subgraph $J=(V,E_J)$ of G such that $T \setminus \{s\}$ is $(k-1)$-connected to s in J.
Objective: Find a minimum-cost edge-set $F \subseteq E \setminus E_J$ such that $T \setminus \{s\}$ is k-connected to s in $J \cup F$.

Subset k-Connectivity Augmentation
Instance: A graph $G=(V,E)$ with edge/node-costs, a set $T \subseteq V$ of terminals, and a subgraph $J=(V,E_J)$ of G such that T is $(k-1)$-connected in J.
Objective: Find a minimum-cost edge-set $F \subseteq E \setminus E_J$ such that T is k-connected in $J \cup F$.

Subset k-Connectivity Augmentation is Label-Cover hard to approximate [9], and Rooted k-Connectivity Augmentation is harder than the Directed Steiner Tree problem [15]; hence obtaining a polylogarithmic approximation ratio for these problems is unlikely. It is known and easy to see that for both edge-costs and node-costs, if Subset k-Connectivity Augmentation admits approximation ratio $\rho(k)$ such that $\rho(k)$ is a monotone increasing function, then Subset k-Connectivity admits ratio $k \cdot \rho(k)$. Moreover, for edge costs, if in addition the approximation ratio $\rho(k)$ is w.r.t. a standard setpair/biset LP-relaxation to the problem, then Subset k-Connectivity admits ratio $H(k) \cdot \rho(k)$, where $H(k)$ denotes the kth harmonic number; see [19,14] for details. For edge-costs, a standard setpair LP-relaxation for Survivable Network (due to Frank and Jordán [5]) is:

$$\min \left\{ \sum_{e \in E} c_e x_e : \sum_{e \in E(X,X^*)} x_e \geq r(X,X^*), X, X^* \subseteq V, X \cap X^* = \emptyset, 0 \leq x_e \leq 1 \right\}$$

where $r(X,X^*) = \max\{r(u,v) : u \in X, v \in X^*\} - |V \setminus (X \cup X^*)|$ and $E(X,X^*)$ is the set of edges in E from X to X^*.

Subset k-Connectivity admits trivial ratios $O(|T|^2)$ for both edge-costs and node-costs, by computing for every $u,v \in V$ an optimal edge-set of k internally-disjoint uv-paths (this is essentially a Min-Cost k-Flow problem, that can be solved in polynomial time), and taking the union of the computed edge-sets. For metric edge-costs the problem admits an $O(1)$ ratio [2]. For $|T| \geq k+1$ the problem can also be decomposed into k instances of Rooted Subset k-Connectivity problems, c.f. [11] for the case $T=V$, where it is also shown that for $T=V$ the number of Rooted Subset k-Connectivity Augmentation instances can be reduced to $O\left(\frac{|T|}{|T|-k} \log k\right)$, which is $O(\log k)$ unless $k = |T| - o(|T|)$.

Recently, Laekhanukit [14] made an important observation that the method of [11] can be extended for the case of arbitrary $T \subseteq V$. Specifically, he proved that if $|T| \geq 2k$, then $O(\log k)$ instances of Rooted Subset k-Connectivity Augmentation will suffice. Thus for $|T| \geq 2k$, the $O(k)$-approximation algorithm of

[19] for Rooted Subset k-Connectivity Augmentation leads to the ratio $O(k \log k)$ for Rooted Subset k-Connectivity. By exploiting additional properties of the algorithm of [19], he reduced the ratio to $O(k)$ in the case $|T| \geq k^2$.

Using a different approach, we will show by a much simpler proof, that for both directed and undirected graphs and edge-costs and node-costs, Subset k-Connectivity Augmentation can be reduced to solving *one* instance (or two instances, in the case of directed graphs) of Rooted Subset k-Connectivity Augmentation and $O(k) + \left(\frac{3|T|}{|T|-k}\right)^2 H\left(\frac{3|T|}{|T|-k}\right)$ instances of the Min-Cost k-Flow problem. This leads to a much simpler algorithm, improves the result of Laekhanukit [14] for $|T| < k^2$, and applies also for node-costs and directed graphs. In addition, we give a more natural and much simpler extension of the algorithm of [11] for $T = V$, that also enables the same bound $O\left(\frac{|T|}{|T|-k} \log k\right)$ as in [11] for arbitrary T with $|T| \geq k+1$, and in addition applies also for directed graphs, for node-costs, and for an arbitrary type of edge-costs, e.g., metric costs, or uniform costs, or $0,1$-costs. When we say "$0,1$-edge-costs" we mean that the input graph G is complete, and the goal is to add to the subgraph J of G formed by the zero-cost edges a minimum size edge-set F (any edge is allowed) such that $J \cup F$ satisfies the connectivity requirements. Formally, our result is the following.

Theorem 1. *For both directed and undirected graphs, and edge-costs and node-costs the following holds. If* Rooted Subset k-Connectivity Augmentation *admits approximation ratio $\rho = \rho(k, |T|)$, then for $|T| \geq k+1$* Subset k-Connectivity Augmentation *admits the following approximation ratios:*

(i) $b(\rho + k) + \left(\frac{|T|}{|T|-k}\right)^2 O\left(\log \frac{|T|}{|T|-k}\right)$, *where $b = 1$ for undirected graphs and $b = 2$ for directed graphs.*

(ii) $\rho \cdot O\left(\frac{|T|}{|T|-k} \log \min\{k, |T|-k\}\right)$, *and this is so also for $0,1$-edge-costs.*

Furthermore, for edge-costs, if the approximation ratio ρ is w.r.t. the setpair LP-relaxation for the problem, then so are the ratios in (i) *and* (ii).

For $|T| > k$, the best known values of ρ on undirected graphs are $O(k)$ for edge-costs and $\min\{O(k \log |T|), |T|\}$ for node-costs [19]; for directed graphs $\rho = |T|$ for both versions. For $0,1$-edge-costs $\rho = O(\log k)$ for undirected graphs [20] and $\rho = O(\log |T|)$ for directed graphs [18]. For edge-costs, these ratios are w.r.t. a standard LP-relaxation. Thus Theorem 1 implies the following.

Corollary 1. *For $|T| \geq k+1$,* Subset k-Connectivity Augmentation *admits the following approximation ratios.*

- *For undirected graphs, the ratios are $O(k) + \left(\frac{|T|}{|T|-k}\right)^2 O\left(\log \frac{|T|}{|T|-k}\right)$ for edge-costs, $O(k \log |T|) + \left(\frac{|T|}{|T|-k}\right)^2 O\left(\log \frac{|T|}{|T|-k}\right)$ for node-costs, and $\frac{|T|}{|T|-k} \cdot O\left(\log^2 k\right)$ for $0,1$-edge-costs.*

– *For directed graphs, the ratio is* $2(|T|+k)+\left(\frac{|T|}{|T|-k}\right)^2 O\left(\log \frac{|T|}{|T|-k}\right)$ *for both edge-costs and node-costs, and* $\frac{|T|}{|T|-k}\cdot O\left(\log |T| \log k\right)$ *for* $0,1$ *edge-costs.*

For Subset k-Connectivity*, the ratios are larger by a factor of* $O(\log k)$ *for edge-costs, and by a factor of* k *for node-costs.*

Note that except the case of $0,1$-edge-costs, Corollary 1 is deduced from part (i) of Theorem 1. However, part (ii) of Theorem 1 might become relevant if Rooted Subset k-Connectivity Augmentation admits ratio better than $O(k)$. In addition, part (ii) applies for *any type* of edge-costs, e.g. metric or $0,1$-edge-costs.

We conclude this section by mentioning some additional related work. The case $T=V$ of Rooted Subset k-Connectivity problem is the k-Outconnected Subgraph problem; this problem admits a polynomial time algorithm for directed graphs [6], which implies ratio 2 for undirected graphs. For arbitrary T, the problem is at least as hard as the Directed Steiner Tree problem [15]. The case $T=V$ of Subset k-Connectivity problem is the k-Connected Subgraph problem. This problem is NP-hard, and the best known ratio for it is $O\left(\log k \log \frac{n}{n-k}\right)$ for both directed and undirected graphs [17]; for the augmentation version of increasing the connectivity by one the ratio in [17] is $O\left(\log \frac{n}{n-k}\right)$. For metric costs the problem admits ratios $2+\frac{k-1}{n}$ for undirected graphs and $2+\frac{k}{n}$ for directed graphs [10]. For $0,1$-edge-costs the problem is solvable for directed graphs [5], which implies ratio 2 for undirected graphs. The Survivable Network problem is Label-Cover hard [9], and the currently best known non-trivial ratios for it on undirected graphs are: $O(k^3 \log |T|)$ for arbitrary edge-costs by Chuzhoy and Khanna [3], $O(\log k)$ for metric costs due to Cheriyan and Vetta [2], $O(k)\cdot \min\left\{\log^2 k, \log |T|\right\}$ for $0,1$-edge-costs [20,13], and $O(k^4 \log^2 |T|)$ for node-costs [19].

2 Proof of Theorem 1

We start by proving the following essentially known statement.

Proposition 1. *For both directed and undirected graphs, and edge-costs and node-costs the following holds. Suppose that* Rooted Subset k-Connectivity Augmentation *admits an approximation ratio* ρ*. If for an instance of* Subset k-Connectivity Augmentation *we are given a set of* q *edges (when any edge is allowed) and* p *stars (may be directed to or from the root, in the case of directed graphs) on* T *whose addition to* G *makes* T k*-connected, then we can compute a* $(\rho p+q)$*-approximate solution* F *to this instance in polynomial time. Furthermore, for edge-costs, if the* ρ*-approximation is w.r.t. a standard setpair LP-relaxation, then* $c(F)\leq(\rho p+q)\tau^*$*, where* τ^* *is an optimal setpair LP-relaxation value for* Subset k-Connectivity Augmentation.

Proof. For every edge uv among the q edges, compute a min-cost edge-set $F_{uv}\subseteq E\setminus E_J$ such that $J\cup F_{uv}$ contains k internally-disjoint uv-paths. This can be

done in polynomial time for both edge and node costs, using a Min-Cost k-Flow algorithm. For edge-costs, it is known that $c(F_{uv}) \leq \tau^*$. Then replace uv by F_{uv}, and note that T remains k-connected. Similarly, for every star S with center s and leaf-set T', compute a ρ-approximate augmenting edge-set $F_S \subseteq E \setminus E_J$ such that $J \cup F_S$ contains k internally-disjoint sv-paths (or vs-paths, in the case of directed graphs and S being directed towards the root) for every $v \in T'$. Then replace S by F_S, and note that T remains k-connected. For edge-costs, it is known that if the ρ-approximation for the rooted version is w.r.t. the standard setpair LP-relaxation, then $c(F_S) \leq (\rho p + q)\tau^*$. The statement follows. □

Motivated by Proposition 1, we consider the following question:

Given a $(k-1)$-connected subset T in a graph J, how many edges and/or stars on T one needs to add to J such that T will become k-connected?

We emphasize that we are interested in obtaining *absolute bounds* on the number of edges in the question, expressed in certain parameters of the graph; namely we consider the *extremal graph theory* question and not the *algorithmic problem*. Indeed, the algorithmic problem of adding the minimum number of edges on T such that T will become k-connected can be shown to admit a polynomial-time algorithm for directed graphs using the result of Frank and Jordán [5]; this also implies a 2-approximation algorithm for undirected graphs (for $T = V$, the undirected problem admits a polynomial time algorithm by Vegh [21]). However, in terms of the parameters $|T|, k$, the result in [5] implies only the trivial bound $O(|T|^2)$ on the the number of edges one needs to add to J such that T will become k-connected.

Our bounds will be derived in terms of the family of the "deficient" sets of the graph J. We need some definitions to state our results.

Definition 1. *An ordered pair $\hat{X} = (X, X^+)$ of subsets of a groundset V is called a* biset *if $X \subseteq X^+$; X is the* inner part *and X^+ is the* outer part *of $\hat{X}$, $\Gamma(\hat{X}) = X^+ \setminus X$ is the boundary of $\hat{X}$, and $X^* = V \setminus X^+$ is the* complementary set *of $\hat{X}$.*

Given an instance of directed/undirected Subset k-Connectivity Augmentation we may assume that T is an independent set in J. Otherwise, we obtain an equivalent instance by subdividing every edge $uv \in J$ with $u, v \in T$ by a new node, c.f. [19].

Definition 2. *Given a $(k-1)$-connected independent set T in a directed/undirected graph $J = (V, E_J)$ let us say that a biset $\hat{X}$ on V is is tight in J, if the following holds: $X \cap T, X^* \cap T \neq \emptyset$, X^+ is the union of X and the set of neighbors of X in J, and $|\Gamma(\hat{X})| = k - 1$.*

A directed/undirected edge covers a biset $\hat{X}$ if it goes from X to X^*. By Menger's Theorem, F is a feasible solution to Subset k-Connectivity Augmentation if, and only if, F covers the biset-family $\mathcal{F}$ of tight bisets; see [5,12,20]. Thus denoting $\ell = k - 1$, our question can be reformulated as follows.

Given an ℓ-connected independent set T in a directed/undirected graph J, how many edges and/or stars on T are needed to cover the family $\mathcal{F}$ of tight bisets?

Definition 3. *The intersection and the union of two bisets $\hat{X}, \hat{Y}$ is defined by $\hat{X} \cap \hat{Y} = (X \cap Y, X^+ \cap Y^+)$ and $\hat{X} \cup \hat{Y} = (X \cup Y, X^+ \cup Y^+)$. Two bisets $\hat{X}, \hat{Y}$* intersect *if $X \cap Y \neq \emptyset$; if in addition $X^* \cap Y^* \neq \emptyset$ then $\hat{X}, \hat{Y}$* cross. *We say that a biset-family $\mathcal{F}$ on T is:*

- crossing *if $\hat{X} \cap \hat{Y}, \hat{X} \cup \hat{Y} \in \mathcal{F}$ for any $\hat{X}, \hat{Y} \in \mathcal{F}$ that cross.*
- ℓ-regular *if $|\Gamma(\hat{X})| \leq \ell$ for every $\hat{X} \in \mathcal{F}$, and if $\hat{X} \cap \hat{Y}, \hat{X} \cup \hat{Y} \in \mathcal{F}$ for any intersecting $\hat{X}, \hat{Y} \in \mathcal{F}$ with $|X \cup Y| \leq |T| - \ell - 1$.*

The following statement is essentially known.

Lemma 1. *Let T be an ℓ-connected independent set in a directed/undirected graph J, and let $\hat{X}, \hat{Y}$ be tight bisets. If the bisets $(X \cap T, X^+ \cap T), (Y \cap T, Y^+ \cap T)$ cross, or if $|(X \cup Y) \cap T| \leq |T| - \ell - 1$, then $\hat{X} \cap \hat{Y}, \hat{X} \cup \hat{Y}$ are both tight.*

Proof. The case when $(X \cap T, X^+ \cap T), (Y \cap T, Y^+ \cap T)$ cross was proved in [20] and in [14]. The proof of the case $|(X \cup Y) \cap T| \leq |T| - \ell - 1$ is identical to the proof of [7, Lemma 1.2] where the case $T = V$ is considered. □

Corollary 2. *Let T be an ℓ-connected independent set in a directed/undirected graph $J = (V, E_J)$. Then the biset-family*

$$\mathcal{F} = \{(X \cap T, X^+ \cap T) : (X, X^+) \text{ is a tight biset in } J\}$$

is crossing and ℓ-regular, and the reverse family $\bar{\mathcal{F}} = \{(T \setminus X^+, T \setminus X) : \hat{X} \in \mathcal{F}\}$ *of $\mathcal{F}$ is also crossing and k-regular. Furthermore, if J is undirected then $\mathcal{F}$ is symmetric, namely, $\mathcal{F} = \bar{\mathcal{F}}$.*

Given two bisets $\hat{X}, \hat{Y}$ we write $\hat{X} \subseteq \hat{Y}$ and say that $\hat{Y}$ contains $\hat{X}$ if $X \subseteq Y$ or if $X = Y$ and $X^+ \subseteq Y^+$; $\hat{X} \subset \hat{Y}$ and $\hat{Y}$ properly contains $\hat{X}$ if $X \subset Y$ or if $X = Y$ and $X^+ \subset Y^+$.

Definition 4. *A biset $\hat{C}$ is a* core *of a biset-family $\mathcal{F}$ if $\hat{C} \in \mathcal{F}$ and $\hat{C}$ contains no biset in $\mathcal{F} \setminus \{\hat{C}\}$; namely, a core is an inclusion-minimal biset in $\mathcal{F}$. Let $\mathcal{C}(\mathcal{F})$ be the family of cores of $\mathcal{F}$.*

Given a biset-family $\mathcal{F}$ and an edge-set I on T, the residual biset-family $\mathcal{F}_I$ of $\mathcal{F}$ consists of the members of $\mathcal{F}$ uncovered by I. We will assume that for any I, the cores of $\mathcal{F}_I$ and of $\bar{\mathcal{F}}_I$ can be computed in polynomial time. For $\mathcal{F}$ being the family of tight bisets this can be implemented in polynomial time using the Ford-Fulkerson Max-Flow Min-Cut algorithm, c.f. [20]. It is known and easy to see (c.f. [17]), that if $\mathcal{F}$ is crossing and/or ℓ-regular, so is $\mathcal{F}_I$, for any edge-set I.

Definition 5. *For a biset-family $\mathcal{F}$ on T let $\nu(\mathcal{F})$ denote the maximum number of bisets in $\mathcal{F}$ which inner parts are pairwise-disjoint. For an integer ℓ let $\mathcal{F}^\ell = \{\hat{X} \in \mathcal{F} : |X| \leq (|T| - \ell)/2\}$.*

Lemma 2. *Let $\mathcal{F}$ be an ℓ-regular biset-family on T and let $\hat{X}, \hat{Y} \in \mathcal{F}^\ell$ intersect. Then $\hat{X} \cap \hat{Y} \in \mathcal{F}^\ell$ and $\hat{X} \cup \hat{Y} \in \mathcal{F}$.*

Proof. Since $|X|, |Y| \leq \frac{|T|-\ell}{2}$, we have $|X \cup Y| = |X| + |Y| - |X \cap Y| \leq |T| - \ell - 1$. Thus $\hat{X} \cap \hat{Y}, \hat{X} \cap \hat{Y} \in \mathcal{F}$, by the ℓ-regularity of $\mathcal{F}$. Moreover, $\hat{X} \cap \hat{Y} \in \mathcal{F}^\ell$, since $|X \cap Y| \leq |X| \leq \frac{|T|-\ell}{2}$. □

Let $H(k)$ denote the kth Harmonic number. We prove the following two theorems that imply Theorem 1.

Theorem 2. *Let $\mathcal{F}$ be a biset-family on T such that both $\mathcal{F}, \bar{\mathcal{F}}$ are crossing and ℓ-regular. Then there exists a polynomial-time algorithm that computes an edge-cover I of $\mathcal{F}$ of size $|I| = \nu\left(\mathcal{F}^\ell\right) + \nu\left(\bar{\mathcal{F}}^\ell\right) + \left(\frac{3|T|}{|T|-\ell}\right)^2 H\left(\frac{3|T|}{|T|-\ell}\right)$. Furthermore, if $\mathcal{F}$ is symmetric then $|I| = \nu\left(\mathcal{F}^\ell\right) + \left(\frac{3|T|}{|T|-\ell}\right)^2 H\left(\frac{3|T|}{|T|-\ell}\right)$.*

Theorem 3. *Let $\mathcal{F}$ be a biset-family on T such that both $\mathcal{F}$ and $\bar{\mathcal{F}}$ are ℓ-regular. Then there exists a collection of $O\left(\frac{|T|}{|T|-\ell} \lg \min\{\nu, |T| - \ell\}\right)$ stars on T which union covers $\mathcal{F}$, and such a collection can be computed in polynomial time. Furthermore, the total number of edges in the stars is at most $\nu\left(\mathcal{F}^\ell\right) + \nu\left(\bar{\mathcal{F}}^\ell\right) + \left(\frac{|T|}{|T|-\ell}\right)^2 \cdot O\left(\log \frac{|T|}{|T|-\ell}\right)$.*

Note that the second statement in Theorem 3 implies (up to constants) the bound in Theorem 2. However, the proof of Theorem 2 is much simpler than the proof of Theorem 3, and the proof of Theorem 2 is a part of the proof of the second statement in Theorem 3.

Let us show that Theorems 2 and 3 imply Theorem 1. For that, all we need is to show that by applying b times the ρ-approximation algorithm for the Rooted Subset k-Connectivity Augmentation, we obtain an instance with $\nu\left(\mathcal{F}^\ell\right), \nu\left(\bar{\mathcal{F}}^\ell\right) \leq k$, where $\mathcal{F}$ is the family of tight sets and $\ell = k - 1$. This is achieved by the following procedure due to Khuller and Raghavachari [8] that originally considered the case $T = V$, see also [1,4,10]; the same procedure is also used by Laekhanukit in [14]. Choose an arbitrary subset $T' \subseteq T$ of k nodes, add a new node s (the root) and all edges between s and T' of cost zero each, both to G and to J. Then, using the ρ-approximation algorithm for the Rooted Subset k-Connectivity Augmentation problem, compute an augmenting edge set F such that $J \cup F$ contains k internally disjoint vs-paths and sv-paths for every $v \in T'$. Now, add F to J and remove s from J. It is a routine to show that $c(F) \leq b\mathsf{opt}$, and that for edge-costs $c(F) \leq b\tau^*$. It is also known that if $\hat{X}$ is a tight biset of the obtained graph J, then $X \cap T', X^* \cap T' \neq \emptyset$, c.f. [1,14]. Combined with Lemma 2 we obtain that $\nu\left(\mathcal{F}^\ell\right), \nu\left(\bar{\mathcal{F}}^\ell\right) \leq |T'| \leq k$ for the obtained instance, as claimed.

3 Proof of Theorem 2

Definition 6. *Given a biset-family $\mathcal{F}$ on T, let $\Delta(\mathcal{F})$ denote the maximum degree in the hypergraph $\mathcal{F}^{in} = \{X : \hat{X} \in \mathcal{F}\}$ of the inner parts of the bisets in*

$\mathcal{F}$. We say that $T' \subseteq T$ is a transversal *of $\mathcal{F}$ if $T' \cap X \neq \emptyset$ for every $X \in \mathcal{F}^{in}$; a function $t : T \to [0,1]$ is a* fractional transversal *of $\mathcal{F}$ if $\sum_{v \in X} t(v) \geq 1$ for every $X \in \mathcal{F}^{in}$.*

Lemma 3. *Let $\mathcal{F}$ be a crossing biset-family. Then $\Delta(\mathcal{C}(\mathcal{F})) \leq \nu\left(\bar{\mathcal{F}}\right)$.*

Proof. Since $\mathcal{F}$ is crossing, the members of $\mathcal{C}(\mathcal{F})$ are pairwise non-crossing. Thus if $\mathcal{H}$ is a subfamily of $\mathcal{C}(\mathcal{F})$ such that the intersection of the inner parts of the bisets in $\mathcal{H}$ is non-empty, then $\bar{\mathcal{H}}$ is a subfamily of $\bar{\mathcal{F}}$ such that the inner parts of the bisets in $\bar{\mathcal{H}}$ are pairwise disjoint, so $|\bar{\mathcal{H}}| \leq \nu\left(\bar{\mathcal{F}}\right)$. □

Lemma 4. *Let T' be a transversal of a biset-family $\mathcal{F}'$ on T and let I' be an edge-set on T obtained by picking for every $s \in T'$ an edge from s to every inclusion-minimal member of the set-family $\{X^* : \hat{X} \in \mathcal{F}', s \in X\}$. Then I' covers $\mathcal{F}'$. Moreover, if $\mathcal{F}'$ is crossing then $|I'| \leq |T'| \cdot \nu(\bar{\mathcal{F}}')$.*

Proof. The statement that I' covers $\mathcal{F}'$ is obvious. If $\mathcal{F}'$ is crossing, then for every $s \in T$ the inclusion-minimal members of $\{X^* : \hat{X} \in \mathcal{F}', s \in X\}$ are pairwise-disjoint, hence their number is at most $\nu(\bar{\mathcal{F}}')$. The statement follows. □

Lemma 5. *Let $\mathcal{F}$ be an ℓ-regular biset-family on T. Then the following holds.*

(i) $\nu(\mathcal{F}) \leq \nu\left(\mathcal{F}^{\ell}\right) + \frac{2|T|}{|T|-\ell}$.

(ii) *If $\nu\left(\mathcal{F}^{\ell}_{\{e\}}\right) = \nu\left(\mathcal{F}^{\ell}\right)$ holds for every edge e on T then $\nu\left(\mathcal{F}^{\ell}\right) \leq \frac{|T|}{|T|-\ell}$.*

(iii) *There exists a polynomial time algorithm that finds a transversal T' of $\mathcal{C}(\mathcal{F})$ of size at most $|T'| \leq \left(\nu\left(\mathcal{F}^{\ell}\right) + \frac{2|T|}{|T|-\ell}\right) \cdot H(\Delta(\mathcal{C}(\mathcal{F})))$.*

Proof. Part (i) is immediate.

We prove (ii). Let $\hat{C} \in \mathcal{C}\left(\mathcal{F}^{\ell}\right)$ and let $\hat{U}_C$ be the union of the bisets in $\mathcal{F}^{\ell}$ that contain $\hat{C}$ and contain no other member of $\mathcal{C}\left(\mathcal{F}^{\ell}\right)$. If $|U_C| \leq |T| - \ell - 1$ then $\hat{U}_C \in \mathcal{F}$, by the ℓ-regularity of $\mathcal{F}$. In this case $\nu\left(\mathcal{F}^{\ell}_{\{e\}}\right) \leq \nu\left(\mathcal{F}^{\ell}\right) - 1$ for any edge e from C to U^*_C. Hence $|U_C| \geq |T| - \ell$ must hold for every $\hat{C} \in \mathcal{C}(\mathcal{F})$. By Lemma 2, the sets in the set-family $\{U_C : \hat{C} \in \mathcal{C}(\mathcal{F})\}$ are pairwise disjoint. The statement follows.

We prove (iii). Let T^{ℓ} be an inclusion-minimal transversal of $\mathcal{F}^{\ell}$. By Lemma 2, $\left|T^{\ell}\right| = \nu\left(\mathcal{F}^{\ell}\right)$. Setting $t(v) = 1$ if $v \in T^{\ell}$ and $t(v) = \frac{2}{|T|-\ell}$ otherwise, we obtain a fractional transversal of $\mathcal{C}(\mathcal{F})$ of value at most $\nu\left(\mathcal{F}^{\ell}\right) + \frac{2|T|}{|T|-\ell}$. Consequently, the greedy algorithm of Lovász [16] finds a transversal T' as claimed. □

The algorithm for computing I as in Theorem 2 starts with $I = \emptyset$ and then continues as follows.

Phase 1
While there exists an edge e on T such that $\nu\left(\mathcal{F}^{\ell}_{I \cup \{e\}}\right) \leq \nu\left(\mathcal{F}^{\ell}_{I}\right) - 1$, or such that $\nu\left(\bar{\mathcal{F}}^{\ell}_{I \cup \{e\}}\right) \leq \nu\left(\bar{\mathcal{F}}^{\ell}_{I}\right) - 1$, add e to I.

Phase 2
Find a transversal T' of $\mathcal{C}(\mathcal{F}')$ as in Lemma 5(iii), where $\mathcal{F}' = \mathcal{F}_I$. Then find an edge-cover I' of $\mathcal{F}'$ as in Lemma 4 and add I' to I.

The edge-set I computed covers $\mathcal{F}$ by Lemma 4. Clearly, the number of edges in I at the end of Phase 1 is at most $\nu\left(\mathcal{F}^\ell\right) + \nu\left(\bar{\mathcal{F}}^\ell\right)$, and is at most $\nu\left(\mathcal{F}^\ell\right)$ if $\mathcal{F}$ is symmetric. Now we bound the size of I'. Note that at the end of Phase 1 we have $\nu\left(\mathcal{F}_I^\ell\right), \nu\left(\bar{\mathcal{F}}_I^\ell\right) \leq \frac{|T|}{|T|-\ell}$ (by Lemma 5(ii)) and thus $\nu\left(\bar{\mathcal{F}}_I\right) \leq \frac{3|T|}{|T|-\ell}$ (by Lemma 5(i)) and $\Delta(\mathcal{C}(\mathcal{F}_I)) \leq \nu\left(\bar{\mathcal{F}}_I\right) \leq \nu\left(\bar{\mathcal{F}}_I^\ell\right) + \frac{2|T|}{|T|-\ell} \leq \frac{3|T|}{|T|-\ell}$ (by Lemma 3). Consequently, $|T'| \leq \left(\nu\left(\mathcal{F}_I^\ell\right) + \frac{2|T|}{|T|-\ell}\right) \cdot H(\Delta(\mathcal{C}(\mathcal{F}_I))) \leq \frac{3|T|}{|T|-\ell} \cdot H\left(\frac{3|T|}{|T|-\ell}\right)$. From this we get $|I'| \leq |T'| \cdot \nu\left(\bar{\mathcal{F}}_I\right) \leq \left(\frac{3|T|}{|T|-\ell}\right)^2 \cdot H\left(\frac{3|T|}{|T|-\ell}\right)$.

The proof of Theorem 2 is now complete.

4 Proof of Theorem 3

We start by analyzing the performance of a natural Greedy Algorithm for covering $\nu\left(\mathcal{F}^\ell\right)$, that starts with $I = \emptyset$ and while $\nu(\mathcal{F}_I^\ell) \geq 1$ adds to I a star S for which $\nu(\mathcal{F}_{I \cup S}^\ell)$ is minimal. It is easy to see that the algorithm terminates since any star with center s in the inner part of some core of $\mathcal{F}_I^\ell$ and edge set $\{vs : v \in T \setminus \{s\}\}$ reduces the number of cores by one. The proof of the following statement is similar to the proof of the main result of [11].

Lemma 6. *Let $\mathcal{F}$ be a k-regular biset-family and let $\mathcal{S}$ be the collection of stars computed by the* Greedy Algorithm. *Then*

$$|\mathcal{S}| = O\left(\frac{|T|}{|T|-\ell} \log \min\left\{\nu\left(\mathcal{F}^\ell\right), |T| - \ell\right\}\right) .$$

Recall that given $\hat{C} \in \mathcal{C}\left(\mathcal{F}_I^\ell\right)$ we denote by $\hat{U}_C$ the union of the bisets in $\mathcal{F}_I^\ell$ that contain $\hat{C}$ and contain no other member of $\mathcal{C}\left(\mathcal{F}_I^\ell\right)$, and that by Lemma 2, the inner parts of the bisets in $\{\hat{U}_C : \hat{C} \in \mathcal{C}(\mathcal{F})\}$ are pairwise disjoint.

Definition 7 ([11]). *Let us say that $s \in V$* out-covers *$\hat{C} \in \mathcal{C}\left(\mathcal{F}^\ell\right)$ if $s \in U_{\hat{C}}^*$.*

Lemma 7. *Let $\mathcal{F}$ be an ℓ-regular biset-family and let $\nu = \nu\left(\mathcal{F}^\ell\right)$.*

(i) *There is $s \in T$ that out-covers at least $\nu\left(1 - \frac{\ell}{|T|}\right) - 1$ members of $\mathcal{C}\left(\mathcal{F}^\ell\right)$.*
(ii) *If s out-covers $\hat{C} \in \mathcal{C}\left(\mathcal{F}^\ell\right)$ then any edge from C to s covers all the bisets in $\mathcal{F}^\ell$ that contain $\hat{C}$ and contain no other member of $\mathcal{C}\left(\mathcal{F}^\ell\right)$.*
(iii) *Let s out-cover the members of $\mathcal{C} \subseteq \mathcal{C}\left(\mathcal{F}^\ell\right)$ and let S be a star with one edge from the inner part of each member of $\mathcal{C}$ to s. Then $\nu(\mathcal{F}^\ell) \leq \nu(\mathcal{F}_S^\ell) - |\mathcal{C}|/2$.*

Consequently, there exists a star S on T such that

$$\nu(\mathcal{F}_S^\ell) \leq \frac{1}{2}\left(1 + \frac{\ell}{|T|}\right) \cdot \nu + \frac{1}{2} = \alpha \cdot \nu + \beta \ . \tag{1}$$

Proof. We prove (i). Consider the hypergraph $\mathcal{H} = \left\{T \setminus \Gamma\left(\hat{U}_C\right) : \hat{C} \in \mathcal{C}\left(\mathcal{F}^\ell\right)\right\}$. Note that the number of members of $\mathcal{C}\left(\mathcal{F}^\ell\right)$ out-covered by any $v \in T$ is at least the degree of s in $\mathcal{H}$ minus 1. Thus all we need to prove is that there is a node $s \in T$ whose degree in $\mathcal{H}$ is at least $\nu\left(1 - \frac{\ell}{|T|}\right)$. For every $\hat{C} \in \mathcal{C}\left(\mathcal{F}^\ell\right)$ we have $\left|T \setminus \Gamma\left(\hat{U}_C\right)\right| \geq |T| - \ell$, by the ℓ-regularity of $\mathcal{F}$. Hence the bipartite incidence graph of $\mathcal{H}$ has at least $\nu(|T| - \ell)$ edges, and thus has a node $s \in T$ of degree at least $\nu\left(1 - \frac{\ell}{|T|}\right)$, which equals the degree of s in $\mathcal{H}$. Part (i) follows.

Part (ii) follows from the simple observation that any biset in $\mathcal{F}^\ell$, that contains $\hat{C}$ and contains no other member of $\mathcal{C}\left(\mathcal{F}^\ell\right)$, is contained in $\hat{U}_C$.

We prove (iii). It is sufficient to show that every $\hat{C} \in \mathcal{C}\left(\mathcal{F}^\ell_S\right)$ contains some $\hat{C}' \in \mathcal{C}\left(\mathcal{F}^\ell\right) \setminus \mathcal{C}$ or contains at least two members in $\mathcal{C}$. Clearly, $\hat{C}$ contains some $\hat{C}' \in \mathcal{C}\left(\mathcal{F}^\ell\right)$. We claim that if $\hat{C}' \in \mathcal{C}$ then $\hat{C}$ must contain some $\hat{C}'' \in \mathcal{C}\left(\mathcal{F}^\ell\right)$ distinct from $\hat{C}'$. This is so since $\hat{C} \in \mathcal{C}\left(\mathcal{F}^\ell_S\right)$ and since S covers all bisets in $\mathcal{F}^\ell$ that contain $\hat{C}$ and contain no other member of $\mathcal{C}\left(\mathcal{F}^\ell\right)$, by part (ii). Part (iii) follows. □

Let us use parameters $\alpha, \beta, \gamma, \delta$ and j set to

$$\alpha = \frac{1}{2}\left(1 + \frac{\ell}{|T|}\right), \quad \beta = \frac{1}{2}, \quad \gamma = 1 - \frac{\ell}{|T|} = 2(1-\alpha), \quad \delta = 1 \ .$$

Note that $\alpha < 1$ and that $\frac{\beta}{1-\alpha} = \frac{|T|}{|T|-\ell}$. Let j be the minimum integer satisfying $\alpha^j\left(\nu - \frac{\beta}{1-\alpha}\right) \leq \frac{2}{1-\alpha}$, namely,

$$j = \left\lceil \frac{\ln \frac{1}{2}(\nu(1-\alpha) - \beta)}{\ln(1/\alpha)} \right\rceil \leq \left\lceil \frac{\ln \frac{1}{2}\nu(1-\alpha)}{\ln(1/\alpha)} \right\rceil \ . \tag{2}$$

We assume that $\nu \geq \frac{2+\beta}{1-\alpha}$ to have $j \geq 0$ (otherwise Lemma 6 follows).

Lemma 8. *Let $0 \leq \alpha < 1$, $\beta \geq 0$, $\nu_0 = \nu$, and for $i \geq 1$ let*

$$\nu_{i+1} \leq \alpha\nu_i + \beta \qquad s_i = \gamma\nu_{i-1} - \delta \ .$$

Then $\nu_i \leq \alpha^i\left(\nu - \frac{\beta}{1-\alpha}\right) + \frac{\beta}{1-\alpha}$ and $\sum_{i=1}^{j} s_i \leq \frac{1-\alpha^j}{1-\alpha} \cdot \gamma\left(\nu - \frac{\beta}{1-\alpha}\right) + j\left(\frac{\gamma\beta}{1-\alpha} - \delta\right)$. Moreover, for j given by (2)

$$\nu_j \leq \frac{2+\beta}{1-\alpha} = \frac{5|T|}{|T|-\ell} \quad \text{and} \quad \sum_{i=1}^{j} s_i \leq 2\left(\nu - \frac{|T|}{|T|-\ell}\right) \ .$$

Proof. Unraveling the recursive inequality $\nu_{i+1} \leq \alpha\nu_i + \beta$ in the lemma we get:

$$\nu_i \leq \alpha^i\nu + \beta\left(1 + \alpha + \cdots + \alpha^{i-1}\right) = \alpha^i\nu + \beta\frac{1-\alpha^i}{1-\alpha} = \alpha^i\left(\nu - \frac{\beta}{1-\alpha}\right) + \frac{\beta}{1-\alpha} \ .$$

This implies $s_i \leq \gamma\left(\nu - \frac{\beta}{1-\alpha}\right)\alpha^{i-1} + \frac{\gamma\beta}{1-\alpha} - \delta$, and thus

$$\begin{aligned}\sum_{i=1}^{j} s_i &\leq \gamma\left(\nu - \frac{\beta}{1-\alpha}\right)\sum_{i=1}^{j}\alpha^{i-1} + j\left(\frac{\gamma\beta}{1-\alpha} - \delta\right)\\ &= \gamma\left(\nu - \frac{\beta}{1-\alpha}\right)\cdot\frac{1-\alpha^j}{1-\alpha} + j\left(\frac{\gamma\beta}{1-\alpha} - \delta\right)\end{aligned}$$

For j given by (2) we have $\nu_j \leq \alpha^j\left(\nu - \frac{\beta}{1-\alpha}\right) + \frac{\beta}{1-\alpha} \leq \frac{2}{1-\alpha} + \frac{\beta}{1-\alpha} = \frac{2+\beta}{1-\alpha}$, and

$$\begin{aligned}\sum_{i=1}^{j} s_i &\leq \frac{1-\alpha^j}{1-\alpha}\cdot\gamma\left(\nu - \frac{\beta}{1-\alpha}\right) + j\left(\frac{\gamma\beta}{1-\alpha} - \delta\right)\\ &\leq 2\left(\nu - \frac{\beta}{1-\alpha}\right) = 2\left(\nu - \frac{|T|}{|T|-\ell}\right)\ .\end{aligned}$$

□

We now finish the proof of Lemma 6. At each one of the first j iterations we out-cover at least $\nu\left(\mathcal{F}_I^k\right)\left(1 - \frac{\ell}{|T|}\right) - 1$ members of $\mathcal{C}\left(\mathcal{F}_I^\ell\right)$, by Lemmas 7. In each one of the consequent iterations, we can reduce $\nu\left(\mathcal{F}_I^\ell\right)$ by at least one, if we choose the center of the star in C for some $\hat{C} \in \mathcal{C}\left(\mathcal{F}_I^\ell\right)$. Thus using Lemma 8, performing the necessary computations, and substituting the values of the parameters, we obtain that the number of stars in $\mathcal{S}$ is bounded by

$$j + \nu_j \leq \left\lceil \frac{\ln \frac{1}{2}\nu(1-\alpha)}{\ln(1/\alpha)} \right\rceil + \frac{5|T|}{|T|-\ell} = O\left(\frac{|T|}{|T|-\ell}\log\min\{\nu, |T|-\ell\}\right)\ .$$

Now we discuss a variation of this algorithm that produces $\mathcal{S}$ with a small number of leaves. Here at each one of the first j iterations we out-cover *exactly* $\nu\left(1 - \frac{\ell}{|T|}\right) - 1$ cores. For that, we need to be able to compute the bisets $\hat{U}_C$, and such a procedure can be found in [14]. The number of edges in the stars at the end of this phase is at most $2\left(\nu - \frac{|T|}{|T|-\ell}\right)$ and $\nu_j \leq \frac{5|T|}{|T|-\ell}$. In the case of non-symmetric $\mathcal{F}$ and/or directed edges, we apply the same algorithm on $\bar{\mathcal{F}}^\ell$. At this point, we apply Phase 2 of the algorithm from the previous section. Since the number of cores of each one of $\mathcal{F}_I^\ell, \bar{\mathcal{F}}_I^\ell$ is now $O\left(\frac{|T|}{|T|-\ell}\right)$, the size of the transversal T' computed is bounded by $|T'| = O\left(\frac{|T|}{|T|-\ell}\cdot\log\frac{|T|}{|T|-\ell}\right)$. The number of stars is at most $|T'|$, while the number of edges in the stars is at most $|T'|\cdot\nu\left(\bar{\mathcal{F}}_I\right) = \left(\frac{|T|}{|T|-\ell}\right)^2\cdot O\left(\log\frac{|T|}{|T|-\ell}\right)$.

This concludes the proof of Theorem 3.

References

1. Auletta, V., Dinitz, Y., Nutov, Z., Parente, D.: A 2-approximation algorithm for finding an optimum 3-vertex-connected spanning subgraph. J. Algorithms 32(1), 21–30 (1999)

2. Cheriyan, J., Vetta, A.: Approximation algorithms for network design with metric costs. SIAM J. Discrete Mathematics 21(3), 612–636 (2007)
3. Chuzhoy, J., Khanna, S.: An $O(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. In: FOCS, pp. 437–441 (2009)
4. Dinitz, Y., Nutov, Z.: A 3-approximation algorithm for finding optimum 4,5-vertex-connected spanning subgraphs. J. Algorithms 32(1), 31–40 (1999)
5. Frank, A., Jordán, T.: Minimal edge-coverings of pairs of sets. J. Combinatorial Theory, Ser. B 65(1), 73–110 (1995)
6. Frank, A., Tardos, E.: An application of submodular flows. Linear Algebra and its Applications 114/115, 329–348 (1989)
7. Jordán, T.: On the optimal vertex-connectivity augmentation. J. Combinatorial Theory, Ser. B 63(1), 8–20 (1995)
8. Khuller, S., Raghavachari, B.: Improved approximation algorithms for uniform connectivity problems. J. Algorithms 21(2), 434–450 (1996)
9. Kortsarz, G., Krauthgamer, R., Lee, J.: Hardness of approximation for vertex-connectivity network design problems. SIAM J. Computing 33(3), 704–720 (2004)
10. Kortsarz, G., Nutov, Z.: Approximating node-connectivity problems via set covers. Algorithmica 37, 75–92 (2003)
11. Kortsarz, G., Nutov, Z.: Approximating k-node connected subgraphs via critical graphs. SIAM J. on Computing 35(1), 247–257 (2005)
12. Kortsarz, G., Nutov, Z.: Approximating minimum-cost connectivity problems. In: Gonzalez, T.F. (ed.) Approximation algorithms and Metaheuristics, Ch. 58. Chapman & Hall/CRC (2007)
13. Kortsarz, G., Nutov, Z.: Tight approximation algorithm for connectivity augmentation problems. J. Computer and System Sciences 74(5), 662–670 (2008)
14. Laekhanukit, B.: An Improved Approximation Algorithm for Minimum-Cost Subset k-Connectivity. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 13–24. Springer, Heidelberg (2011)
15. Lando, Y., Nutov, Z.: Inapproximability of survivable networks. Theoretical Computer Science 410(21-23), 2122–2125 (2009)
16. Lovász, L.: On the ratio of optimal integral and fractional covers. Discrete Mathematics 13, 383–390 (1975)
17. Nutov, Z.: Approximating minimum-cost edge-covers of crossing biset families. In: Manuscript. Preliminary version: An almost $O(\log k)$-approximation for k-connected subgraphs, SODA 2009, pp. 912–921 (2009)
18. Nutov, Z.: Approximating rooted connectivity augmentation problems. Algorithmica 44, 213–231 (2006)
19. Nutov, Z.: Approximating minimum cost connectivity problems via uncrossable bifamilies and spider-cover decompositions. In: FOCS, pp. 417–426 (2009)
20. Nutov, Z.: Approximating Node-Connectivity Augmentation Problems. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 286–297. Springer, Heidelberg (2009)
21. Végh, L.: Augmenting undirected node-connectivity by one. SIAM J. Discrete Mathematics 25(2), 695–718 (2011)

Learning in Stochastic Machine Scheduling

Sebastián Marbán, Cyriel Rutten, and Tjark Vredeveld

Department of Quantitative Economics, Maastricht University
P.O. Box 616, 6200 MD Maastricht, The Netherlands
{s.marban,c.rutten,t.vredeveld}@maastrichtuniversity.nl

Abstract. We consider a scheduling problem in which two classes of independent jobs have to be processed non-preemptively by a single machine. The processing times of the jobs are assumed to be exponentially distributed with parameters depending on the class of each job. The objective is to minimize the sum of expected completion times. We adopt a Bayesian framework in which both job class parameters are assumed to be unknown. However, by processing jobs from the corresponding class, the scheduler can gradually learn about the value of these parameters, thereby enhancing the decision making in the future.

For the traditional stochastic scheduling variant, in which the parameters are known, the policy that always processes a job with Shortest Expected Processing Time (SEPT) is an optimal policy. In this paper, we show that in the Bayesian framework the performance of SEPT is at most a factor 2 away from the performance of an optimal policy. Furthermore, we introduce a second policy learning-SEPT (ℓ-SEPT), which is an adaptive variant of SEPT. We show that ℓ-SEPT is no worse than SEPT and empirically outperforms SEPT. However, both policies have the same worst-case performance, that is, the bound of 2 is tight for both policies.

1 Introduction

In this paper, we consider the classical non-preemptive single machine scheduling problem to minimize the total completion time. In deterministic and traditional stochastic scheduling, this problem is well understood and can be solved to optimality by the Shortest (Expected) Processing Time (SPT or SEPT) policy: process the jobs in non-decreasing order of their (expected) processing time [19,22]. In traditional stochastic scheduling, it is assumed that the jobs' processing times are independent random variables of which the parameters, such as the expected value, are fully known. We relax this assumption by introducing parameter uncertainty. Like in [2,8,10,11,12], we adopt a Bayesian viewpoint in which we have prior distributions for the uncertain parameters. These priors represent our beliefs on the values of the parameters. Furthermore, the Bayesian framework allows us to learn about the value of the parameters by processing jobs and observing their realized processing times. However, experimenting with different jobs to learn about the value of the corresponding parameters can be costly in terms of the waiting times of the still to be processed jobs. Hence, learning

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 21–34, 2012.

should be conducted carefully in order to minimize the sum of completion times in expectation.

Problem definition. There are two classes of independent jobs that have to be processed by a single machine. Each class J_i consists of n_i jobs ($i = 1, 2$). All jobs are available for processing from the beginning and preemption of jobs is not allowed, that is, once a job has been initiated it must remain on the machine until completion. The processing time of a job in class J_i is a random variable, which is independently and exponentially distributed with parameter ϑ_i. Distinguishing from traditional stochastic scheduling, in the scheduling model under consideration the value of ϑ_i is unknown. The goal is to minimize the total completion time in expectation, $\sum_j \mathbf{E}[C_j]$.

We introduce a random variable Θ_i describing the scheduler's beliefs regarding the value of ϑ_i. In the Bayesian approach, ϑ_i can be considered as a realization of the random variable Θ_i. The initial distribution of Θ_i, that is, before any job has been processed, is called the prior. As in [11,12], we assume that the prior is a gamma distribution with parameters $\omega_i > 0$ and $\alpha_i > 1$. Depending on the confidence in his beliefs about ϑ_i, the scheduler can choose the values of ω_i and α_i such that the prior is very peaked (the scheduler is very certain about his beliefs) or relatively flat (the scheduler is not certain about his beliefs) or anywhere in between.

After a job of class J_i is processed, we observe this job's processing time x. Since the gamma distribution is a conjugate prior for the exponential distribution, the posterior distribution of Θ_i, representing the beliefs of ϑ_i after having observed processing time realization x, is a gamma distribution with parameters $\omega_i + x$ and $\alpha_i + 1$. This result is stated in a.o. Section 9.4 of [5] and is also derived from Bayes' theorem for probability density functions. In this way, the scheduler gradually learns about the unknown parameter, thereby enhancing his decision making in the future.

A solution to a stochastic scheduling problem is not merely a simple schedule, but a so-called *scheduling policy*. We follow the notion of scheduling policies as proposed by Möhring, Radermacher, and Weiss [17]. A scheduling policy makes decisions on which job to schedule at certain decision times. We require a policy to be *non-anticipatory*: at any time, it may not utilize the actual processing times of jobs that have not yet been completed. A scheduling policy may, of course, at any decision time, use the information that it has gathered up to this time. An *optimal scheduling policy* is defined as a non-anticipatory scheduling policy that minimizes the objective value in expectation. Note hereby that an optimal scheduling policy underlies the uncertainty about processing times as well as the uncertainty about the parameters.

Burnetas and Katehakis [2] and Hamada and Glazebrook [11] present optimal policies for different number of job classes. Even for the case of two job classes, one of which has known parameter, these policies require solving extensive dynamic programs. This is in contrast to the traditional stochastic scheduling variant of the problem in which the optimal scheduling policy is SEPT [19]. The reason why SEPT is not an optimal policy in the Bayesian setting lies in the fact that when the expected processing times of the job classes are close to each

other and the parameter of the class with higher expected value is more uncertain it may be beneficial to learn about the value of the underlying parameter of this class. As SEPT is a very simple policy that is optimal for the traditional stochastic scheduling problem under consideration, it is interesting to know how well it performs in the setting with parameter uncertainty.

In the Bayesian setting, there are two natural versions of SEPT. The first one, which we keep calling SEPT, determines the order in which the jobs will be processed at the beginning based on its initial beliefs. The second version, which we denote by *learning*-SEPT or ℓ-SEPT, updates its beliefs on ϑ_i every time a job of class J_i is completed. After each completion of a job, ℓ-SEPT will schedule the job with shortest expected processing time with respect to its current beliefs. In this paper, we investigate the quality of the solution value obtained by both policies. Adopting the definition of [18], we define a policy Π to be a *ρ-approximative policy* when $\mathbf{E}[\Pi(I)] \leq \rho\,\mathbf{E}[\mathrm{OPT}(I)]$ on any scheduling instance I. Here $\mathbf{E}[\Pi(I)]$ is the expected total completion time of policy Π on instance I and OPT is the optimal non-anticipatory policy. The value ρ is called the *(worst case) performance guarantee*.

Related work. In traditional stochastic scheduling, the processing times of jobs are random variables for which the parameters of the underlying distribution are known. Rothkopf [19] shows that WSEPT (Weighted Shortest Expected Processing Time) is an optimal policy for the stochastic single machine scheduling problem, where the objective is to minimize the sum of weighted expected completion times. Weiss [23,24] analyzes the performance of WSEPT for the stochastic parallel machine scheduling problem. He shows asymptotic optimality of WSEPT for a certain class of processing time distributions. The first guarantee on the quality of an approximative policy was given by Möhring, Schulz, and Uetz [18]. Other approximative policies have been considered in [4,15,16,21].

This paper contributes to the field by applying a Bayesian framework to the single machine scheduling problem. Examples of papers that apply the same framework to scheduling problems are limited. In the pioneering paper of Gittins and Glazebrook [8], the distributions of processing times of jobs depend all upon the same unknown parameter. The optimal schedule is obtained by calculating appropriate dynamic allocation indices, first proposed by Gittins and Jones [9]. Hamada and Glazebrook [11] present another example studying the Bayesian scheduling problem with multiple weighted job classes. Optimal policies are derived using dynamic allocation indices similar to the ones in [7]. Burnetas and Katehakis [2] derive optimality conditions for the same problem with two job classes: one with known and one with unknown underlying parameter. Glazebrook and Owen [10] quantify the difference between adaptive scheduling policies based on Bayesian methodology and non-adaptive classical stochastic scheduling policies.

Bayesian methodology is widely applied in research fields related to scheduling. In inventory management for example, there is a large body of literature dealing with uncertain demand distributions and Bayesian learning. Pioneered by [20], some recent papers are given by [3,13]. The majority of these papers assumes that prices are exogenous and studies the problem of making optimal

inventory decisions. Bayesian demand learning has also received a great deal of attention within the field of pricing, see [1,14]. All these papers are experimental in that they focus on developing heuristics and studying their computational aspects. The first, and so far only, paper to analyze the theoretical worst-case performance of a Bayesian pricing heuristic is [6].

Our results. In Section 3, we first show that ℓ-SEPT is in expectation better than the non-adaptive version SEPT. Furthermore, we show that the performance guarantee for both SEPT and ℓ-SEPT is a function depending on the number of jobs in both classes and that this function can be arbitrarily close to, but is bounded by, 2. If one of the two job classes has a constant number of jobs and the number of jobs of the other class tends to infinity, then SEPT and linebreak ℓ-SEPT are asymptotically optimal. In Section 4, we show that the bound for SEPT as well as the bound for ℓ-SEPT is tight. To the best of our knowledge, this is one of the first tight performance guarantees in stochastic scheduling, where the tightness follows from non-degenerate processing time distributions. Section 5 complements our theoretical findings with some preliminary computational results, showing that ℓ-SEPT in practice outperforms the non-adaptive variant, although the worst-case performance guarantees are the same. Finally, we conclude with some remarks on the case of m job classes.

2 Preliminaries and Scheduling Policies

In this section, we introduce the Bayesian scheduling framework and policies SEPT, ℓ-SEPT, and OPT. Additionally, we give useful bounds on the performance of these policies.

2.1 Bayesian Methodology

Bayesian methodology offers a method to formally recognize the uncertainty regarding parameter ϑ_i. A random variable Θ_i is introduced which describes the scheduler's beliefs regarding the value of ϑ_i. In the Bayesian approach, ϑ_i can be considered as a realization of the random variable Θ_i. For some $\theta > 0$, let $g_i(\theta) := \frac{\partial}{\partial\theta} \mathbf{Pr}\left[\Theta_i \leq \theta\right]$ denote a (prior) probability density function. Intuitively, the probability expresses how strongly we believe that the value of ϑ_i is less than or equal to θ, prior to seeing any realization of processing times of jobs of class J_i. We assume $g_i(\theta)$ to be a gamma distribution with parameters $\omega_i > 0$ and $\alpha_i > 1$. Once k jobs of class J_i have been completed with processing time realizations x_1 up to x_k, the beliefs with respect to the unknown value of ϑ_i will be updated and expressed by the (posterior) probability density function

$$g_i(\theta|x_1,\ldots,x_k) := \frac{\partial}{\partial\theta} \mathbf{Pr}\left[\Theta_i \leq \theta | X_1 = x_1,\ldots,X_k = x_k\right] .$$

Since the gamma distribution provides a conjugate prior for the exponential distribution, the posterior $g_i(\theta|x_1,\ldots,x_k)$ is also a gamma distribution with parameters $\omega_i' := \omega_i + \sum_{j=1}^{k} x_j$ and $\alpha_i' := \alpha_i + k$ (see e.g. Section 9.4 of [5]).

Updating beliefs toward ϑ_i results in updated beliefs regarding the processing times of uncompleted jobs in class J_i. The probability density function expressing these latter beliefs, after having completed k jobs of class J_i, is denoted by

$$f_{i,k+1}(x_{k+1}) := \frac{\partial}{\partial x_{k+1}} \mathbf{Pr}\left[X_{k+1} \leq x_{k+1} | X_1 = x_1, \ldots, X_k = x_k\right] ,$$

which is equal to

$$\begin{aligned} f_{i,k+1}(x_{k+1}) &= \int_0^\infty f(x_{k+1}|\theta) g_i(\theta|x_1, \ldots, x_k) \partial\theta \qquad (1) \\ &= \int_0^\infty \theta e^{-\theta x_{k+1}} \frac{{\omega_i'}^{\alpha_i'}}{\Gamma(\alpha_i')} \theta^{\alpha_i'-1} e^{-\theta\omega_i'} \partial\theta = \frac{\alpha_i' {\omega_i'}^{\alpha_i'}}{(\omega_i' + x_{k+1})^{\alpha_i'+1}}, \end{aligned}$$

where $f(x_{k+1}|\theta)$ is an exponential probability density function with parameter θ. Furthermore, straightforward integration yields the first moment of X_{k+1}:

$$\mathbf{E}\left[X_{k+1}|x_1, \ldots, x_k\right] = \int_0^\infty x_{k+1}\, f_{i,k+1}(x_{k+1}) \partial x_{k+1} = \frac{\omega_i + \sum_{j=1}^k x_j}{\alpha_i + k - 1}. \qquad (2)$$

The more jobs of job class i have been processed, the more accurate the scheduler's beliefs regarding ϑ_i will be. First, the expected value of $(\Theta_i|x_1, \ldots, x_k)$ will converge to ϑ_i by the law of large numbers. Secondly, the variance of $(\Theta_i|x_1, \ldots, x_k)$ will decrease since ω_i and α_i will be increased with every new observation. Hence, the more jobs we process, the more peaked and the more centered around ϑ_i the distribution of $(\Theta_i|x_1, \ldots, x_k)$ will become, i. e., the more we learn about the value of ϑ_i.

2.2 Bayesian Scheduling Policies

An optimal policy for the Bayesian scheduling problem at hand, OPT, minimizes total completion time in expectation, thereby taking into account the uncertainty regarding the job class parameters. That is, the values of the parameters ϑ_i are unknown to OPT, but the policy will anticipate and act in its decision making upon the additional information to be revealed when processing a job of a certain class. In order to characterize OPT, we formulate the problem as a dynamic program, introduced by [11].

Let $\boldsymbol{n} = (n_1, n_2)$, $\boldsymbol{\omega} = (\omega_1, \omega_2)$, and $\boldsymbol{\alpha} = (\alpha_1, \alpha_2)$. Then, $(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha}) = (n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha}) \in \mathbb{Z}_+^2 \times \mathbb{R}_{>0}^2 \times \mathbb{R}_{>1}^2$ denotes a state vector encompassing all relevant information of the state the system is in. It consists of the number of jobs in each class J_i as well as the parameters of the current belief for ϑ_i. Let $\boldsymbol{e}_i$ be the ith unit vector. If in state $(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})$, a job of class J_i is processed and completed having realization x, then the state changes to $(\boldsymbol{n} - \boldsymbol{e}_i, \boldsymbol{\omega} + x\boldsymbol{e}_i, \boldsymbol{\alpha} + \boldsymbol{e}_i)$. Let $\mathbf{E}\left[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]$ denote the expected sum of completion times when the optimal policy is adopted from state $(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})$ onwards. Further, let $\mathbf{E}\left[\Pi_i^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]$ denote the sum of the expected completion times of a policy which first processes a job of class J_i (assuming $n_i \geq 1$) and follows an optimal policy afterwards.

An optimal policy can then be modeled by the following dynamic program:

$$\mathbf{E}\left[\Pi^*(\boldsymbol{n},\boldsymbol{\omega},\boldsymbol{\alpha})\right] = \min\left\{\mathbf{E}\left[\Pi_1^*(\boldsymbol{n},\boldsymbol{\omega},\boldsymbol{\alpha})\right], \mathbf{E}\left[\Pi_2^*(\boldsymbol{n},\boldsymbol{\omega},\boldsymbol{\alpha})\right]\right\} \quad \forall\, \boldsymbol{n} \geq \mathbf{1} \tag{3}$$

and

$$\mathbf{E}\left[\Pi^*(n_1,0,\boldsymbol{\omega},\boldsymbol{\alpha})\right] = \left(\sum_{i=1}^{n_1} i\right)\frac{\omega_1}{\alpha_1-1} = \frac{n_1(n_1+1)}{2}\frac{\omega_1}{\alpha_1-1} \quad \forall\, n_1 \geq 0,$$

$$\mathbf{E}\left[\Pi^*(0,n_2,\boldsymbol{\omega},\boldsymbol{\alpha})\right] = \left(\sum_{i=1}^{n_2} i\right)\frac{\omega_2}{\alpha_2-1} = \frac{n_2(n_2+1)}{2}\frac{\omega_2}{\alpha_2-1} \quad \forall\, n_2 \geq 0.$$

As the length of the first job to be processed by a policy influences the completion time of all jobs, straightforward calculations show that

$$\mathbf{E}\left[\Pi_i^*(\boldsymbol{n},\boldsymbol{\omega},\boldsymbol{\alpha})\right] = (n_1+n_2)\frac{\omega_i}{\alpha_i-1} + \int_0^\infty \mathbf{E}\left[\Pi^*(\boldsymbol{n}-\boldsymbol{e}_i,\boldsymbol{\omega}+x\boldsymbol{e}_i,\boldsymbol{\alpha}+\boldsymbol{e}_i)\right] f_{i1}(x)dx, \tag{4}$$

for all $n_i \geq 1$.

In the traditional stochastic scheduling variant, in which the parameters ϑ_i are known, the policy SEPT processes jobs in non-decreasing order of expected processing times. In the Bayesian scheduling problem at hand, SEPT processes the jobs of each job class en bloc, starting with the class having the shortest expected processing time. Formally, SEPT starts processing all jobs of class J_1 in case $\frac{\omega_1}{\alpha_1-1} < \frac{\omega_1}{\alpha_1-1}$ followed by all jobs of class J_2, and vice versa otherwise. The random variable for the sum of completion times of SEPT is denoted by Π^s, and its expected value can be written as

$$\begin{aligned}\mathbf{E}\left[\Pi^s(n_1,n_2,\boldsymbol{\omega},\boldsymbol{\alpha})\right] =& \frac{n_1(n_1+1)}{2}\frac{\omega_1}{\alpha_1-1} + \frac{n_2(n_2+1)}{2}\frac{\omega_2}{\alpha_2-1} \\ &+ n_1 n_2 \min\left\{\frac{\omega_1}{\alpha_1-1}, \frac{\omega_2}{\alpha_2-1}\right\}.\end{aligned} \tag{5}$$

The non-adaptive character of SEPT could result in performance loss in comparison to a policy which makes use of additional information being revealed when processing the jobs. This shortcoming of SEPT is illustrated by the following example.

Example 1. Consider the Bayesian scheduling problem with two job classes. Let $\omega_1 = 10, \alpha_1 - 1 = 90, \omega_2 = 0.2$ and $\alpha_2 - 1 = 2$ such that $\mathbf{E}\left[X_1\right] = \frac{\omega_1}{\alpha_1-1} = \frac{10}{90} > 0.1$ and $\mathbf{E}\left[X_2\right] = \frac{\omega_2}{\alpha_2-1} = \frac{0.2}{2} = 0.1$, where X_i denotes the processing time of the first job to be processed of class J_i. Since $\mathbf{E}\left[X_1\right] > \mathbf{E}\left[X_2\right]$, SEPT will first process all jobs of class J_2 and afterward all jobs of class J_1. However, we picked our values in such a way that the distribution of Θ_1 is peaked, i. e., we are relatively sure about the value of ϑ_1, whereas the distribution of Θ_2 is flat, i. e., we are relatively unsure about the value of ϑ_2 (see Figure 1). Consequently, it might be

that actually $\vartheta_2 < \vartheta_1$, such that, in contrast to SEPT, it would be best to first start processing all jobs of class J_1. Just like SEPT, OPT will start processing the jobs of class J_2 since $\mathbf{E}[X_2] < \mathbf{E}[X_1]$ and the beliefs regarding ϑ_2 are not that strong. However, in case $\vartheta_2 < \vartheta_1$, OPT will observe high processing times for the first few jobs of job class J_2 and realize his mistake. After processing a few jobs of job class J_2, OPT will therefore switch to processing jobs of class J_1, whereas SEPT continues with processing all jobs of job class J_2. By choosing appropriate values for the parameters $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ the probability that $\vartheta_2 < \vartheta_1$ can be made even larger. Hence, the performance of SEPT can be far away from that of OPT.

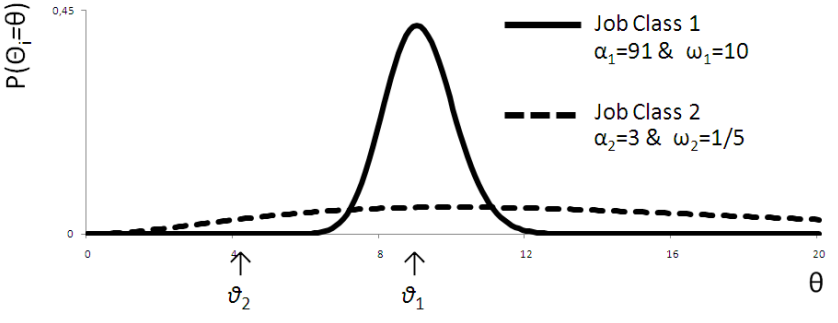


Fig. 1. Gamma distributions describing the beliefs with respect to the unknown parameters ϑ_1 and ϑ_2. Since the distribution corresponding to job class J_1 (J_2) is relatively peaked (flat), we are quite sure (unsure) about the value of ϑ_1 (ϑ_2).

To overcome the shortcoming discussed in the example above, we propose an adaptive policy *learning*-SEPT (ℓ-SEPT). Whenever the machine is idle, this policy starts processing the job with shortest expected processing time. Thereby, it updates the expected processing time of jobs in a class every time a job of this specific class has been completed. Formally, after k_1 jobs of class J_1 and k_2 jobs of class J_2 have been finished, ℓ-SEPT starts processing a job of class J_1 in case $\frac{\omega_1+\sum_{j=1}^{k_1} x_j}{\alpha_1+k_1-1} < \frac{\omega_2+\sum_{j=1}^{k_2} y_j}{\alpha_2+k_2-1}$, and a job of class J_2 otherwise, where x_i denotes the observed value of the processing time of the ith job of class J_1 and y_j denotes the realized value of the processing time of the jth job of class J_2. Note that in Example 1, ℓ-SEPT also starts processing jobs of class J_2. However, in case $\vartheta_2 < \vartheta_1$, just like OPT, ℓ-SEPT will realize his mistake after having processed a few jobs of class J_2 and continue with processing jobs of class J_1. In what follows, Π^{ℓ} denotes the random variable for the sum of completion times when policy ℓ-SEPT is used.

To summarize, we observe that ℓ-SEPT uses more information than SEPT whereas OPT uses all available information, although none of the three policies know the values of ϑ_i. All three policies know the values of ω_i and α_i which are derived from the scheduler's beliefs about ϑ_i. Based on these values SEPT

processes first all jobs of the job class with minimal expected processing time for the first job to be processed. OPT and ℓ-SEPT are more intelligent in the sense that they make use of the underlying distribution of Θ_i and update this distribution in light of new realizations. OPT in particular uses $g_i(\theta|x_1,\ldots,x_k)$ through equations (1), (3), and (4). ℓ-SEPT actually only uses the first moment of the updated distribution of $(\Theta_i|x_1,\ldots,x_k)$ to determine that the expected processing time of the next job of job class J_i equals (2), once k jobs of job class J_i have been processed.

In terms of decision making, one could thus interpret OPT as having a long-term view whereas SEPT and ℓ-SEPT both have a short-term view. Both policies process a job of class J_i only if the expected processing time of the next job in this class is minimal. OPT, however, might choose to process a job of class J_i for which the expected processing time is not necessarily minimal. As a trade-off, OPT benefits from the additional information which is acquired regarding the uncertain parameter ϑ_i. This information could then lead to better future decision making and a lower sum of completion times.

2.3 Bounds on Scheduling Policies

A trivial lower bound on the performance of an arbitrary policy is based on the fact that in any policy jobs of a class have to wait for other jobs of the same class. Hence, in constructing the lower bound we neglect waiting times caused by jobs having to wait for jobs of a different class.

Lemma 1. *Let Π be an arbitrary scheduling policy. Then, for any $n_1, n_2 \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$,*

$$\begin{aligned}\mathbf{E}\left[\Pi(n_1,n_2,\boldsymbol{\omega},\boldsymbol{\alpha})\right] &\geq \mathbf{E}\left[\Pi(n_1,0,\boldsymbol{\omega},\boldsymbol{\alpha})\right] + \mathbf{E}\left[\Pi(0,n_2,\boldsymbol{\omega},\boldsymbol{\alpha})\right] \\ &= \frac{(n_1+1)n_1}{2}\frac{\omega_1}{\alpha_1-1} + \frac{(n_2+1)n_2}{2}\frac{\omega_2}{\alpha_2-1}.\end{aligned}$$

As the expected completion time of each job is delayed by the expected processing time of the first job to be processed by the optimal policy, we can bound the value of the optimal policy as in the following lemma.

Lemma 2. *For any $n_1, n_2 \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$,*

$$\begin{aligned}\mathbf{E}\left[\Pi^*(n_1,n_2,\boldsymbol{\omega},\boldsymbol{\alpha})\right] \geq{}& \frac{n_1(n_1+1)}{2}\frac{\omega_1}{\alpha_1-1} + \frac{n_2(n_2+1)}{2}\frac{\omega_2}{\alpha_2-1} \\ &+ \min\{n_1,n_2\}\min\left\{\frac{\omega_1}{\alpha_1-1},\frac{\omega_2}{\alpha_2-1}\right\}.\end{aligned}$$

3 Upper Bound on Performance Guarantees

In this section, we prove that both SEPT and ℓ-SEPT have a performance guarantee less than 2. First, we show that the adaptive policy is indeed better than sequencing the jobs a priori. The proof of this theorem is postponed to the full version.

Theorem 1. *For any* $\boldsymbol{n} \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$, *and* $\boldsymbol{\alpha} > \mathbf{1}$,

$$\mathbf{E}\left[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})\right] \leq \mathbf{E}\left[\Pi^{s}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})\right].$$

Given the relation between SEPT and ℓ-SEPT, we can prove the performance guarantee on both SEPT and ℓ-SEPT.

Theorem 2. *For any* $n_1, n_2 \geq 0$, $\boldsymbol{\omega} > \mathbf{0}$, *and* $\boldsymbol{\alpha} > \mathbf{1}$,

$$\frac{\mathbf{E}\left[\Pi^{s}(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]}{\mathbf{E}\left[\Pi^{*}(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]} \leq \frac{n_1^2 + n_2^2 + 2n_1n_2 + n_1 + n_2}{n_1^2 + n_2^2 + n_1 + n_2 + 2\min\{n_1, n_2\}} < 2.$$

Proof. The first inequality follows directly from Theorem 1. To prove the second and last inequality, let $n_1, n_2 \geq 0$, $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$. Combining (5) and Lemma 2, we obtain

$$\frac{\mathbf{E}\left[\Pi^{s}(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]}{\mathbf{E}\left[\Pi^{*}(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]} \leq \frac{n_1(n_1+1)\frac{\omega_1}{\alpha_1-1} + n_2(n_2+1)\frac{\omega_2}{\alpha_2-1} + 2n_1n_2\min\left\{\frac{\omega_1}{\alpha_1-1}, \frac{\omega_2}{\alpha_2-1}\right\}}{n_1(n_1+1)\frac{\omega_1}{\alpha_1-1} + n_2(n_2+1)\frac{\omega_2}{\alpha_2-1} + 2\min\{n_1, n_2\}\min\left\{\frac{\omega_1}{\alpha_1-1}, \frac{\omega_2}{\alpha_2-1}\right\}}$$

observing that for any $0 < c \leq b$ and $0 < d \leq a$, it holds that $\frac{a+b}{a+c} \leq \frac{d+b}{d+c}$ and replacing $\frac{\omega_1}{\alpha_1-1}$ and $\frac{\omega_2}{\alpha_2-1}$ by the minimum of the two, we can bound this by

$$\leq \frac{n_1^2 + n_2^2 + 2n_1n_2 + n_1 + n_2}{n_1^2 + n_2^2 + n_1 + n_2 + 2\min\{n_1, n_2\}} \leq \frac{4n_{\max}^2 + 2n_{\max}}{2n_{\max}^2 + 2n_{\max}} < 2,$$

where $n_{\max} = \max\{n_1, n_2\}$.

Note that it follows from Theorem 2 that the performance guarantee will be close to one in case the number of jobs in one class is of a different order than the number of jobs in the second class. To be more explicit, when the number of jobs in one class is fixed while the number of jobs in the second class tends to infinity, then the performance guarantee will go to one, yielding asymptotic optimality of SEPT and ℓ-SEPT.

4 Tightness of the Performance Guarantees

In this section, we show that the performance guarantee shown in the previous section is tight for SEPT as well as ℓ-SEPT. Although by Theorem 1, it suffices to show that the guarantee of ℓ-SEPT is tight, we first give a lower bound on the performance guarantee of SEPT, as this one is more intuitive, whereas the lower bound for ℓ-SEPT is rather technical.

4.1 Lower Bound on the Performance Guarantee of SEPT

We show that for any $\epsilon > 0$ there exists an instance for which the ratio of the value of SEPT to the value of OPT is only an additive ϵ away from the performance guarantee of Theorem 2. In order to obtain this result, we make use of the following two facts.

Fact 1. *For any $\omega > 0$ and $\alpha > 1$,*

$$\int_0^\infty \min\left\{\frac{\omega + x}{\alpha}, 1\right\} f_{11}(x)dx = \frac{\omega}{\alpha - 1} - \frac{1}{\alpha - 1}\left(\frac{\omega}{\alpha}\right)^\alpha.$$

Fact 2. *For any $\alpha > 1$,*

$$\lim_{\alpha \downarrow 1} \frac{1}{\alpha - 1}\left(\frac{\alpha - 1}{\alpha}\right)^\alpha = 1$$

Additionally, we need a lower bound on SEPT and an upper bound on OPT.

Lemma 3. *For any $n_1, n_2 \geq 0$, there exist parameter settings $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$ such that $\frac{\omega_1}{\alpha_1 - 1} < \frac{\omega_2}{\alpha_2 - 1} = 1$ and*

$$\mathbf{E}\left[\Pi^s(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right] > \frac{n_1(n_1+1)}{2} + \frac{n_2(n_2+1)}{2} + n_1 n_2 - \epsilon,$$

for any $\epsilon > 0$.

Proof. For all $\epsilon' > 0$ and arbitrary $\alpha_1 > 1$, let $\omega_1 = (1 - \epsilon')(\alpha_1 - 1)$. By (5), we have

$$\mathbf{E}\left[\Pi^s(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right] = \frac{n_1(n_1+1)}{2}(1 - \epsilon') + \frac{n_2(n_2+1)}{2} + n_1 n_2(1 - \epsilon').$$

Hence, for any $\epsilon > 0$, there exists an $\epsilon' > 0$ for which the lemma holds.

Lemma 4. *For any $n_1, n_2 \geq 0$, there exist parameter settings $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$ such that $\frac{\omega_1}{\alpha_1 - 1} < \frac{\omega_2}{\alpha_2 - 1} = 1$ and*

$$\mathbf{E}\left[\Pi^*(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right] < n_1 + \frac{n_1(n_1+1)}{2} + \frac{n_2(n_2+1)}{2} + \epsilon,$$

for any $\epsilon > 0$.

Proof. Consider the following policy Π: first process one job of class J_2, observing realization y, and schedule all remaining jobs according to SEPT. That is, if $\frac{\omega_1}{\alpha_1 - 1} \leq \frac{\omega_2 + y}{\alpha_2}$ then process first all jobs of class J_1 and then the remaining jobs of class J_2 and otherwise first process the remaining jobs of class J_2 and then all jobs of class J_1. Using y to denote the observed value of the first job of class J_2, we have that for any $n_1, n_2 \geq 0$, $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$ such that $\frac{\omega_1}{\alpha_1 - 1} < \frac{\omega_2}{\alpha_2 - 1} = 1$,

$$\begin{aligned}
&\mathbf{E}\left[\Pi^*(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right] \leq \mathbf{E}\left[\Pi(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right] \\
&= (n_1 + n_2)\frac{\omega_2}{\alpha_2 - 1} + \int_0^\infty \mathbf{E}\left[\Pi^s(n_1, n_2 - 1, \boldsymbol{\omega} + y\boldsymbol{e}_2, \boldsymbol{\alpha} + \boldsymbol{e}_2)\right] f_{21}(y)dy \\
&\overset{(5)}{=} n_1 + n_2 + \frac{n_1(n_1+1)}{2}\frac{\omega_1}{\alpha_1 - 1} + \frac{n_2(n_2-1)}{2} + n_1(n_2 - 1)\int_0^\infty \min\left\{\frac{\omega_1}{\alpha_1 - 1}, \frac{\omega_2 + y}{\alpha_2}\right\} f_{21}(y)dy \\
&< n_1 + \frac{n_1(n_1+1)}{2} + \frac{n_2(n_2+1)}{2} + n_1(n_2 - 1)\int_0^\infty \min\left\{1, \frac{\omega_2 + y_1}{\alpha_2}\right\} f_{21}(y)dy \\
&\overset{\text{Fact 1}}{=} n_1 + \frac{n_1(n_1+1)}{2} + \frac{n_2(n_2+1)}{2} + n_1(n_2 - 1)\left[\frac{\omega_2}{\alpha_2 - 1} - \frac{1}{\alpha_2 - 1}\left(\frac{\omega_2}{\alpha_2}\right)^{\alpha_2}\right]. \qquad (6)
\end{aligned}$$

Recall that by assumption $\omega_2 = \alpha_2 - 1$. Combining (6) and Fact 2, and letting α_2 tend to 1 from above, we find

$$\lim_{\alpha_2 \downarrow 1} \mathbf{E}\left[\Pi^*(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right] < n_1 + \frac{n_1(n_1+1)}{2} + \frac{(n_2+1)n_2}{2}.$$

Hence, it follows that for any n_1, n_2, $\omega_1 < \alpha_1 - 1$, there exists for any $\epsilon > 0$ an $\alpha^* > 1$ such that for all $1 < \alpha_2 = \omega_2 + 1 < \alpha^*$

$$\mathbf{E}\left[\Pi^*(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right] < n_1 + \frac{n_1(n_1+1)}{2} + \frac{n_2(n_2+1)}{2} + \epsilon.$$

As a straightforward consequence of Lemmata 3 and 4, we obtain the following theorem.

Theorem 3. *For any n_1 and n_2, there exist parameter settings $\boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$, such that, for any $\epsilon > 0$*

$$\frac{\mathbf{E}\left[\Pi^s(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]}{\mathbf{E}\left[\Pi^*(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]} > \frac{n_1^2 + n_2^2 + 2n_1n_2 + n_1 + n_2}{n_1^2 + n_2^2 + 3n_1 + n_2} - \epsilon.$$

Furthermore, there exist parameter settings $n_1, n_2 \geq 0$, $\boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$, such that for any $\epsilon > 0$,

$$\frac{\mathbf{E}\left[\Pi^s(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]}{\mathbf{E}\left[\Pi^*(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]} > 2 - \epsilon.$$

Proof. The restrictions imposed on the values α_1 and α_2 in Lemmas 3 and 4 can be satisfied simultaneously. Therefore, the first part of the theorem follows directly from these lemmas. To see the second part, we set $n_1 = n_2 = n$ and let n tend to infinity.

$$\lim_{n\to\infty} \frac{\mathbf{E}\left[\Pi^s(n, n, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]}{\mathbf{E}\left[\Pi^*(n, n, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]} > \lim_{n\to\infty} \frac{2n^2 + n}{n^2 + 2n} - \epsilon = \lim_{n\to\infty} \frac{2n+1}{n+2} - \epsilon = 2 - \epsilon.$$

4.2 Lower Bound on the Performance Guarantee of ℓ-SEPT

Similarly to the previous section, we show that for any $\epsilon > 0$ there exists an instance for which the ratio of the value of ℓ-SEPT to the value of OPT is only an additive ϵ away from the performance guarantee of Theorem 2.

Theorem 4. *There exist parameter settings $n_1, n_2 \geq 0$, $\boldsymbol{\omega} > \mathbf{0}$, $\boldsymbol{\alpha} > \mathbf{1}$ such that*

$$\frac{\mathbf{E}\left[\Pi^\ell(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]}{\mathbf{E}\left[\Pi^*(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})\right]} > \frac{n_1^2 + n_2^2 + 2n_1n_2 + n_1 + n_2}{n_1^2 + n_2^2 + 3n_1 + n_2} - \epsilon$$

for any $\epsilon > 0$.

A formal proof of this theorem is given in the full version of the paper. In order to give this proof, we need a lower bound on the performance of ℓ-SEPT. To obtain this bound, we adjust the worst case instance of SEPT, given in Lemma 3. In that instance, we set our parameters in such a way that $\mathbf{E}[X_1]$ is slightly less than $\mathbf{E}[X_2]$. Hence, SEPT starts processing all jobs of class J_1, followed by the jobs of class J_2. OPT however, starts processing a job from class J_2, since the distribution of Θ_2 is flat, making it is beneficial to process a few jobs of the second class to get a better idea about the value of ϑ_2.

To create a bad instance for ℓ-SEPT, we would like to keep the same structure. Therefore, we need to make sure ℓ-SEPT does not switch to processing jobs from the second class after it processed a few jobs of the first class. This is done by setting the values of ω_1 and α_1 extremely large such that we are almost certain about the value of ϑ_1. Consequently, the realizations of processing times of jobs from class J_1 barely affect the expected processing time for the next job to be processed, i. e., when ω_1 and α_1 are big enough we have

$$\frac{\omega_1 + \sum_{j=1}^{k} x_k}{\alpha_1 + k - 1} \approx \frac{\omega_1}{\alpha_1 - 1} = 1 - \epsilon < 1 = \frac{\omega_2}{\alpha_2 - 1}$$

after k observations on the first job class.

5 Computational Results

In this section, we present preliminary computational results to investigate the performance of SEPT and ℓ-SEPT with respect to the optimal value in a Bayesian setting. That is, for several job class settings, we compare the values of SEPT and ℓ-SEPT with the optimal Bayesian solution. All computations are performed in MATLAB. In order to compute the values of OPT, we used the algorithm presented in Section 4 of the paper of Hamada and Glazebrook [11].

The Bayesian scheduling instances studied are as follows: the number of jobs in both job classes is set to 15, since the theoretical worst-case performance is reached for equal number of jobs in both classes. Furthermore, the gamma prior settings are set such that ω_i and $(\alpha_i - 1)$ are both an element of $\{0.5\,;\,1.0\,;\,5.0\,;\,25.0\}$ for each job class J_i. This results in 100 different computations covering the majority of interesting job class settings, i.e., the cases in which both job classes have high or low parameter uncertainty, and the mixed case in which one class has high and the other one low parameter uncertainty. Moreover, these computations could still be performed in a reasonable amount of time. Choosing our settings in a more extreme fashion immediately results in difficulties with the precision in calculating OPT, and also significantly increases the computation time of this optimal policy.

In our computations, 50.000 simulations are run for each Bayesian scheduling instance. In each of those simulations, we draw for each job class a parameter realization from a gamma distribution. This realization is subsequently used to draw 15 processing time realizations from an exponential distribution. Using

these realizations the sum of completion times for each of the policies is calculated. Performance of the policies SEPT and ℓ-SEPT is measured by average objective value of the policy over the average objective value of OPT.

The preliminary computational results indicate that in case both job classes have high parameter uncertainty ℓ-SEPT is only about 1% away from the optimal value, while for SEPT the deviation is more than 13%. On the other hand, when the parameter uncertainty is low, we find that SEPT performs already better (1% away from OPT), but ℓ-SEPT obtains exactly the same value as OPT. In case both job classes have the same expected processing time, SEPT has the worst performance ratio among the instances tested: for high parameter uncertainty SEPT is about 30% above OPT, and for medium parameter uncertainty it is still 7% away from the optimal value. Intuitively, this was to be expected, because in these cases SEPT will just randomly choose a job class to start with. Also ℓ-SEPT performs the worst when both job classes have the same expected processing time, and in addition one job class has high parameter uncertainty, whereas the other one has low parameter uncertainty. This is explained by the fact that ℓ-SEPT makes its decisions based only on the first moment of the distribution and disregards further moments. Still in these cases, ℓ-SEPT outperforms SEPT, and it has a maximum deviation from OPT of only 9%. To conclude, on all instances ℓ-SEPT clearly outperforms the non-adaptive variant SEPT, thereby emphasizing the impact of learning on the performance of the algorithm. Finally, we remark that when averaging over the 50.000 trials, SEPT has a much higher variance than the other two policies. Again this is explained by the fact that SEPT, in case that the two job classes have the same expected processing time, randomly picks a job class to start with.

6 Concluding Remarks

In this paper, we studied the performance guarantee of two natural extensions of the traditional stochastic scheduling policy SEPT to the setting of Bayesian scheduling. We only considered the case in which there are 2 job classes and gave tight performance guarantees for both policies. An interesting extension will be the case of m job classes. For this case, we can prove a performance guarantee of m on both SEPT and ℓ-SEPT. For the non-adaptive policy SEPT this bound is tight, whereas for the adaptive policy ℓ-SEPT, we have a lower bound of $1 + \sqrt{m-1}$ and we conjecture that this is the right performance guarantee.

Acknowledgments. We thank three anonymous reviewers for their helpful comments to improve the exposition of the paper.

References

1. Araman, V.F., Caldentey, R.: Dynamic pricing for nonperishable products with demand learning. Operations Research 57(5), 1169–1188 (2009)
2. Burnetas, A.N., Katehakis, M.N.: On sequencing two types of tasks on a single processor under incomplete information. Probability in the Engineering and Informational Sciences 7(1), 85–119 (1993)

3. Chen, L., Plambeck, E.L.: Dynamic inventory management with learning about the demand distribution and substitution probability. Manufacturing & Service Operations Management 10(2), 236–256 (2008)
4. Dean, B.C.: Approximation Algorithms for Stochastic Scheduling Problems. PhD thesis, Massachusetts Institute of Technology (2005)
5. DeGroot, M.H.: Optimal Statistical Decisions. McGraw-Hill, New York (1970)
6. Farias, F.F., Van Roy, B.: Dynamic pricing with a prior on market response. Operations Research 58(1), 16–29 (2010)
7. Gittins, J.C.: Multi-armed bandit allocation indices. Wiley, N.Y. (1989)
8. Gittins, J.C., Glazebrook, K.D.: On Bayesian models in stochastic scheduling. Journal of Applied Probability 14(3), 556–565 (1977)
9. Gittins, J.C., Jones, D.M.: A dynamic allocation index for the sequential design of experiments. In: Progress in Statistics, pp. 241–266 (1974)
10. Glazebrook, K.D., Owen, R.W.: On the value of adaptive solutions to stochastic scheduling problems. Mathematics of Operations Research 20(1), 65–89 (1995)
11. Hamada, T., Glazebrook, K.D.: A Bayesian sequential single machine scheduling problem to minimize the expected weighted sum of flowtimes of jobs with exponential processing times. Operations Research 41(5), 924–934 (1993)
12. Hamada, T., Tamaki, M.: Some results on a Bayesian sequential scheduling on two identical parallel processors. Journal of the Operations Research Society of Japan 42(14), 316–329 (1999)
13. Lariviere, M.A., Porteus, E.L.: Stalking information: Bayesian inventory management with unobserved lost sales. Management Science 45(3), 346–363 (1999)
14. Lin, K.Y.: Dynamic pricing with real-time demand learning. Operations Research 174(1), 522–538 (2003)
15. Megow, N., Uetz, M., Vredeveld, T.: Models and algorithms for stochastic online scheduling. Mathematics of Operations Research 31(3), 513–525 (2006)
16. Megow, N., Vredeveld, T.: Approximation in Preemptive Stochastic Online Scheduling. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 516–527. Springer, Heidelberg (2006)
17. Möhring, R.H., Radermacher, F.J., Weiss, G.: Stochastic scheduling problems I: General strategies. ZOR – Zeitschrift für Operations Research 28, 193–260 (1984)
18. Möhring, R.H., Schulz, A.S., Uetz, M.: Approximation in stochastic scheduling:the power of LP-based priority policies. Journal of ACM 46(6), 924–942 (1999)
19. Rothkopf, M.H.: Scheduling with random service times. Management Science 12(9), 703–713 (1966)
20. Scarf, H.: Bayes solutions of the statistical inventory problem. The Annals of Mathematical Statistics 30(2), 490–508 (1959)
21. Schulz, A.S.: New old algorithms for stochastic scheduling. In: Algorithms for Optimization with Incomplete Information (2005); Dagstuhl Seminar Proceedings, vol. 05031
22. Smith, W.E.: Various optimizers for single stage production. Naval Research Logistics Quaterly 3, 59–66 (1956)
23. Weiss, G.: Approximation results in parallel machines stochastic scheduling. Annals of Operations Research 26(1), 195–242 (1990)
24. Weiss, G.: Turnpike optimality of Smith's rule in parallel machines stochastic scheduling. Mathematics of Operations Research 17(2), 255–270 (1992)

An Online Algorithm Optimally Self-tuning to Congestion for Power Management Problems

Wolfgang Bein[1], Naoki Hatta[2], Nelson Hernandez-Cons[3], Hiro Ito[2], Shoji Kasahara[3], and Jun Kawahara[4]

[1] Center for Information Technology and Algorithms, School of Computer Science, University of Nevada, Las Vegas
beinw@unlv.nevada.edu
[2] Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University
{nhatta,itohiro}@kuis.kyoto-u.ac.jp
[3] Department of Systems Science, Graduate School of Informatics, Kyoto University
{shoji,nelson}@i.kyoto-u.ac.jp
[4] JST ERATO MINATO Project
jkawahara@erato.ist.hokudai.ac.jp

Abstract. We consider the classical power management problem: There is a device which has two states ON and OFF and one has to develop a control algorithm for changing between these states as to minimize (energy) cost when given a sequence of service requests. Although an optimal 2-competitive algorithm exists, that algorithm does not have good performance in many practical situations, especially in case the device is not used frequently. To take the frequency of device usage into account, we construct an algorithm based on the concept of "slackness degree." Then by relaxing the worst case competitive ratio of our online algorithm to $2 + \varepsilon$, where ε is an arbitrary small constant, we make the algorithm flexible to slackness. The algorithm thus automatically tunes itself to slackness degree and gives better performance than the optimal 2-competitive algorithm for real world inputs. In addition to worst case competitive ratio analysis, a queueing model analysis is given and computer simulations are reported, confirming that the performance of the algorithm is high.

1 Introduction

Consider an electric light which is turned on automatically when someone passes by, say, for example at the entrance of a building. Or consider a device which can enter a "sleep mode" – a state for energy saving when not used for a certain period of time (e.g. a server or a copy machine). We abstract such a situation to the automatic operation of a two-state device, which has an *ON-state* and an *OFF-state*. In this paper we equate the sleep mode with the OFF-state, which consumes no power. If a user utilizes the device, the state of it must be ON during usage. When the user has finished, if another user needs it almost immediately, it is wasteful to turn it off, because additional power consumption

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 35–48, 2012.

occurs when switching the state. For example, a copy machine consumes extra electrical power when it comes back from the sleep mode. In the case of a compact fluorescent light, switching states frequently shortens the life of the bulb and although the switching cost may be negligible one can amortize the shorter lifespan appropriately.

It is, of course, not known in advance when and how many users will request service. Since we have to control switching the states of the device without knowledge of future requests, the situation can be formulated as an online optimization problem. More formally, there is a two-state device. Users request the device one after another and use it for an arbitrary period of time. When it is used, the device must be kept in the ON-state. Once it is not used it can be turned off at an arbitrary time. In the ON-state the device uses an amount of power proportional to usage time, one unit of cost per unit of time. We call this the *running cost*. There is no cost in the OFF-state. There is also a constant *switching cost* $a > 0$ for turning the device on and no cost for turning it off. An optimal online algorithm is well known for this problem in the context of the worst case competitive analysis [7,11,12]. We call this the optimal worst case competitive ratio algorithm, or *OWCR*, for short. *OWCR* turns it on when a user requests service. After use *OWCR* waits for the next service request for time a. If another user requests service within the period, *OWCR* does not turn it off; otherwise, *OWCR* turns it off after time a. The *OWCR* is 2-competitive, which is optimal.

However, *OWCR* does not perform well in various natural situations. If the arrival interval of users is spaced long enough, *OWCR* keeps the device in the ON-state for an extra a units for each user. This action seems to be quite wasteful. If the device usage time is infinitesimal the cost for *OWCR* is $2a$ (the switching cost and running cost are both a). On the other hand, the behavior of the optimal offline algorithm "OPT" is to turn off immediately after the user has finished. OPT pays only switching cost a (and the infinitesimal running cost). Therefore, the competitive ratio for such a request sequence is 2. The fact is that *although OWCR does not seem clever at all, under worst case competitive analysis this algorithm has the best performance.*

Basic Concept of Our Algorithm. Since *OWCR* is optimal it appears as if there is not much hope for improvement. However, if we allow the worst case competitive ratio to increase by only a small positive constant ε from 2 (say, for example, by 0.01), we can design various algorithms that have lower cost than *OWCR* for real world inputs. For example, we can decrease the duration of keeping the ON-state ("*standby time*") gradually when the frequency of the device usage becomes low: If the *waiting time* – the time from the preceding user leaving to the latest user arriving – was more than a (something like an "off-peak" situation), the system may be slack and the algorithm may elect to make the standby time shorter. On the other hand, if the preceding waiting time was less than or equal to a (similar to a "rush hour"), the algorithm may reset the standby time.

In other words, standby time may decrease as a sequence $x_1 > x_2 > \cdots$ while off-peak arrivals continue. Such an algorithm may have better performance.

However we cannot evaluate this easily in the context standard competitive analysis. On the other hand, we are not satisfied only with heuristics or experimental analysis and we seek a rigorous theoretical analysis.

For the above aim, we introduce a parameter called "slackness degree," which represents the frequency of arrivals. We set the parameters $x_1, x_2, \ldots$ to optimize the competitive ratio for each slackness degree. The proposed algorithm adaptively reacts to fluctuation of inputs and works optimally in the sense of the competitive ratio according to slackness degrees. Moreover, an important property of our algorithm is that it need not know the actual slackness degree. Its worst case competitive ratio is at most $2 + \varepsilon$ and it is close to the optimal offline algorithm for real world inputs. E.g., if we set $\varepsilon = 0.01$ and if the slackness degree of an input is 10, 20, 50 and 100, then the competitive ratio is 1.58, 1.29, 1.12 and 1.06, respectively.

Related Work. As mentioned above it is well known that the optimal competitive ratio of this problem is 2 [9,11,12,17]. In the randomized model where the algorithm uses a probability distribution the best competitive ratio improves to $e/(e-1) \approx 1.582$ (the expected cost of the algorithm for all inputs is within a factor of $e/(e-1)$ of the optimal cost). Furthermore, it is known that this bound is tight [9,10,11]. Systems with multiple power saving states have also been studied, and it is clear that the additive model, where costs at any states are cumulative, is reduced to the two state model. Thus, it is sufficient to consider the two state model [1,8,15,18].

Strategies for this problem are categorized into two groups: adaptive and non-adaptive. Non-adaptive strategies set a threshold only once at first on the idle time interval for switching from the active state to the sleep state [8]. Adaptive strategies use the history of idle periods to make the decisions for future inputs [6,8].

There are several lines of investigation for evaluating algorithms more adequately by considering alternatives to the competitive ratio or analysis. In substitution for the competitive ratio, the accommodation ratio [4], the Max/Max ratio [2] and the random order ratio [13] were proposed. To analyze more realistic situations, competitive analysis with restriction to the adversary or using parameters have also been considered. In the access graph model [3] and the diffuse adversary model [14], the competitive ratio is evaluated by using weak adversaries whose action is restricted (see the survey [5]). Panagiotou et. al. [16] analyzed the LRU algorithm for paging by introducing parameters α and β which characterize the degree of the locality of reference. They showed the competitive ratio is bounded by a function of α and β.

Organization. Section 2 gives the basic statement of the problem. Section 3 presents our algorithm and gives the concept of slackness degree and competitive ratio analysis under this concept. Section 4 gives an analysis using queueing theory with computer simulations. Conclusions are given in Section 5.

2 Problem Statement

In this paper we consider a device with infinite capacity that has two states, an *ON-state* and an *OFF-state* (simply ON and OFF), for which we design a control algorithm for changing between ON and OFF. Let $t_1^s, \ldots, t_n^s, t_1^e, \ldots, t_n^e$ be non-negative real values that represent the service times $t\_^s$ and end-of-service times $t\_^e$ for n requests and which satisfy $0 \leq t_1^s < t_1^e < t_2^s < t_2^e < \cdots < t_n^s < t_n^e$. The input for this problem is given as $\sigma = \langle (t_1^s, t_1^e), (t_2^s, t_2^e), \ldots, (t_n^s, t_n^e) \rangle$. For this input, the device must be kept ON between times t_i^s and t_i^e for each $i = 1, \ldots, n$.

The state of the device can be switched at an arbitrary time. There is no cost for switching from ON to OFF, while there is a *switching cost* $a\,(> 0)$ when switching from OFF to ON. For keeping the ON-state, it takes *running cost* of one unit per unit time.

The strategy of the optimal offline algorithm (OPT) for this problem is clear: If the period between the current request and next one is less than a, the device is kept ON. Otherwise, it is turned off immediately. Therefore OPT's total cost for the input $\sigma = \langle (t_1^s, t_1^e), (t_2^s, t_2^e), \ldots, (t_n^s, t_n^e) \rangle$ is $a + \sum_{i=1}^{n} (t_i^e - t_i^s) + \sum_{i=1}^{n-1} \min\{t_{i+1}^s - t_i^e, a\}$.

For some i-th request (t_i^s, t_i^e), we consider the sum of the i-th running cost and the next $(i+1)$-th switching cost. Let ALG be any algorithm. Let u be the period between the end of the request and the start of the next request (i.e., $u = t_{i+1}^s - t_i^e$). Then the optimal offline algorithm pays $t_i^e - t_i^s + \min\{u, a\}$. If ALG turns the device off after $v\,(< u)$ standby time, the cost is $t_i^e - t_i^s + v + a$. Otherwise, it must pay $t_i^e - t_i^s + u$. In each case, the smaller $t_i^e - t_i^s$ is, the worse the competitive ratio becomes. Therefore, from the standpoint of competitive analysis, it is enough to consider that usage times of the device (i.e., $t_i^e - t_i^s$ for each i) are tiny. On the basis of the above discussion, we redefine this problem as follows.

Let $t_1, \ldots, t_n$ be non-negative real values satisfying $0 \leq t_1 < \cdots < t_n$ representing the time of service of the device for n requests. An input is given as $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$. We do nothing if the state is ON at t_i $(i = 1, \ldots, n)$, and should turn a device ON if it is OFF at that time. For a given input $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$ (n may be ∞), the action of an algorithm is determined by a sequence $\langle w_1, w_2, \ldots, w_n \rangle$, where w_i is standby time after ith request is leaving. In other words, the problem is how to determine w_i from $\langle t_1, t_2, \ldots, t_i \rangle$. For each $i = 2, \ldots, n$, let $u_i = t_i - t_{i-1}$ be an *idle period*. OPT's cost for this redefined problem is $\mathrm{OPT}(\sigma) = a + \sum_{i=1}^{n-1} \min\{t_{i+1} - t_i, a\}$. We denote ALG's cost for input σ by $\mathrm{ALG}(\sigma)$ and the competitive ratio of ALG for σ by $\mathcal{R}_{\mathrm{ALG}}(\sigma) = \mathrm{ALG}(\sigma)/\mathrm{OPT}(\sigma)$. Let Σ be the set of whole inputs σ. For a subset $\Sigma' \subseteq \Sigma$, we define $\mathcal{R}_{\mathrm{ALG}}(\Sigma') = \sup_{\sigma \in \Sigma'} \mathcal{R}_{\mathrm{ALG}}(\sigma)$. And we represent $\mathcal{R}_{\mathrm{ALG}} = \mathcal{R}_{\mathrm{ALG}}(\Sigma)$, which is the (worst case) competitive ratio of ALG.

3 Our Algorithm

3.1 Decrease and Reset Algorithm (DRA)

We propose an algorithm which decreases the standby time gradually when the frequency of the device usage becomes low.

Decrease and Reset Algorithm (DRA).
Let $x_1, x_2, \ldots,$ be an infinite non-increasing sequence of non-negative values. In DRA, $w_i = x_{f(i)}$ such that

$$f(i) = \begin{cases} f(i-1)+1 & \text{if } u_i \geq a \text{ and } i \neq 1, \\ 1 & \text{otherwise.} \end{cases}$$

If $x_i = a$ for all i, DRA is equivalent to *OWCR*. Setting x_i be larger than a is clearly wasteful, and hence we consider cases such that $x_i \leq a$ for all $i = 1, 2, \ldots$. From a simple observation we see that x_1 gives a lower bound of $\mathcal{R}_{\mathrm{DRA}}$:

Observation 1. $\mathcal{R}_{DRA} \geq 1 + a/x_1$.

Proof. Let m be an integer. For an input $\sigma = \langle x_1, 2x_1, \ldots, mx_1 \rangle$, OPT's total cost is $a + (m-1)x_1$ and DRA's total cost is $m(a + x_1)$. Thus the competitive ratio of them is the following:

$$\mathcal{R}_{\mathrm{DRA}} \geq \mathcal{R}_{\mathrm{DRA}}(\sigma) = \frac{m(a+x_1)}{a+(m-1)x_1} \overset{(m\to\infty)}{\to} 1 + \frac{a}{x_1}.$$

□

From this observation, it follows that x_1 cannot be much smaller than a, otherwise $\mathcal{R}_{\mathrm{DRA}}$ becomes very large. In other words, if the difference between a and x_1 is small, the effect to $\mathcal{R}_{\mathrm{DRA}}$ is not so large. Thus we relax the worst case competitive ratio from 2 to $2 + \varepsilon$ for small $\varepsilon > 0$, i.e., we let $x_1 = a/(1+\varepsilon)$.

The above observation is easily extended to the other values $x_2, x_3, \ldots,$ as follows.

Observation 2. *For any integer k,*

$$\mathcal{R}_{DRA} \geq \frac{ka + \sum_{i=1}^{k} x_i}{(k-1)a + x_k}.$$

Proof. Let m be an integer and $t_1, t_2, \ldots, t_{mk}$ be a sequence such that $t_1 = a$ and if $i = 1 \bmod k$ then $t_i = t_{i-1} + x_k$, otherwise $t_i = t_{i-1} + a$. For input $\sigma = \langle t_1, t_2, \ldots, t_{mk} \rangle$, DRA sets $w_i = x_{g(i)}$ for all i, where $g(i) = ((i-1) \bmod k) + 1$. OPT's total cost is $a + m(k-1)a + (m-1)x_k$ and DRA's total cost is $mka + m\sum_{i=1}^{k} x_i$. Thus the competitive ratio of them is the following:

$$\mathcal{R}_{\mathrm{DRA}} \geq \mathcal{R}_{\mathrm{DRA}}(\sigma) = \frac{mka + m\sum_{i=1}^{k} x_i}{a + m(k-1)a + (m-1)x_k} \overset{(m\to\infty)}{\to} \frac{ka + \sum_{i=1}^{k} x_i}{(k-1)a + x_k}.$$

So this observation is satisfied. □

Our upper bound of the competitive ratio is $2+\varepsilon$, and thus, the following inequalities must be satisfied for every $k = 1, 2, \ldots$:

$$\frac{ka + \sum_{i=1}^{k} x_i}{(k-1)a + x_k} \leq 2 + \varepsilon.$$

Solving this equation for x_k, we have

$$x_k \geq \frac{1}{1+\varepsilon}\left((2+\varepsilon)a + \sum_{i=1}^{k-1} x_i\right) - ka.$$

By elementary induction, we obtain

$$x_k \geq -\varepsilon\left(\frac{2+\varepsilon}{1+\varepsilon}\right)^k a + (1+\varepsilon)a. \tag{1}$$

This is a necessary condition for keeping the competitive ratio less than or equal to $2+\varepsilon$. But this condition is not sufficient to guarantee optimality within the ε bound. We propose next an algorithm that sets exact values for $x_2, x_3, \ldots$, to guarantee optimality within the ε bound.

3.2 How to Set the Coefficients for "Optimality"

Before turning to this problem, we need to define what "optimal" means here. Our motivation is to give a better algorithm for slack systems. Thus we introduce a measure, "slackness degree" for representing the slackness of input sequences. For an input sequence $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$, request i is called a *busy request* if $u_i \leq a$ or a *slack request*, otherwise. The first request is neither busy nor slack one. We denote the number of slack requests in σ by $s(\sigma)$, and that of busy requests in σ by $b(\sigma)$.

Definition 1. *For an input σ, if $s(\sigma)/b(\sigma) \geq d$ $(b(\sigma) \neq 0)$ for a real number $d \geq 0$, σ is called d-slack. The slackness degree $d(\sigma)$ is defined as the maximum d such that σ is d-slack.*

The slackness degree describes how busy the inputs are. Clearly if $d(\sigma)$ is larger, σ has more slack. We will optimize DRA under the assumption that an input is d-slack without knowing the value of d.

We consider asymptotic performance, and assume that σ is large enough. In other words σ has a sufficient number of busy requests, i.e., $b(\sigma) = \omega(1)$ if $b(\sigma) \neq 0$.

Note that for $b(\sigma) = 0$ (i.e., all arrivals are slack), we can easily get the upper bound of the competitive ratio of $1 + \frac{\sum_{i=1}^{|\sigma|} x_i}{|\sigma| a}$, which is close to 1 when $|\sigma|$ is large and $\lim_{i\to\infty} x_i = 0$. This case is so particular that we ignore it in the following.

We will show that it is sufficient to consider inputs which end with a busy request:

For a detailed analysis, let us separate an input $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$ into some ($b(\sigma)$ or $b(\sigma)+1$) blocks as follows. Assume that $t_{b_1}, t_{b_2}, \ldots, t_{b_{b(\sigma)}}$ ($0 \le b_1 < b_2 < \cdots < b_{b(\sigma)} \le n$) are the busy requests in σ. The blocks are defined as $B_1 = \{t_1, \ldots, t_{b_1}\}$, $B_2 = \{t_{b_1+1}, \ldots, t_{b_2}\}$, $\ldots$, $B_{b(\sigma)} = \{t_{b_{b(\sigma)-1}+1}, \ldots, t_{b_{b(\sigma)}}\}$. If $b_{b(\sigma)} < n$, then $B_{b(\sigma)+1} = \{t_{b_{b(\sigma)}+1}, \ldots, t_n\}$ also exists. For analyzing the worst case competitive ratio we will show that the final block $B_{b(\sigma)+1}$ can be ignored even if it exists. Let $\beta(\sigma)$ be the number of blocks in σ. (Then $\beta(\sigma) = b(\sigma)$ or $b(\sigma)+1$.) Let $s(B_i)$ be the number of slack requests in block B_i.

Lemma 1. *If $s(B_{b(\sigma)}) \le s(B_{b(\sigma)+1}) - 2$ holds, then $\mathcal{R}_{DRA}(\sigma) \le \mathcal{R}_{DRA}(\sigma')$, where σ' is obtained from σ by exchanging $t_{b_{b(\sigma)}}$ with $t_{b_{b(\sigma)}+1}$, i.e., $\sigma' = \langle t_1, \ldots, t_{b_{b(\sigma)}-1}, t_{b_{b(\sigma)}+1}, t_{b_{b(\sigma)}}, t_{b_{b(\sigma)}+2}, \ldots, t_n \rangle$. (Note that $t_{b_{b(\sigma)}+1}$ is a slack request from $s(B_{b(\sigma)})$ (≥ 2).)*

Proof. The competitive ratio of DRA for σ' is

$$\mathcal{R}_{\mathrm{DRA}}(\sigma') = \frac{DRA(\sigma) - x_{s(B_{b(\sigma)+1})} + x_{s(B_{b(\sigma)})+2}}{OPT(\sigma) - x_{s(B_{b(\sigma)})+1} + x_{s(B_{b(\sigma)})+2}}.$$

Since $x_1, x_2, \ldots$ is a non-increasing sequence and $s(B_{b(\sigma)}) \le s(B_{b(\sigma)+1}) - 2$, $x_{s(B_{b(\sigma)})+2} \ge x_{s(B_{b(\sigma)+1})}$ and $x_{s(B_{b(\sigma)})+1} \ge x_{s(B_{b(\sigma)})+2}$ hold. Thus we get

$$\mathcal{R}_{\mathrm{DRA}}(\sigma') = \frac{DRA(\sigma) - x_{s(B_{b(\sigma)+1})} + x_{s(B_{b(\sigma)})+2}}{OPT(\sigma) - x_{s(B_{b(\sigma)})+1} + x_{s(B_{b(\sigma)})+2}} \ge \frac{DRA(\sigma)}{OPT(\sigma)} = \mathcal{R}_{\mathrm{DRA}}(\sigma).$$

□

Lemma 2. *For any $d \ge 0$ and sufficiently long inputs, there exists an input which finishes with a busy request and gives the worst competitive ratio in the same slackness degree d.*

Proof. By Lemma 1 for a d-slack input σ we can shift the last busy request later as long as the last two blocks satisfy $s(B_{b(\sigma)}) \le s(B_{b(\sigma)+1}) - 2$ without decreasing the competitive ratio (Operation 1). We can clearly exchange the two subsequences in σ which begin with a slack request and end with a busy request without changing the competitive ratio (Operation 2).

When we apply Operation 1 and Operation 2 for a sufficiently long input σ repeatedly and let the result be σ^*, which is d-slack and gives the worst competitive ratio, approximately we can assume that σ^* finishes with a busy request. □

Lemma 3. *For any input σ, if each x_i satisfies inequality (1) then $\mathcal{R}_{DRA}(\sigma) \le 2 + \varepsilon$.*

Proof. In Observations 1 and 2, the given input is clearly the worst for the competitive ratio among one-block input σ (i.e., σ includes one busy requests at the end) and the slackness degree is fixed. This means $\mathcal{R}_{\mathrm{DRA}} \le 2 + \varepsilon$ for any one-block input. From Lemma 2, $\mathcal{R}_{\mathrm{DRA}} \le 2 + \varepsilon$ for any long enough input. □

Lemma 4. *For a sufficiently long input σ, if σ is d-slack ($d > 0$) and $x_1, x_2, \ldots$ ($x_i \leq a$) satisfy inequality (1), the following inequality holds:*

$$\mathcal{R}_{DRA}(\sigma) \leq 1 + \frac{1}{d} + \frac{\sum_{i=1}^{\infty} x_i}{ad}. \tag{2}$$

And the equality holds for $d \geq h - 1$ where $h = \min\{i \mid x_i = 0\}$.

Proof. To analyze the worst case input, we define $\sigma(k)$ as an input in which one busy request arrives after $(k-1)$ slack requests, where $k = 1, 2, \ldots$, is any positive integer. Then we find that all the worst case input instances are described as the combination of $\sigma(k)$ by Lemma 2. Let the combination of them be σ_w, which can be represented by a sequence of $\sigma(\cdot)$, i.e., $\sigma_w = \sigma(f(1))\sigma(f(2))\cdots\sigma(f(n))$ where $n = b(\sigma_w)$, and each $f(i)$ is a positive integer ($i = 1, \ldots, n$).

Against this input, DRA must pay the switching cost for all the requests. The cost of DRA for σ_w is $a + \sum_{i=1}^{n} \left\{ \sum_{j=1}^{f(i)} (x_j + a) \right\} + x_1$. OPT keeps the ON-state during $x_{f(i)}$ for the last input in each $\sigma(f(i))$ and switches to OFF immediately for the other inputs. The cost is $a + \sum_{i=1}^{n} x_{f(i)} + \sum_{i=1}^{n} (f(i) - 1)a$. Therefore the competitive ratio is

$$\begin{aligned}
\mathcal{R}_{\mathrm{DRA}}(\sigma_w) &= \frac{a + \sum_{i=1}^{n} (\sum_{j=1}^{f(i)} (x_j + a)) + x_1}{a + \sum_{i=1}^{n} x_{f(i)} + \sum_{i=1}^{n} (f(i) - 1)a} \\
&\leq \frac{a + \sum_{i=1}^{n} \sum_{j=1}^{f(i)-1} x_j + \sum_{i=1}^{n} f(i)a + x_1}{a + \sum_{i=1}^{n} (f(i) - 1)a} \\
&\leq \frac{a + n \sum_{i=1}^{\infty} x_i + \sum_{i=1}^{n} (f(i) - 1)a + an + x_1}{a + \sum_{i=1}^{n} (f(i) - 1)a}.
\end{aligned}$$

Since $\sum_{i=1}^{n} (f(i) - 1)/n = s(\sigma_w)/b(\sigma_w) \geq d$ and σ_w is d-slack, we have

$$\begin{aligned}
\mathcal{R}_{\mathrm{DRA}}(\sigma) \leq \mathcal{R}_{\mathrm{DRA}}(\sigma_w) &\leq 1 + \frac{\sum_{i=1}^{\infty} x_i + a + x_1/n}{a/n + ad} \\
&\overset{(n \to \infty)}{\to} 1 + \frac{1}{d} + \frac{\sum_{i=1}^{\infty} x_i}{ad}.
\end{aligned}$$

The inequalities are tight when $\sum_{i=1}^{n} x_{f(i)} = 0$ and the slackness degree of σ_w is just d, and such input exists only when $\sum_{i=1}^{n} f(i)/n \geq h$. Thus for a sufficiently long input when $d \geq h - 1$, we find that the bound is tight. □

Note that even for 0-slack inputs ($s(\sigma) = 0$), if $x_1, x_2, \ldots$ satisfy (1), the competitive ratio is guaranteed to be $2 + \varepsilon$ according to Observation 1.

We get the upper bound of the worst competitive ratio with parameter d. To minimize it, we should minimize each x_i such that they satisfy (1).

Theorem 1. *We set the coefficients x_i as $x_i = \max\{-\varepsilon\,((2+\varepsilon)/(1+\varepsilon))^i\, a + (1+\varepsilon)a,\ 0\}$. Then for any sufficiently long d-slack input σ, DRA guarantees the following competitive ratio:*

$$\mathcal{R}_{DRA}(\Sigma_d) = \min\left\{ 1 + \frac{1}{d} + \frac{\sum_{i=1}^{h-1} x_i}{ad},\ 2 + \varepsilon \right\}, \tag{3}$$

where Σ_d is the set of sufficiently long d-slack inputs, and $h = \lfloor(\log(1+\varepsilon)-\log\varepsilon)/(\log(2+\varepsilon)-\log(1+\varepsilon))\rfloor + 1$.

Proof. Let h be defined as in Lemma 4. Then the value of h is obtained as shown above. From Lemmas 4 and 3 we get

$$\mathcal{R}_{\mathrm{DRA}}(\Sigma_d) \le \min\left\{1 + \frac{1}{d} + \frac{\sum_{i=1}^{\infty} x_i}{ad},\ 2+\varepsilon\right\}.$$

To optimize the competitive ratio we should minimize each x_i in range of satisfying inequality (1). So we get

$$x_i = \begin{cases} -\varepsilon\left(\dfrac{2+\varepsilon}{1+\varepsilon}\right)^i a + (1+\varepsilon)a & \text{if } i < h, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

This means $\sum_{i=1}^{\infty} x_i = \sum_{i=1}^{h-1} x_i$.

Furthermore, from Lemma 4, when $d \ge h-1$ there are inputs which hold the equation in (2) tightly. On the other hand, from Lemma 3, when $d < h-1$, there are inputs such that DRA uses only $x_1, \ldots, x_d$ (i.e., they satisfy (1) tightly.) and then achieve $2+\varepsilon$-competitive ratio tightly. Therefore we obtain the desired equation (3). □

From this, we will call the *DRA* satisfying the condition of Theorem 1 *the optimal DRA* (*ODRA*).

Corollary 1. *For the value that* $0 < \varepsilon < 0.2$,

$$\mathcal{R}_{ODRA} \le \min\left\{1 + \frac{(1+\varepsilon)^2 + 2(1+\varepsilon)\log\frac{1}{\varepsilon}}{d},\ 2+\varepsilon\right\}.$$

We also get such a heuristic bound, but skip the details of proof. If $d \to \infty$ then $\mathcal{R}_{\mathrm{ODRA}} \to 1$. Therefore we confirm that the competitive ratio is close to 1 when the frequency of requests within any time period is small enough.

Note that this algorithm works certainly without information of the input σ. Since we can define the d-slackness from some period from the entire input, we can evaluate the competitive ratio considering a part when the slackness degree changes.

4 Queueing Analysis

4.1 Analysis

In this section, we analyze the cost performance of DRA using queueing theory. We assume that customers arrive at the system according to a Poisson process with rate λ. The sojourn time of a customer is independently and identically distributed (i.i.d.) with a general distribution with mean $1/\mu$. As we mentioned before, the system capacity is infinity. Then the system we consider here is an M/G/∞ queueing model.

In the M/G/∞ model, the busy period is defined as the time interval during which the number of customers in the system is greater than zero, while in the idle period, no customers are in the system. For analytical simplicity, we assume that the system is in equilibrium at time 0, and that the first busy period starts at time 0. Let B_n and I_n denote the nth busy period of the system and the nth idle period, respectively. Note that both busy periods and idle periods are i.i.d., and hence independent of n. The mean busy period and the mean idle period of the M/G/∞ system are given by

$$E[B_n] = \frac{e^{\rho} - 1}{\lambda} (\equiv E[B]), \quad E[I_n] = \frac{1}{\lambda} (\equiv E[I]), \tag{5}$$

respectively, where $\rho = \lambda/\mu$. We define the nth cycle as the time interval consisting of B_n and I_n.

The power control process under DRA with coefficients given by (4) evolves as follows. When the first busy period B_1 starts, the initial power cost a is required. During the busy period, the power cost per unit time is one. When B_1 ends, the system is kept in the ON-state for the standby time of x_1. Note that x_1 is the power cost of the first idle period I_1. If $I_1 > a$, the next standby time for I_2 is set to x_2. If $I_1 \leq a$, then the standby time for I_2 is initialized to x_1. Similarly, if $I_1 > a$ and $I_2 > a$, then the standby time for I_3 is set to x_3, while if $I_1 > a$ and $I_2 \leq a$, the standby time for I_3 is initialized to x_1, and so on. In the following, the time interval from the beginning of the busy period with x_1 standby time to the end of the idle period which is smaller than a is referred to as the reset interval.

Let L (≥ 1) denote the number of cycles in a reset interval. Consider the amount of power consumption during a reset interval. When the number of cycles in the reset interval is $L = k$, the amount of power consumption is given by the power consumption for k busy periods and k standby times. Let T_k denote the total amount of power consumption of standby times in the reset interval consisting of k cycles. We obtain

$$T_k = \begin{cases} \sum_{i=1}^{k-1} x_i + I_k \cdot 1_{\{I_k \leq x_k\}}, & k = 1, 2, \ldots, h-1, \\ \sum_{i=1}^{h-1} x_i, & k \geq h, \end{cases}$$

where 1_χ is the indicator function of event χ. Then we have the following lemma.

Lemma 5. *The mean of the total amount of power consumption of standby times in a reset interval $E[T_L]$ is given by*

$$\begin{aligned} E[T_L] = {} & \epsilon a(2+\epsilon)e^{-\lambda a}(1-e^{-\lambda a})^{h-2} \\ & -\epsilon a(2+\epsilon)(1-e^{-\lambda a})\frac{(2+\epsilon)e^{-\lambda a}}{1+\epsilon-(2+\epsilon)e^{-\lambda a}} \left\{1 - \left(\frac{2+\epsilon}{1+\epsilon} \cdot e^{-\lambda a}\right)^{h-2}\right\} \\ & +(1+\epsilon)a \cdot \frac{e^{-\lambda a}}{1-e^{-\lambda a}} \left\{1 - (h-1)e^{-\lambda a(h-2)} + (h-2)e^{-\lambda a(h-1)}\right\} \\ & +\frac{1}{\lambda} \cdot \frac{1-e^{-\lambda a(h-1)}}{1-e^{-\lambda a}} - \sum_{k=1}^{h-1} e^{-\lambda a(k-1)} \left(\frac{1}{\lambda} + x_k\right) e^{-\lambda x_k} \end{aligned}$$

$$+\left[\epsilon a(2+\epsilon)\left\{1-\left(\frac{2+\epsilon}{1+\epsilon}\right)^{h-1}\right\}+(1+\epsilon)a(h-1)\right]\cdot e^{-\lambda a(h-1)}. \quad (6)$$

From the Poisson arrival assumption we can obtain this formula, however, we skip the details of proof.

Let Q_{ODRA} denote the mean power-consumption cost per unit time. Then we obtain the following theorem.

Theorem 2. *Q_{ODRA} is given by*

$$Q_{\mathrm{ODRA}}=\frac{1}{e^{\rho}}\left[\lambda a\left\{(1-e^{-\lambda a})\cdot\sum_{k=1}^{h-1}e^{-\lambda\{a(k-1)+x_k\}}+e^{-\lambda a(h-1)}\right\}\right.$$
$$\left.+e^{\rho}-1+\lambda(1-e^{-\lambda a})E[T_L]\right], \quad (7)$$

where $E[T_L]$ is given by (6).

Proof. Let R denote the reset interval. Using L, the number of cycles in a reset interval, we obtain $R=\sum_{n=1}^{L}(B_n+I_n)$. We skip the details of analysis, but we can obtain the mean reset interval $E[R]$ as follows.

$$E[R]=E\left[\sum_{n=1}^{L}B_n\right]+E\left[\sum_{n=1}^{L}I_n\right]=\frac{e^{\rho}}{\lambda(1-e^{-\lambda a})}.$$

Let $W(k)$ denote the total amount of power consumption for a reset interval which consists of k cycles. $W(k)$ is given by $W(k)=(k-1)a+a\cdot 1_{\{x_k<I_k\le a\}}+\sum_{i=1}^{k}B_i+T_k$. Taking the mean of $W(L)$ yields

$$E[W(L)]=a(E[L]-1)+aE[1_{\{x_k<I_k\le a\}}]+E[L]E[B]+E[T_L].$$

Note that reset intervals are i.i.d. and that the amount of power consumption during the reset interval is also i.i.d. Therefore, we have $Q_{\mathrm{ODRA}}=E[W(L)]/E[R]$ from the renewal-reward theorem [19]. □

4.2 Numerical Examples

In this section, we present some numerical examples using the analysis shown in the previous section. In order to validate the analysis, we also perform some simulations with the algorithms and compare the results obtained with the ones from the analysis.

Algorithms Compared to ODRA. For comparing with *ODRA* we consider the following algorithm, which is a simple variant of DRA.

Given parameter k let ALG(k) be the DRA algorithm where $x_1=x_2=\ldots=x_k=a$, $x_{k+1}=x_{k+2}=\ldots=0$.

Table 1. Basic parameters

Parameter	Value
Value of a	1, 3, 10 [unit]
ALG(10) parameter k	10
Value of ϵ	0.1, 0.01, 0.001
Consuming cost while in ON-state	1 [unit]
Customer arrival rate λ	0.001, 0.01, 0.1, 0.5, 0.99
Mean sojourn time $1/\mu$	1
Number of events	100000
Number of simulations	100

The worst case competitive ratio of $ALG(k)$ is $2+\frac{1}{k}$, which can be obtained easily. We consider three cases: $ALG(\infty)$ (all x_i are equal to a), $ALG(1)$, $ALG(10)$. Note that $ALG(\infty)$ is equal to $OWCR$. Let Q_ξ denote the mean of the power-consumption cost per unit time when the algorithm of $\xi \in \{OPT, ALG(\infty), ALG(1), ALG(10)\}$ is employed. Q_ξ's can be derived in a straightforward manner, and we obtain

$$
\begin{aligned}
Q_{\mathrm{OPT}} &= 1 - e^{-(a\lambda+\rho)}, \\
Q_{\mathrm{ALG}(\infty)} &= 1 + (a\lambda - 1)e^{-(a\lambda+\rho)}, \\
Q_{\mathrm{ALG}(1)} &= 1 + 2(a\lambda - 1)e^{-(a\lambda+\rho)} - (a\lambda - 1)e^{-(2a\lambda+\rho)}, \\
Q_{\mathrm{ALG}(10)} &= 1 + (a\lambda - 1)e^{-(a\lambda+\rho)} + (a\lambda - 1)e^{-(ka\lambda+\rho)} - (a\lambda - 1)e^{-((k+1)a\lambda+\rho)}.
\end{aligned}
$$

We omit detailed derivations of the above equations due to the page limitation.

Competitive Ratios. We calculated the average power consumption per unit time of each algorithm and the competitive ratio with Q_{opt}. We use the basic set of parameters shown in Table 1. The analytical results are shown in Table 2. We also conducted experiments with Monte Carlo simulation, in order to validate the results obtained through the analysis. (Skipping the details of simulation results.) The analytical results exhibit good agreement with simulation, and this validates the analytical derivations for Q_ξ's.

We can observe that the performance of these algorithms is almost the same when the system is congested, e.g., $\rho \geq 0.5$. The reason is clearly that it never be OFF. The difference appears when the system becomes slack. Especially $ALG(\infty)$, which must be the optimal worst competitive ratio algorithm ($OWCR$), shows very bad average competitive ratio (e.g., it is around 1.9 for $\rho = 0.01, a = 10$). $ALG(1)$ shows the best average competitive ratio in every case. However it's worst competitive ratio is 3, i.e., it may perform badly for adversary inputs. Our algorithm $ODRA$ performs almost the same as $ALG(1)$ in every case. From these results, we can observe that $ODRA$ has good performance not only in the worst case but also in average case.

Table 2. Analytical results of competitive ratios

ρ	a	$Q_{ALG(\infty)}/Q_{\mathrm{OPT}}$	$Q_{ALG(1)}/Q_{\mathrm{OPT}}$	$Q_{ALG(10)}/Q_{\mathrm{OPT}}$	$Q_{\mathrm{ODRA}}/Q_{\mathrm{OPT}}$		
					$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.001$
CR_{worst}		2	3	2.1	2.1	2.01	2.001
0.001	1	1.49950	1.00075	1.00052	1.001271	1.002656	1.004223
0.001	3	1.74850	1.00336	1.00232	1.005697	1.011871	1.018820
0.001	10	1.90410	1.01350	1.09015	1.022692	1.046865	1.073527
0.01	1	1.49501	1.00739	1.04936	1.012424	1.025660	1.040258
0.01	3	1.73510	1.03264	1.19886	1.053683	1.108154	1.164850
0.01	10	1.86001	1.12344	1.15605	1.188589	1.350484	1.488426
0.1	1	1.45167	1.06483	1.29439	1.099045	1.184070	1.256516
0.1	3	1.60997	1.24109	1.58518	1.304441	1.472030	1.555845
0.1	10	1.49896	1.49896	1.49896	1.400979	1.479744	1.496660
0.5	1	1.29099	1.17649	1.28972	1.189663	1.260917	1.284193
0.5	3	1.23478	1.29557	1.24777	1.200283	1.228261	1.234040
0.5	10	1.02052	1.03682	1.02052	1.019213	1.020304	1.020495
0.99	1	1.15672	1.15662	1.15672	1.127199	1.152416	1.157843
0.99	3	1.05614	1.09169	1.05614	1.052462	1.056840	1.057616
0.99	10	1.00017	1.00032	1.00017	1.000182	1.000183	1.000184

5 Conclusions

We have introduced the concept of slackness degree, which reflects the frequency of requests, and developed the "optimal" online algorithm under this concept. We strongly believe that it is important to consider inputs of problems based on the real world and to design more practical algorithms in online problems. In future work, we plan to consider the randomized version and the multi-state version of this problem.

Acknowledgments. This research has been carried out in collaboration with the "Consumer Electronics Network Eco Management" project sponsored by Panasonic Corporation. We would like to thank the project members, Mr. Toshiya Naka, Mr. Hideyuki Yoshida and Mr. Kazuhiro Aizu. We also would like to thank Prof. Hiroshi Fujiwara of Toyohashi University of Technology for his valuable comments on power consumption problems and helpful discussions.

References

1. Augustine, J., Irani, S., Swamy, C.: Optimal power-down strategies. In: Proc. 45th Symp. Foundations of Computer Science (FOCS), pp. 530–539. IEEE (2004)
2. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica 11, 73–91 (1994)
3. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. J. Comput. Systems Sci. 50, 244–258 (1995)

4. Boyar, J., Krarup, S., Nielsen, M.N.: Seat reservation allowing seat changes. J. Algorithms 52, 169–192 (2004)
5. Chrobak, M.: Sigact news online algorithms column 8. SIGACT News 36, 67–81 (2005)
6. Chung, E., Benini, L., Bogliolo, A.: Dynamic power management for non-stationary service requests. In: Proceedings of the Design and Automation and Test in Europe Conference and Exhibition, pp. 77–81 (1999)
7. Eggers, S.J., Katz, R.H.: Evaluating the performance of four snooping cache coherency protocols. In: Proc. 16th International Symp. on Computer Architecture (ISCA). IEEE (1989)
8. Irani, S., Gupta, R., Shukla, S.: Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In: DATE 2002: Proceedings of the Conference on Design, Automation and Test in Europe, p. 117. IEEE Computer Society, Washington, DC, USA (2002)
9. Irani, S., Pruhs, K.R.: Algorithmic problems in power management. ACM SIGACT News (2005)
10. Karlin, A.R., Kenyon, C., Randall, D.: Dynamic tcp acknowledgement and other stories about $e/(e-1)$. In: Proc. 33rd STOC, pp. 502–509. ACM (2001)
11. Karlin, A., Manasse, M., McGeoch, L., Owicki, S.: Competitive randomized algorithms for nonuniform problems. Algorithmica 11, 542–571 (1994)
12. Karlin, A., Manasse, M., Rudolph, L., Sleator, D.: Competitive snoopy caching. Algorithmica 3, 79–119 (1988)
13. Kenyon, C.: Best-fit bin-packing with random order. In: Proc. 7th Symp. on Discrete Algorithms (SODA), pp. 359–364. ACM/SIAM (1996)
14. Koutsoupias, E., Papadimitriou, C.: Beyond competitive analysis. SIAM J. Comput. 30, 300–317 (2000)
15. Lotker, Z., Patt-Shamir, B., Rawitz, D.: Rent, lease or buy: Randomized algorithms for multislope ski rental. In: Albers, S., Weil, P. (eds.) 25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008). Leibniz International Proceedings in Informatics (LIPIcs), vol. 1, pp. 503–514. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2008)
16. Panagiotou, K., Souza, A.: On adequate performance measures for paging. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC 2006, pp. 487–496. ACM, New York (2006)
17. Phillips, S., Westbrook, J.: Competitive analysis and beyond. In: Algorithms and Theory of Computation Handbook, ch.10. CRC Press (1999)
18. Ramanathan, D., Irani, S., Gupta, R.: Latency effects of system level power management algorithms. In: Proceedings of the IEEE International Conference on Computer Aided Design (2000)
19. Wolff, R.W.: Stochastic modeling and the theory of queues. Prentice-Hall (1989)

Single Approximation for Biobjective Max TSP*

Cristina Bazgan[1,2,3], Laurent Gourvès[1,2],
Jérôme Monnot[1,2], and Fanny Pascual[4]

[1] Université Paris-Dauphine, LAMSADE,
Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France
[2] CNRS, UMR 7243
[3] Institut Universitaire de France
[4] Université Pierre et Marie Curie, LIP6, 4 place Jussieu, 75005 Paris, France
{bazgan,laurent.gourves,monnot}@lamsade.dauphine.fr,
fanny.pascual@lip6.fr

Abstract. We propose an algorithm which returns a single Hamiltonian cycle with performance guarantee on both objectives. The algorithm is analysed in three cases. When both (resp. at least one) objective function(s) fulfill(s) the triangle inequality, the approximation ratio is $\frac{5}{12}-\varepsilon \approx 0.41$ (resp. $\frac{3}{8}-\varepsilon$). When the triangle inequality is not assumed on any objective function, the algorithm is $\frac{1+2\sqrt{2}}{14}-\varepsilon \approx 0.27$-approximate.

1 Introduction

The traveling salesman problem (TSP) is one of the most studied problems in combinatorial optimization. Given an undirected complete graph with weights on the edges, the problem consists of finding a Hamiltonian cycle (also called tour) of maximum or minimum total weight, defined as the sum of its edges' weight. In this paper we study the approximation of the biobjective maximization version, Biobjective Max TSP. In this case every edge has two weights and the total weight of a tour is a couple defined as the componentwise sum of its edges' weights. We are interested in the existence and the computation in polynomial time of a *single* tour with simultaneous performance guarantees on the two objectives. Our work falls into a recent stream of research on the approximability of multiobjective optimization problems [21,20,18,10,5,11,3,1,6] where multiobjective TSP takes a prominent place [2,4,16,7,13,14].

In many real optimization problems not only one objective function is considered but several ones (see [9] about multiobjective combinatorial optimization). This is also the case for TSP where we might want to minimize the travel time, the cost or to maximize the profit, the number of viewpoints along the way etc. This gives rise to Multiobjective TSP. Unfortunately it is unlikely that optimality is met simultaneously by a single feasible solution on all objectives. However there always exists a set of efficient (also called Pareto optimal) solutions for

* This research has been supported by the project ANR-09-BLAN-0361 GUaranteed Efficiency for PAReto optimal solutions Determination (GUEPARD).

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 49–62, 2012.

which any improvement on an objective induces a deterioration of (at least) another one.

Generating the whole set of efficient solutions is a major challenge in multiobjective combinatorial optimization. However, even for moderately-sized problems, it is usually computationally prohibitive to identify the efficient set for two major reasons. First, the number of efficient solutions can be very large. Second, the associated decision version is often NP-complete, even if the underlying single objective problem is polynomial time solvable. To handle these two difficulties, researchers have been interested in developing approximation algorithms with a priori provable performance guarantees.

Given a positive real $\rho \leq 1$, and considering that all objectives have to be maximized, a ρ-approximation of the set of efficient solutions is a set of solutions that includes, for each efficient solution, a solution that approximates it within a factor ρ on *all* objectives. The ρ-approximation typically contains several incomparable solutions and it is assumed that one solution is selected with the help of a, yet unkown, *a posteriori* decision process.

One of the most important results concerning the approximation of multiobjective problems was given by Papadimitriou and Yannakakis [18]: under certain general assumptions, multiobjective optimization problems always have at least one $(1 - \varepsilon)$-approximation of size polynomial in the size of the instance and $1/\varepsilon$, for any given accuracy $\varepsilon > 0$. This result makes the computation of approximate efficient sets of multiobjective problems accessible to polynomial time algorithms.

Nevertheless the efficient set is not the unique object that one can approximate. A popular approach in multiobjective optimization consists in optimizing only one objective while the others are turned into budget constraints [21,20,11,6]. Budget constraints come from an *a priori* decision process which restricts the set of desired solutions. It is noteworthy that the efficient set approach and the budget approach are essentially the same [18].

In another popular approach, no decision process is sought. The goal is to compute a single solution which approximates a vector composed of the optimal values on every objective taken separately [22,19,3,1]. Contrasting with the previous approaches, this framework aims at approximating an ideal point which is the image of a not necessarily feasible solution. Hence no ρ-approximation for every ρ is guaranteed to exist. Note that the ideal point approach and the efficient set approach restricted to sets of size 1 coincide. The former is a particular case of the latter. Since generating several solutions allows better approximations than what a single solution can achieve, approximation ratios under the respective approaches are not directly comparable.

Previous results for the multiobjective TSP are known; most of them follow the efficient set approach, approximating the Pareto set with two or more solutions, but some of them use the ideal point approach. In this article we exclusively follow the ideal point approach and provide deterministic approximation algorithms whose performance guarantees improve on previous results.

Previous Results. Multiobjective TSP is well studied from the approximation point of view. Manthey and Ram [16] follow the efficient set approach for several variants of multiobjective Min TSP. In particular they generalize the well known tree doubling algorithm to provide a $(2+\epsilon)$-approximation of the efficient set. The other results of [16] deal with multiobjective Min TSP with the sharpened triangle inequality and multiobjective Min TSP with distance 1 or 2. This latter problem is investigated in [2,4] under the efficient set approach.

More rencently Bläser et al. [7] study the multiobjective Max TSP with k objective functions. Using the efficient set approach they devise randomized approximation algorithms with ratios $\frac{1}{k}-\epsilon$ and $\frac{1}{k+1}-\epsilon$ for the symmetric and asymmetric versions respectively. Subsequently these results were significantly improved by Manthey [14] who provides randomized approximation algorithms, using the efficient set approach, with ratios $\frac{2}{3}-\epsilon$ and $\frac{1}{2}-\epsilon$ for the symmetric and asymmetric versions respectively. These algorithms use as a black box the randomized PTAS for min-weight matching given by Papadimitriou and Yannakakis [18]. Recently, Manthey [15] establishes deterministic approximation algorithms, using the efficient set approach, with ratios $\frac{1}{2k}-\epsilon$ and $\frac{1}{4k-2}-\epsilon$ for the symmetric and asymmetric versions respectively that can be improved for the biobjective case to ratios $\frac{3}{8}-\epsilon$ and $\frac{1}{4}-\epsilon$ respectively.

Manthey also investigates the approximation of Biobjective Max TSP under the ideal point approach [14,15], i.e. approximate efficient sets of size one. If the single objective Max TSP problem is ρ-approximable then Biobjective Max TSP is $\frac{\rho}{3}$-approximable with one solution [14]. Taking the best polynomial time approximation algorithms known so far for the symmetric Max TSP, he derives a $\frac{61}{243}$-approximate (resp. $\frac{7}{24}$-approximate) tour without (resp. with) the triangle inequality. The ratios come from a $\frac{61}{81}$-approximation and a 7/8-approximation given in [8] and [12] respectively. As mentioned very recently in [15], using a new $\frac{7}{9}$-approximation [17], the first ratio becomes $\frac{7}{27}$ instead of $\frac{61}{243}$. Another positive consequence of the general technique is that every biobjective instance admits a single $\frac{1}{3}$-approximate tour. From the negative side, Manthey [14] gives a 5 node non metric instance in which no single tour can be $(1/3+\epsilon)$-approximate $(\epsilon > 0)$, thus meeting the previous bound. To our best knowledge, no such upper bound is known for metric instances so it is still possible that a single ρ-approximate tour exists in biobjective Max TSP for some $\rho > 1/3$. Finally one can observe that known inapproximability results on the single objective Max TSP imply that the general technique is limited to provide biobjective $(1/3-\epsilon)$-approximation in polynomial time $(\epsilon > 0)$.

New Results. In this paper, we establish a general algorithm which computes a maximum value matching on each objective taken separately and combines them into a single Hamiltonian cycle having a performance guarantee on both objectives. The algorithm is analyzed in three cases. When both objective functions fulfill the triangle inequality, we obtain a $\frac{5}{12}-\epsilon \approx 0.41$-approximate algorithm which improves the aforementioned $\frac{7}{24}-\epsilon \approx 0.291$-approximation. In this case, we also propose a 4-node instance without any single $(\frac{1}{2}+\epsilon)$-approximate solution and a family of instances without any single $(\frac{3}{4}+\epsilon)$-approximate solution

when the number of nodes tends to infinity. If only one objective function fulfills the triangle inequality, we obtain a $(\frac{3}{8} - \epsilon)$-approximate algorithm. In the case where no objective function satisfies the triangle inequality, a quick analysis gives a ratio $1/4 - \epsilon$ but in a more accurate case analysis, we can show that the algorithm is $\frac{1+2\sqrt{2}}{14} - \epsilon \approx 0.27$-approximate, improving the aforementioned $\frac{7}{27} \approx 0.259$-approximation. An extension of Manthey's instance to any number of vertices precludes any $(\frac{1}{3} + \epsilon)$-approximate algorithm returning one solution.

The following table gives a summary of mentioned results on the biobjective Max TSP ($k = 2$). Approximations achieved with several solutions follow the Pareto set approach while those limited to one solution follow the ideal point approach.

Biobjective Max TSP			
	randomized algo.	deterministic algo.	this paper (deterministic)
general case	$2/3 - \epsilon$ [14] several solutions	$7/27 \approx 0.259$ one solution [14,17] $3/8 - \epsilon$ several solutions [15]	$\frac{1+2\sqrt{2}}{14} - \epsilon \approx 0.27$ one solution
metric case	$2/3 - \epsilon$ [14] several solutions	$7/24 \approx 0.291$ one solution [14,12]	$5/12 - \epsilon \approx 0.41$ one solution

Organization of the Article. In Section 2 we give definitions on the problems and concepts used throughout the article. In Section 3 we establish some non existence results which give upper bounds on possible approximation ratios under the ideal point approach. Section 4 presents a general algorithm for Biobjective Max TSP and its analysis in three cases depending on the (non) metric nature of the objective functions. In Section 5 we improve the analysis of the previous algorithm in the non metric case. Future works are provided in a final section. Due to space limitation some proofs are omitted.

2 Preliminaries

Let $G = (V, E)$ be a complete undirected graph with a nonnegative weight $w(e)$ on every edge $e \in E$ and $n = |V|$ vertices. The *weight* of a set of edges $E' \subseteq E$ is the sum of the weights of the edges in E' and is denoted by $w(E')$. An instance is metric if its weights satisfy the triangle inequality, namely $w(x, z) \leq w(x, y) + w(y, z)$ for all distinct vertices $x, y, z \in V$.

Max TSP is to find a Hamiltonian cycle or tour (i.e. a cycle that visits every vertex of the graph exactly once) of maximum weight in a complete graph. In the multiobjective Maximum Traveling Salesman Problem every edge is endowed with k nonnegative values. For the biobjective case ($k = 2$), each edge $e \in E$ has a nonnegative weight $w(e)$ and a nonnegative length $\ell(e)$. Similarly the length of a set of edges E', denoted by $\ell(E')$, is the sum of the lengths of its elements.

Each feasible tour T is represented in the objective space by its corresponding objective vector $(w(T), \ell(T))$. A tour T *dominates* a tour T' if and only if

$w(T) \geq w(T')$ and $\ell(T) \geq \ell(T')$ with at least one strict inequality. A tour T is *efficient* if and only if no other tour T' dominates T, and $(w(T), \ell(T))$ is said to be *non-dominated*. An *efficient set* contains, for each non-dominated vector, a corresponding efficient solution (no need to keep two tours having the same objective vector).

Unfortunately computing the *efficient set* of multiobjective Max TSP cannot be done in polynomial time, unless $P = NP$, so we are interested in its polynomial time computable approximations. For any $0 < \rho \leq 1$, a tour T ρ-approximates another tour T^* if and only if $w(T) \geq \rho w(T^*)$ and $\ell(T) \geq \rho \ell(T^*)$. A set of feasible tours $\mathcal{A}$ is a ρ-approximation of the efficient set $\mathcal{P}$ if for every $T^* \in \mathcal{P}$, there exists $T \in \mathcal{A}$ such that T ρ-approximates T^*. If $\mathcal{A}$ is reduced to a single tour, we say that we follow the ideal point approach.

Define opt_w (resp. opt_ℓ) as $\max_{T \in \mathcal{F}} w(T)$ (resp. $\max_{T \in \mathcal{F}} \ell(T)$) where $\mathcal{F}$ denotes the set of feasible tours. Under the ideal point approach, a tour T is a ρ-approximation if and only if $w(T) \geq \rho\ opt_w$ and $\ell(T) \geq \rho\ opt_\ell$.

3 Non Existence of a Single ρ-Approximate Solution

It is unlikely that every instance admits a single solution which is nearly optimal for w and ℓ at the same time. Thus instances without any ρ-approximate solution imply that no deterministic ρ-approximate algorithm (even exponential) exists.

If the triangle inequality is satisfied on both objectives, the example given in Figure 1 (left) shows that there does not *always* exist a $(\frac{1}{2} + \epsilon)$-approximate solution, for all $\epsilon > 0$. The three possible tours in this instance are indeed (a, b, c, d, a), (a, c, d, b, a), and (a, c, b, d, a) whose values are $(2, 2)$, $(2, 4)$, and $(4, 2)$. However this instance only contains 4 nodes so it does not prevent an algorithm to provide a $(0.5 + \epsilon)$-approximate solution for 5 nodes and more.

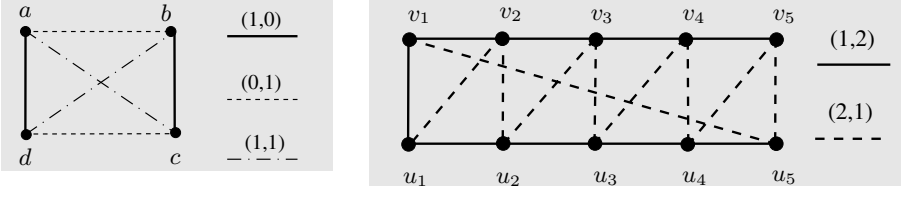


Fig. 1. (Left) There is no $(0.5 + \epsilon)$-approximate solution in this instance where every objective function satisfies the triangle inequality. (Right) Instance with $r = 5$ where non represented edges have value $(1, 1)$.

However one can build an instance which does not contain any $(\frac{3}{4} + \epsilon)$-approximate solution for n sufficiently large. The instance contains $2r$ nodes $\{v_1, \cdots, v_r\} \cup \{u_1, \cdots, u_r\}$. Edges (u_i, v_i) have value $(2, 1)$ for $i = 2, \cdots r$, see Figure 1 (right). Edges (u_i, v_{i+1}) have value $(2, 1)$ for $i = 1, \cdots r-1$. Edges (u_i, u_{i+1}) and (v_i, v_{i+1}) have value $(1, 2)$ for $i = 1, \cdots r-1$. Edges (u_1, v_1) and (u_r, v_1) have

value $(1,2)$ and $(2,1)$ respectively. Any other edge has value $(1,1)$. The coordinates being 1 or 2, the triangle inequality is satisfied. The tour containing all edges of value $(2,1)$ (resp. $(1,2)$) has value $(4r-1,2r)$ (resp. $(2r,4r-1)$) so the optimal weight/length is $4r-1$. Any given tour uses α edges with value $(2,1)$, β edges with value $(1,2)$ whereas $\alpha+\beta \leq 2r$. Its value is then $(2\alpha+\beta, 2\beta+\alpha) \leq (2r+\alpha, 2r+\beta)$. Observe that $\min\{2r+\alpha, 2r+\beta\} = 2r+\min\{\alpha,\beta\} \leq 3r$. Hence any tour is at most $\frac{3r}{4r-1}$-approximate.

If the objective functions do not necessarily fulfill the triangle inequality, Manthey [14] proved that for a K_5 there does not exist a $(\frac{1}{3}+\epsilon)$-approximate algorithm, for all $\epsilon > 0$. We can easily generalize his result to K_n with $n \geq 5$ in order to obtain an asymptotic result. For every $n \geq 5$, consider K_n where a fixed K_4 is decomposable into 2 Hamiltonian paths P_w and P_ℓ. For every edge $e \in E(K_n)$, set $w(e) = 1$ and $\ell(e) = 0$ if $e \in P_w$, $w(e) = 0$ and $\ell(e) = 1$ if $e \in P_\ell$ and $w(e) = 0$ and $\ell(e) = 0$ if $e \notin P_w \cup P_\ell$. We can check that there are four non-dominated tours T_i, $i = 1,\ldots,4$ with $w(T_1) = w(P_w) = 3$, $\ell(T_1) = \ell(P_w) = 0$, $w(T_2) = w(P_\ell) = 0$, $\ell(T_2) = \ell(P_\ell) = 3$, $w(T_3) = 2$, $\ell(T_3) = 1$ and $w(T_4) = 1$, $\ell(T_4) = 2$. In conclusion, a single solution never approximates the Pareto set of the biobjective Max TSP with ratio better than $1/3$ for K_n with $n \geq 5$.

4 A Generic Algorithm for Biobjective Max TSP

In this section, we present an algorithm for the Biobjective Max TSP. This algorithm is based on the combination of the edges of a maximum weight matching for the objective w and a maximum weight matching for the objective ℓ. The algorithm is as follows :

1. Build a maximum weight (resp. length) matching of G and denote it by M_w (resp. M_ℓ).
   The set of edges $M_w \cup M_\ell$ is made of p connected components $C_1,\ldots,C_p$. Each C_i is a cycle of even size, or a path of length at least one. Note that there is at most one path of length at least two in $M_w \cup M_\ell$ (because the graph is complete and we can assume that M_w are M_ℓ are of maximum size). Likewise, each path of length one is in $M_w \cap M_\ell$.
2. For each component C_i which is a cycle, remove the edge in $C_i \cap M_w$ which has a minimum weight.
   We thus obtain a set of paths, which is called a *partial tour*.
3. Add edges in order to connect these paths and obtain an Hamiltonian cycle of K_n (edges are added arbitrarily unless otherwise noted. This step is detailled inside the proofs when needed).

Let us now show that the Hamiltonian cycle obtained with this algorithm has a weight larger than or equal to $\alpha w(M_w)$ and a length larger than or equal to $\alpha\ell(M_\ell)$, where $0 < \alpha \leq 1$. We will determine the value of α in a general graph (cf. Lemma 1), in a graph where one objective function (w.l.o.g. w) fulfills the triangle inequality (cf. Lemma 2), and in a graph where both objective functions fulfill the triangle inequality (cf. Lemma 3).

Lemma 1. *Step 1 and 2 of the algorithm build in polynomial time a partial tour on K_n with weight at least $\frac{1}{2}w(M_w)$ and length at least $\frac{1}{2}\ell(M_\ell)$.*

Proof. For each component C_i which is a cycle, step 2 of the algorithm removes the edge in $C_i \cap M_w$ with minimum weight. Since $|C_i \cap M_w| \geq 2$ the loss in weight is at most $w(C_i \cap M_w)/2$. The resulting set of edges is a partial tour of weight at least $\frac{1}{2}\sum_{i=1}^{p} w(C_i \cap M_w) = \frac{1}{2}w(M_w)$ and length $\sum_{i=1}^{p} \ell(C_i \cap M_\ell) = \ell(M_\ell)$. □

In the following Lemmas we consider two cases:

- Case 1: at the end of Step 1 of the algorithm, every component C_i is a cycle
- Case 2: at the end of Step 1 of the algorithm, at least one component C_i is a cycle and at least one component $C_{i'}$ is not a cycle.

If no component is a cycle then we are already done since the set of edges is then a partial tour of weight $w(M_w)$ and length $\ell(M_\ell)$.

Lemma 2. *Assuming that w satisfies the triangle inequality, we can build in polynomial time a partial tour on K_n with weight at least $\frac{3}{4}w(M_w)$ and length at least $\frac{3}{4}\ell(M_\ell)$.*

Proof. We distinguish two cases depending on the value of p that is the number of connected components of $M_w \cup M_\ell$. If $p = 1$ then C_1 is either a tour or a cycle on $n-1$ nodes (in this case n is odd) with weight at least $w(M_w)$ and length at least $\ell(M_\ell)$. If C_1 is a cycle on $n-1$ nodes, let x be the isolated node. Then by replacing any edge $(u, v) \in M_w$ by $(u, x), (x, v)$, we get a tour C' of K_n satisfying $w(C') \geq w(C_1) \geq w(M_w)$ due to the triangle inequality and $\ell(C') \geq \ell(M_\ell)$.

Let us now consider the case where $p \geq 2$. Assume that case 1 occurs, that is each component C_i is a cycle and thus it contains at least four edges. Since $p \geq 2$ and $|M_\ell \cap C_i| \geq 2$ for each C_i we have $|M_\ell| \geq 4$. It follows that if $e \in M_\ell$ is an edge of minimum length among the edges of M_ℓ, then $\ell(e) \leq \ell(M_\ell)/4$. Thus, by deleting e, we are in case 2 since $\cup_{i=1}^{p} C_i \setminus \{e\}$ contain at least one cycle and at least one path with $w(\cup_{i=1}^{p} C_i \setminus \{e\}) \geq w(M_w)$ and

$$\ell(\cup_{i=1}^{p} C_i \setminus \{e\}) \geq 3\ell(M_\ell)/4 \tag{1}$$

Now, assume that case 2 occurs. By renaming the connected components, we can assume that there is an integer $r \in \{1, \ldots, p\}$ such that C_i for $i \geq r$ is not a cycle whereas C_i for $1 \leq i < r$ is a cycle. Let x and y be the two extremities of C_r. Proceed repeatedly as follows, for $i = r-1$ down to 1. Remove an edge of minimum weight in $M_w \cap C_i$ and call it (v_1^i, v_2^i). Add the edge with maximum weight between (v_1^i, x) and (v_2^i, x). If $w(v_1^i, x) \geq w(v_2^i, x)$ then $x := v_2^i$, otherwise $x := v_1^i$. By this way the procedure maintains a path with extremities x and y, while reducing the number of cycles. At the end of the procedure we get a partial tour that is the union between a path and $\cup_{i=r}^{p} C_i$. Using the triangle inequality we know that $\max\{w(v_1^i, x), w(v_2^i, x)\} \geq (w(v_1^i, x) + w(v_2^i, x))/2 \geq w(v_1^i, v_2^i)/2$, meaning that each time an edge (v_1^i, v_2^i) is removed ($i \in \{1, \ldots, r-1\}$), another

one with at least half its weight is added so, in total, the loss in weight is bounded by $\frac{1}{2}\sum_{i=1}^{r-1} w(v_1^i, v_2^i)$. Since $|M_w \cap C_i| \geq 2$ we deduce that $w(v_1^i, v_2^i) \leq w(M_w \cap C_i)/2$. Summing up the previous inequality, we deduce that $\sum_{i=1}^{r-1} w(v_1^i, v_2^i) \leq w(\cup_{i=1}^{r-1} C_i \cap M_w)/2 \leq w(M_w)/2$. Thus the total loss in weight is bounded by $w(M_w)/4$.

In conclusion the partial tour has weight at least $3w(M_w)/4$ and length at least $3\ell(M_\ell)/4$ by inequality (1). □

Lemma 3. *Assuming that w and ℓ satisfy the triangle inequality, we can build in polynomial time a partial tour on K_n with weight at least $\frac{5}{6}w(M_w)$ and length at least $(\frac{5}{6} - \varepsilon(n))\ell(M_\ell)$. Here $\varepsilon(n) = 2/(n-1)$ and then tends to 0 when n tends to ∞.*

Proof. As it is done in Lemma 2, we transform case 1 into case 2. Thus, suppose that we are in case 1 that is each component C_i is a cycle and w.l.o.g. that the edge of M_ℓ with minimum length is e. Remove this edge e to create a path with endpoints denoted by x and y. When n is even (resp. odd) this deletion induces a loss of at most $2\ell(M_\ell)/n = \varepsilon(n)\ell(M_\ell)$ (resp. $2\ell(M_\ell)/(n-1) = \varepsilon(n)\ell(M_\ell)$). Note that $\varepsilon(n)$ tends to 0 when n tends to ∞.

Suppose now that we are in the case 2. As it is done in Lemma 2 we can assume that there is an integer $r \in \{1, \ldots, p\}$ such that C_i for $i \geq r$ is not a cycle whereas C_i for $1 \leq i < r$ is a cycle. We are going to patch the cycles to C_r, one by one. We explain how to patch C_1, and the procedure is repeated for the cycles $C_2, \cdots, C_{r-1}$. Let x and y be the two extremities of C_r.

If $|C_1 \cap M_w| \geq 3$ then delete an edge of minimum weight and call it (v_1^1, v_2^1). We get that $w(v_1^1, v_2^1) \leq \frac{1}{3}w(C_1 \cap M_w)$. Add the edge with maximum weight between (v_1^1, x) and (v_2^1, x). By the triangle inequality, $\max\{w(v_1^1, x), w(v_2^1, x)\} \geq w(v_1^1, v_2^1)/2$. If $w(v_1^1, x) \geq w(v_2^1, x)$ then $x := v_2^1$, otherwise $x := v_1^1$. Disregarding the weight of the edges in $C_1 \cap M_\ell$, the modification causes a loss in weight of at most $w(v_1^1, v_2^1) - w(v_1^1, v_2^1)/2 = w(v_1^1, v_2^1)/2 \leq \frac{1}{6}w(C_1 \cap M_w)$. Since no edge from M_ℓ was removed, and disregarding the length of the edges in $C_1 \cap M_w$, the modification does not cause any loss in length. Hence the patching guarantees that the new path P satisfies $w(P) \geq w(C_r) + 5w(C_1 \cap M_w)/6$ and $\ell(P) \geq \ell(C_r) + \ell(C_1 \cap M_\ell)$.

Now suppose that C_1 is a cycle on 4 nodes and contains four edges (a, b), (b, c), (c, d), (d, a) such that $C_1 \cap M_w = \{(a, b), (c, d)\}$ and $C_1 \cap M_\ell = \{(b, c), (a, d)\}$. Using the triangle inequality we get that

$$w(a, c) + w(b, d) + w(C_1 \cap M_\ell) \geq w(C_1 \cap M_w) \quad (2)$$

$$\ell(a, c) + \ell(b, d) + \ell(C_1 \cap M_w) \geq \ell(C_1 \cap M_\ell) \quad (3)$$

– Suppose that $\ell(C_1 \cap M_w) \geq \ell(C_1 \cap M_\ell)/8$. W.l.o.g., assume $\ell(a, d) \geq \ell(b, c)$. Remove (b, c) and add the edge with maximum length between (b, x) and (x, c). Since $\max\{\ell(b, x), \ell(x, c)\} \geq \ell(b, c)/2$ by the triangle inequality, we get that the new path P satisfies $\ell(P) \geq \ell(C_r) + \ell(C_1 \cap M_w) + \ell(a, d) + \ell(b, c)/2 \geq \ell(C_r) + \ell(C_1 \cap M_\ell)/8 + \ell(C_1 \cap M_\ell)/2 + \ell(a, d)/2 \geq \ell(C_r) + \ell(C_1 \cap M_\ell)/8 + \ell(C_1 \cap M_\ell)/2 + \ell(C_1 \cap M_\ell)/4 = \ell(C_r) + 7\ell(C_1 \cap M_\ell)/8$.

- Suppose that $w(C_1 \cap M_\ell) \geq w(C_1 \cap M_w)/8$. W.l.o.g., assume $w(a,b) \geq w(c,d)$. Remove (c,d) and add the edge with maximum length between (c,x) and (x,d). Since $\max\{w(c,x), w(x,d)\} \geq w(c,d)/2$ by the triangle inequality, we get as in the previous case that $w(P) \geq w(C_r) + w(C_1 \cap M_\ell) + w(a,b) + w(c,d)/2 \geq w(C_r) + 7w(C_1 \cap M_w)/8$.
- Now suppose that $\ell(C_1 \cap M_w) < \ell(C_1 \cap M_\ell)/8$ and $w(C_1 \cap M_\ell) < w(C_1 \cap M_w)/8$. Using Inequalities (2) and (3) we get that $w(a,c)+w(b,d) > 7w(C_1 \cap M_w)/8$ and $\ell(a,c) + \ell(b,d) > 7\ell(C_1 \cap M_\ell)/8$. In this case the new path P obtained by adding any two edges to $(a,c), (b,d)$ and C_r satisfies $w(P) \geq w(C_r) + 7w(C_1 \cap M_w)/8$ and $\ell(P) \geq \ell(C_r) + 7\ell(C_1 \cap M_\ell)/8$.

In conclusion, when C_1 contains four nodes, we can always patch it to C_r so that the loss in weight (resp. length) is at most $w(C_1 \cap M_w)/8$ (resp. $\ell(C_1 \cap M_\ell)/8$).

We have seen that this loss is of (at most) 1/6 on both objective functions when C_1 contains at least six nodes. We deduce that after the patching of all cycles C_i for $i < r$, the current solution is a path P and its weight (resp. length) is at least $w(C_r) + \frac{5}{6}w\big(\bigcup_{i=1}^{r-1} C_i \cap M_w\big)$ (resp. $\ell(C_r) + \frac{5}{6}\ell\big(\bigcup_{i=1}^{r-1} C_i \cap M_\ell\big)$). Adding $\cup_{i=r+1}^{p} C_i$ to P, we get a partial tour P'. Using $w(C_r) \geq w(C_r \cap M_w)$ and $\ell(C_r) \geq \ell(C_r \cap M_\ell) - \varepsilon(n)\ell(M_\ell)$ we get that the solution P' has weight (resp. length) at least $\frac{5}{6}w(M_w)$ (resp. $(\frac{5}{6} - \varepsilon(n))\ell(M_\ell)$). □

Theorem 1. *We can build in polynomial time a single tour on K_n which constitutes a $(\rho - \xi(n))$-approximate Pareto set for the biobjective Max TSP where $\rho = 5/12$ when w and ℓ satisfy the triangle inequality, $\rho = 3/8$ when only w satisfies the triangle inequality and $\rho = 1/4$ when neither w nor ℓ satisfies the triangle inequality. Here $\xi(n) = \Theta(1/n)$ and then tends to 0 when n tends to ∞.*

Proof. Consider first the case when x and ℓ satisfy the triangle inequality. Lemma 3 states that we can build a partial tour with weight (resp. length) at least $5w(M_w)/6$ (resp. $(\frac{5}{6} - \varepsilon(n))\ell(M_\ell)$) where $\varepsilon(n) = \frac{2}{n-1}$. If the partial tour is not a tour then connect its components to create a tour. Using the fact that every edge weight (resp. length) is nonnegative, the weight (resp. length) cannot decrease. Denote by opt_w (resp. opt_ℓ) the optimal weight (resp. length) of a tour. It is well known that $w(M_w) \geq (\frac{1}{2} - \varepsilon'(n))opt_w$ and $\ell(M_\ell) \geq (\frac{1}{2} - \varepsilon'(n))opt_\ell$ where $\varepsilon'(n) = 0$ when n is even, otherwise $\varepsilon'(n) = \frac{1}{2n}$. Let $\xi(n) = \frac{\varepsilon(n)}{2} + \frac{5\varepsilon'(n)}{6} - \varepsilon'(n)\varepsilon(n)$. We get that the tour constructed has weight at least $\frac{5}{6}w(M_w) \geq \frac{5}{6}(\frac{1}{2} - \varepsilon'(n))opt_w > (\frac{5}{12} - \xi(n))opt_w$. The length is at least $(\frac{5}{6} - \varepsilon(n))\ell(M_\ell) \geq (\frac{5}{6} - \varepsilon(n))(\frac{1}{2} - \varepsilon'(n))opt_\ell = (\frac{5}{12} - \xi(n))opt_\ell$. Use Lemmas 2 and 3 and similar arguments for the other cases. □

5 An Improved Analysis

In this section, we refine the analysis of our approximation algorithm when the triangle inequality is not assumed on any objective function. We show that the tour returned by our algorithm is an asymptotic $\frac{1+2\sqrt{2}}{14} \approx 0.273$ approximation

of the ideal point. Recall that some instances of the problem do not admit any $(\frac{1}{3}+\epsilon)$-approximate solution, for all $\epsilon > 0$ [14].

The intuition behind the improved analysis is the following. The ratio 1/4 of Theorem 1 follows from two observations: the tour returned by the approximation algorithm is a 1/2-approximation of the maximum weight/length matching, and this latter is an asymptotic 1/2-approximation of the maximum weight/length tour. Taken separately both observations are tight but we exploit the fact that they cannot occur simultaneously.

Theorem 2. *We can build in polynomial time a* $(\frac{1+2\sqrt{2}}{14} - \xi(n))$*-approximate Pareto set containing a single tour on* K_n *for Biobjective Max TSP. Here* $\xi(n) = \Theta(1/n)$ *and then, tends to* 0 *when* n *tends to* ∞.

Proof. (Sketch) Define δ as $\frac{4\sqrt{2}-5}{14} \approx 0.0469$. Actually, δ is the positive root of equation $-1 + 20x + 28x^2 = 0$. We can show that every instance K_n of the problem satisfies one of the following statements:

- (i) a partial tour P' on K_n with weight at least $(\frac{1}{2}+\delta)w(M_w)$ and, at the same time, length at least $(\frac{1}{2}+\delta)\ell(M_\ell)$ exists and can be computed in polynomial time.
- (ii) every Hamiltonian cycle has weight at most $(\frac{3}{2}+7\delta)w(M_w)$ and, at the same time, its length is at most $(\frac{3}{2}+7\delta)\ell(M_\ell)$.

Recall that $w(M_w) \geq (\frac{1}{2} - \varepsilon'(n))opt_w$, $\ell(M_\ell) \geq (\frac{1}{2} - \varepsilon'(n))opt_\ell$ where $\varepsilon'(n) = 0$ when n is even, otherwise $\varepsilon'(n) = 1/2n$. If K_n satisfies (i), then by hypothesis the partial tour P' has weight (resp. length) at least $(1/4 + \delta/2 - \xi(n))opt_w$ (resp. $(1/4 + \delta/2 - \xi(n))opt_\ell$) with $\xi(n) = \varepsilon'(n)(1/2 + \delta)$. If K_n satisfies (ii), then starting from $M_w \cup M_\ell$ as it is done in previous section and using Lemma 1, a partial solution P with weight (resp. length) at least $w(M_w)/2$ (resp. $\ell(M_\ell)/2$) can be built in polynomial time. Now, since by hypothesis $opt_w \leq (\frac{3}{2}+7\delta)w(M_w)$, and $opt_\ell \leq (\frac{3}{2}+7\delta)\ell(M_\ell)$, the partial solution P has a weight (resp. length) at least $\frac{1}{2}\frac{opt_w}{(\frac{3}{2}+7\delta)}$ (resp. $\frac{1}{2}\frac{opt_\ell}{(\frac{3}{2}+7\delta)}$).

Finally remark that on the one hand, a tour can be obtained by connecting the components of a partial tour without decreasing the weight/length since every edge weight/length is nonnegative and on the other hand, $\frac{1}{2}\frac{1}{(\frac{3}{2}+7\delta)} = 1/4+\delta/2 = \frac{1+2\sqrt{2}}{14}$ because δ is the positive root of equation $-1 + 20x + 28x^2 = 0$.

We assume $n \geq 5$, since otherwise the partial solution P given in Lemma 1 has weight (resp. length) at least $opt_w/2$ (resp. $opt_\ell)/2$).

We consider three distinct cases which can be distinguished in polynomial time.

Case 1. Let us suppose that there exists a cycle, say C_1 w.l.o.g., such that the edge with minimum weight in $C_1 \cap M_w$ has weight at least $(\frac{1}{2} - \delta)w(M_w)$ and, at the same time, the edge with minimum length in $C_1 \cap M_\ell$ has length at least $(\frac{1}{2} - \delta)\ell(M_\ell)$. Since $1/2 - \delta > 1/3$, C_1 must be a cycle on four nodes, i.e. $C_1 \cap M_w = \{(a, b), (c, d)\}$ and $C_1 \cap M_\ell = \{(b, c), (a, d)\}$ (see Figure 2 for an illustration).

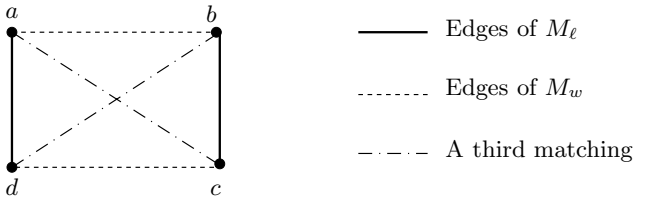


Fig. 2. The cycle C_1. Bold edges belong to M_ℓ and dashed edges belong to M_w; the remaining edges form a third matching $M_r = \{(a,c),(b,d)\}$.

We conduct a subcase analysis depending on the weight or the length of the edges having at least one endpoint in $V(C_1)$: case (1.1.w) $\max\{w(e) : e \in C_1 \cap M_\ell\} > 2\delta w(M_w)$, case (1.1.$\ell$) $\max\{\ell(e) : e \in C_1 \cap M_w\} > 2\delta\ell(M_\ell)$, case (1.2.$w$) $\max\{w(a,c), w(b,d)\} > (\frac{1}{2}+\delta)w(M_w)$, case (1.2.$\ell$) $\max\{\ell(a,c), \ell(b,d)\} > (\frac{1}{2}+\delta)\ell(M_\ell)$, case (1.3.$w$) $\max\{w(i,j) : i \in V(C_1), j \notin V(C_1)\} > 2\delta w(M_w)$, case (1.3.$\ell$) $\max\{\ell(i,j) : i \in V(C_1), j \notin V(C_1)\} > 2\delta\ell(M_\ell)$ and case (1.4) $\max\{w(e) : e \in C_1 \cap M_\ell\} \leq 2\delta w(M_w)$, $\max\{\ell(e) : e \in C_1 \cap M_w\} \leq 2\delta\ell(M_\ell)$, $\max\{w(a,c), w(b,d)\} \leq (\frac{1}{2}+\delta)w(M_w)$, $\max\{\ell(a,c), \ell(b,d)\} \leq (\frac{1}{2}+\delta)\ell(M_\ell)$, $\max\{w(i,j) : i \in V(C_1), j \notin V(C_1)\} \leq 2\delta w(M_w)$ and $\max\{\ell(i,j) : i \in V(C_1), j \notin V(C_1)\} \leq 2\delta\ell(M_\ell)$.

One can prove that in case (1.4) the instance K_n satisfies (ii) whereas in other cases the instance K_n satisfies (i). Due to space limitation, we only give the details of the first four cases.

(1.1.w) If $w(a,d) > 2\delta w(M_w)$ or $w(b,c) > 2\delta w(M_w)$ then remove (c,d). We get that $w(a,b) + w(b,c) + w(a,d) > (\frac{1}{2}+\delta)w(M_w)$ and $\ell(a,b) + \ell(b,c) + \ell(a,d) \geq (1-2\delta)\ell(M_\ell) \geq (1/2+\delta)\ell(M_\ell)$.

(1.1.ℓ) If $\ell(a,b) > 2\delta\ell(M_\ell)$ or $\ell(c,d) > 2\delta\ell(M_\ell)$ then remove (b,c). We get that $\ell(a,d) + \ell(a,b) + \ell(c,d) > (\frac{1}{2}+\delta)\ell(M_\ell)$ and $w(a,d) + w(a,b) + w(c,d) \geq (1-2\delta)w(M_w) \geq (1/2+\delta)w(M_w)$.

(1.2.w) If $\max\{w(a,c), w(b,d)\} > (\frac{1}{2}+\delta)w(M_w)$ then remove $\{(a,b),(c,d)\}$ and add the edge with maximum weight between (a,c) and (b,d), say (a,c) without loss of generality. We get that $w(a,c) + w(b,c) + w(a,d) > (\frac{1}{2}+\delta)w(M_w)$ and $\ell(a,d) + \ell(a,c) + \ell(b,c) \geq (1-2\delta)\ell(M_\ell) \geq (1/2+\delta)\ell(M_\ell)$.

(1.2.ℓ) If $\max\{\ell(a,c), \ell(b,d)\} > (\frac{1}{2}+\delta)\ell(M_\ell)$ then remove $\{(a,d),(b,c)\}$ and add the edge with maximum length between (a,c) and (b,d), say (a,c) without loss of generality. We get that $w(a,c) + w(a,b) + w(c,d) > (1-2\delta)w(M_w) > (1/2+\delta)w(M_w)$ and $\ell(a,c) + \ell(a,b) + \ell(c,d) \geq (\frac{1}{2}+\delta)\ell(M_\ell)$.

Case 2. Suppose that there exists a cycle, say C_1 w.l.o.g., such that the edge with minimum weight in $C_1 \cap M_w$ has weight *at most* $(\frac{1}{2}-\delta)w(M_w)$ and, at the same time, the edge with minimum length in $C_1 \cap M_\ell$ has length *at least* $(\frac{1}{2}-\delta)\ell(M_\ell)$. We will prove that the instance K_n satisfies (i). Again, since $1/2 - \delta > 1/3$, C_1 must be a cycle on four nodes. Again we suppose that $C_1 \cap M_w = \{(a,b),(c,d)\}$ and $C_1 \cap M_\ell = \{(b,c),(a,d)\}$.

Remove the edge in $C_1 \cap M_w$ with minimum weight and for any other cycle C_i remove one edge in $C_i \cap M_\ell$ arbitrarily. We get a partial tour. Since $w(M_w) - \min\{w(a,b), w(c,d)\} \geq (\frac{1}{2} + \delta)w(M_w)$ and $\ell(C_1 \cap M_\ell) = \ell(a,d) + \ell(b,c) \geq 2(\frac{1}{2} - \delta)\ell(M_\ell) \geq (\frac{1}{2} + \delta)\ell(M_\ell)$, the partial tour has weight (resp. length) at least $(\frac{1}{2} + \delta)w(M_w)$ (resp. $(\frac{1}{2} + \delta)\ell(M_\ell)$).

The case where there exists a cycle C_1 such that the edge with minimum weight in $C_1 \cap M_w$ has weight *at least* $(\frac{1}{2} - \delta)w(M_w)$ and, at the same time, the edge with minimum length in $C_1 \cap M_\ell$ has length *at most* $(\frac{1}{2} - \delta)\ell(M_\ell)$ is dealt with similar arguments by flipping w and ℓ.

Case 3. Denote by e_i^w (resp. e_i^ℓ) the edge in $C_i \cap M_w$ (resp. $C_i \cap M_\ell$) with minimum weight (resp. length). We deal with the remaining case where $w(e_i^w) \leq (\frac{1}{2} - \delta)w(M_w)$ and $\ell(e_i^\ell) \leq (\frac{1}{2} - \delta)\ell(M_\ell)$ for all $i \in \{1, \ldots, p\}$. We will prove that the instance K_n satisfies (i). Since every cycle contains at least two edges of M_w and also two edges of M_ℓ we deduce that

$$\sum_{i=1}^{p} w(e_i^w) \leq w(M_w)/2 \text{ and } \sum_{i=1}^{p} \ell(e_i^\ell) \leq \ell(M_\ell)/2 \tag{4}$$

- Suppose there is an index i^* such that $w(e_{i^*}^w) \geq \delta w(M_w)$. Then for every cycle C_i except C_{i^*} remove e_i^w. Remove $e_{i^*}^\ell$. Using the first part of inequality (4) we get a partial tour with weight at least $w(M_w) - \sum_{i=1}^{p} w(e_i^w) + w(e_{i^*}^w) \geq (1/2 + \delta)w(M_w)$ and length at least $\ell(M_\ell) - \ell(e_{i^*}^\ell) \geq (1/2 + \delta)\ell(M_\ell)$.
- Suppose there is an index i^* such that $\ell(e_{i^*}^\ell) \geq \delta\ell(M_\ell)$. With similar arguments we can build a partial tour with weight at least $(1/2 + \delta)w(M_w)$ and length at least $(1/2 + \delta)\ell(M_\ell)$.
- Suppose that $w(e_i^w) < \delta w(M_w)$ and $\ell(e_i^\ell) < \delta\ell(M_\ell)$ for all i. If $\sum_{i=1}^{p} w(e_i^w) \leq (\frac{1}{2} - \delta)w(M_w)$, then by removing e_i^w for $i = 1, \ldots, p$ we get a partial tour P with weight at least $(1/2 + \delta)w(M_w)$ and length at least $\ell(M_\ell)$. Otherwise, there exists an index $i^* < p$ such that

$$\sum_{i=1}^{i^*} w(e_i^w) \leq (\frac{1}{2} - \delta)w(M_w) \text{ and } \sum_{i=1}^{i^*+1} w(e_i^w) > (\frac{1}{2} - \delta)w(M_w) \tag{5}$$

Using inequalities (4), (5) and $w(e_{i^*+1}^w) < \delta w(M_w)$ we get that

$$\begin{aligned} \sum_{i=1}^{i^*+1} w(e_i^w) + \sum_{i=i^*+2}^{p} w(e_i^w) &\leq w(M_w)/2 \\ \sum_{i=i^*+2}^{p} w(e_i^w) &< \delta w(M_w) \\ \sum_{i=i^*+1}^{p} w(e_i^w) &< 2\delta w(M_w) \leq (\frac{1}{2} - \delta)w(M_w) \end{aligned} \tag{6}$$

Now remark that

$$\min\{\sum_{i=1}^{i^*} \ell(e_i^\ell), \sum_{i=i^*+1}^{p} \ell(e_i^\ell)\} \leq \frac{1}{2}\sum_{i=1}^{p} \ell(e_i^\ell) \leq \frac{1}{4}\ell(M_\ell) \quad (7)$$

where the right part of inequality (4) is used. If $\sum_{i=1}^{i^*} \ell(e_i^\ell) \leq \sum_{i=i^*+1}^{p} \ell(e_i^\ell)$ then remove e_i^ℓ for $i = 1, \ldots, i^*$ and remove e_i^w for $i = i^*+1, \ldots, p$. We get a partial tour with weight at least $(1/2+\delta)w(M_w)$ by inequality (6) and length at least $3\ell(M_\ell)/4 \geq (1/2+\delta)\ell(M_\ell)$ by inequality (7). If $\sum_{i=1}^{i^*} \ell(e_i^\ell) > \sum_{i=i^*+1}^{p} \ell(e_i^\ell)$ then remove e_i^w for $i = 1, \ldots, i^*$ and remove e_i^ℓ for $i = i^*+1, \ldots, p$. We get a partial tour with weight at least $(1/2+\delta)w(M_w)$ by inequality (5) and length at least $3\ell(M_\ell)/4 \geq (1/2+\delta)\ell(M_\ell)$ by inequality (7). □

6 Future Work

We considered the biobjective Max TSP. It would be interesting to study the cases where there is a fixed number $k \geq 3$ of objectives. There are still gaps between positive and negative results given in this article. For example, when both objective functions are metric, we provide a polynomial time $(\frac{5}{12} - \epsilon)$-approximation and an upper bound of $\frac{3}{4}$. Maybe both results can be improved. An interesting future work would be to investigate randomized algorithms. Another direct extension of our work is to consider the multiobjective asymmetric Max TSP.

References

1. Angel, E., Bampis, E., Fishkin, A.V.: A note on scheduling to meet two min-sum objectives. Operations Research Letters 35(1), 69–73 (2007)
2. Angel, E., Bampis, E., Gourvès, L.: Approximating the Pareto curve with local search for the bicriteria TSP(1,2) problem. Theoretical Computer Science 310(1-3), 135–146 (2004)
3. Angel, E., Bampis, E., Gourvès, L.: Approximation algorithms for the bi-criteria weighted max-cut problem. Discrete Applied Mathematics 154(12), 1685–1692 (2006)
4. Angel, E., Bampis, E., Gourvès, L., Monnot, J.: (Non)-Approximability for the Multi-criteria *TSP*(1,2). In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 329–340. Springer, Heidelberg (2005)
5. Angel, E., Bampis, E., Kononov, A.: On the approximate tradeoff for bicriteria batching and parallel machine scheduling problems. Theoretical Computer Science 306(1-3), 319–338 (2003)
6. Berger, A., Bonifaci, V., Grandoni, F., Schäfer, G.: Budgeted matching and budgeted matroid intersection via the gasoline puzzle. Mathematical Programming 128(1-2), 355–372 (2011)
7. Bläser, M., Manthey, B., Putz, O.: Approximating Multi-criteria Max-TSP. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 185–197. Springer, Heidelberg (2008)

8. Chen, Z.-Z., Okamoto, Y., Wang, L.: Improved deterministic approximation algorithms for max TSP. Information Processing Letters 95(2), 333–342 (2005)
9. Ehrgott, M.: Multicriteria Optimization. LNEMS. Springer, Heidelberg (2005)
10. Erlebach, T., Kellerer, H., Pferschy, U.: Approximating multiobjective knapsack problems. Management Science 48(12), 1603–1612 (2002)
11. Hong, S.-P., Chung, S.-J., Park, B.H.: A fully polynomial bicriteria approximation scheme for the constrained spanning tree problem. Operations Research Letters 32(3), 233–239 (2004)
12. Kowalik, L., Mucha, M.: Deterministic 7/8-approximation for the metric maximum TSP. Theoretical Computer Science 410(47-49), 5000–5009 (2009)
13. Manthey, B.: Multi-Criteria TSP: Min and Max Combined. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 205–216. Springer, Heidelberg (2010)
14. Manthey, B.: On approximating multi-criteria TSP. In: Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009), pp. 637–648 (2009)
15. Manthey, B.: Deterministic Algorithms for Multi-criteria TSP. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 264–275. Springer, Heidelberg (2011)
16. Manthey, B., Shankar Ram, L.: Approximation algorithms for multi-criteria Traveling Salesman Problems. Algorithmica 53(1), 69–88 (2009)
17. Paluch, K., Mucha, M., Mądry, A.: A 7/9 - Approximation Algorithm for the Maximum Traveling Salesman Problem. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 298–311. Springer, Heidelberg (2009)
18. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000), pp. 86–92 (2000)
19. Rasala, A., Stein, C., Torng, E., Uthaisombut, P.: Existence theorems, lower bounds and algorithms for scheduling to meet two objectives. In: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), pp. 723–731 (2002)
20. Ravi, R., Goemans, M.X.: The Constrained Minimum Spanning Tree Problem. In: Karlsson, R., Lingas, A. (eds.) SWAT 1996. LNCS, vol. 1097, pp. 66–75. Springer, Heidelberg (1996)
21. Ravi, R., Marathe, M.V., Ravi, S.S., Rosenkrantz, D.J., Hunt III, H.B.: Many birds with one stone: multi-objective approximation algorithms. In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC 1993), pp. 438–447 (1993)
22. Stein, C., Wein, J.: On the existence of schedules that are near-optimal for both makespan and total weighted completion time. Operations Research Letters 21(3), 115–122 (1997)

Parameterized Approximation Algorithms for Hitting Set

Ljiljana Brankovic[1] and Henning Fernau[2]

[1] School of Electrical Engineering and Computer Science
The University of Newcastle, Callaghan, NSW 2308, Australia
Ljiljana.Brankovic@newcastle.edu.au
[2] Fachbereich 4, Abteilung Informatik
Universität Trier, 54286 Trier, Germany
fernau@uni-trier.de

Abstract. We are going to analyze simple search tree algorithms for approximating d-Hitting Set, focussing on the case of factor-2 approximations for $d = 3$. We also derive several results for hypergraph instances of bounded degree, including a new polynomial-time approximation.

1 Introduction

Our approach—in general. There is now a growing body of literature concerned with combining two very natural ideas to cope with intractability: that of approximation and that of parameterized algorithms; see [4] for background information. We are putting here a recent approach of ours [3] to another test: namely, the idea of combining search tree algorithms with approximation to obtain better approximation ratios at the cost of (moderately) exponential time.

Problem statement. The d-Hitting Set (d-HS) can be viewed as a "vertex cover problem" on hypergraphs, formally stated as follows:

Given: A hypergraph $G = (V, E)$ with *edge size* bounded by d: $\forall e \in E(|e| \leq d)$
Parameter: a non-negative integer K
Question: Is there a *hitting set* C of cardinality of at most K, i.e.,
$\exists C \subseteq V \forall e \in E(C \cap e \neq \emptyset)$ and $|C| \leq K$?

We will mostly consider the case $d = 3$ in what follows.

In this paper, we follow the definition proposed in [4], which in our context can be phrased as follows. Given a hypergraph G and a parameter K such that a minimum hitting set C^* of G satisfies $|C^*| \leq K$, a *parameterized approximation algorithm with (constant) approximation ratio* ρ for minimum hitting set produces a hitting set C such that $|C| \leq \rho|C^*|$. If $K < |C^*|$, the algorithm will answer NO. Such an algorithm runs in time $\mathcal{O}^*(f(K))$ for some function f, where $\mathcal{O}^*$ notation suppresses polynomial factors.

Bibliographical notes. Despite all efforts, the best known constant factor polynomial time approximation algorithm for general (unweighted) hypergraphs with

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 63–76, 2012.

edge size d is still a factor-d approximation for fixed d, and this is even optimal under the unique games conjecture [10]. So, even the factor-2 approximation we focus on in this paper is is a considerable progress. Several exact parameterized algorithms have been developed for our problem. Let us only mention the best published results here: The best publically available algorithm for 3-HS is Wahlström's, as it appeared in his PhD Thesis [12], having a running time of $\mathcal{O}^*(2.076^K)$. The best published one has only a running time of $\mathcal{O}^*(2.179^K)$; see [7].

Why Hitting Set*?* (1) Hitting Set problems show up in many places; e.g., Reiter's ground-breaking research on *model-based diagnosis* [11] relates the automatic diagnosis of systems to Hitting Set, or HS for short. (2) vertex cover, or VC for short, is the paradigmatic test-bed problem for parameterized algorithms. As HS can be seen as a vertex cover problem on hypergraphs, it is quite a natural question to see how the ideas developed in [3] might generalize to the case of hypergraphs. It should be noted that the paper [2] can be seen as a sort of precursor of [3], although the techniques are quite different.

Why using exponential time for approximation? As we have seen when working on our VC-approximation paper, this kind of work often also gives new insights and ideas for polynomial time approximations, for instance, new reduction rules. On the other hand, we believe that this is also interesting for "classical FPT", keeping in mind that, at least with search tree algorithms, what slows them down is finding (proving) a NO answer: In that case, often the whole search tree has to be traversed. Approximation algorithms, be them polynomial time or FPT, can serve to find a quicker NO. Since we are talking exponential time anyways, a "very fast" exponential time algorithm may be worthwhile running first (or in parallel) to find this quick NO answer, even though we might be mainly interested in finding exact answers. As it is considered unlikely to find a better polynomial time approximation algorithm than the (trivial) one offering a factor of three. Whoever is interested in finding better approximation guarantees must therefore use exponential time.

The results of this paper. We show a branching algorithm that enables to approximate 3-HS within a factor of two, running in time $\mathcal{O}^*(1.29^K)$; see Sec. 3 and 4. For subcubic instances, the analysis can be improved to show an upper bound of $\mathcal{O}^*(1.26^K)$; see Sec. 5. These figures compare favorably with the ones that can be obtained by employing the best exact algorithms for approximation purposes, as explained in Sec. 2. We also give several results for approximating d-HS for general d, as well as a new polynomial-time algorithm for approximating 3-HS for instances of degree bounded by three up to a factor of $\frac{5}{2}$.

General notions and definitions. We introduce some terminology on hypergraphs as needed for Hitting Set. A *hypergraph* $G = (V, E)$ is given by its finite set of *vertices* V and its set of *(hyper)-edges* E, where a hyperedge is a subset of V. The cardinality $|e|$ of a hyperedge e is also called its *size*. The cardinality of the set of edges which contain the vertex v is called the *degree* of v, written $\deg(v)$.

2 A Simple Design for Parameterized Approximation

Most of the currently best algorithms for 3-HS are all based on a search tree algorithm, combined with the use of reduction rules. To each node n of the search tree, a set of vertices C_n of the input hypergraph $G = (V, E)$ can be associated that collects a partial hitting set, i.e., in the subtree T_n rooted at n, we are only interested in hitting set solutions S that contain C_n. C_n has been constructed on the path from the root of the search tree down to n by invoking the simple operation "`Put` x `into the solution.`" $|C_n|$ times. We can also associate a "current hypergraph" $G_n = (V_n, E_n)$ to n that can be easily obtained from G and C_n.

The technique for obtaining an approximative solution from such an algorithm is very simple through *interleaving* a step that deliberately worsens a solution in order to speed up the branching. So, we associate a hyperedge set $H_n \subseteq E$ to n that has been formed as follows (to obtain a factor-2 approximation for the ease of presentation): Whenever some vertex x is put into the solution, we pick some $e \in E_n$ and put it into H_n. The approximative solution associated to n is now $S_n := C_n \cup V(H_n)$. Of course, the "current hypergraph" $G'_n = (V_n, E_n)$ associated to n now depends on G and (C_n, H_n).

This procedure guarantees the following properties:

- If $G = (V, E)$ has a hitting set S of size K, then there is a path in the search tree of the approximative branching algorithm of length at most $K/2$ such that C_l and H_l are associated to the leaf l of that path, with $C_l \subseteq S \subseteq S_l$.
- Any valid hitting set solution C with $C_l \subseteq C \subseteq S_l$ contains at least one vertex from each edge from H_l, i.e., $|C| \geq 2|C_l|$ by construction, since $|H_l| = |C_l|$.
- Hence, $2|C_l| \leq |S|$ and $4|C_l| = |S_l|$, and S_l is an approximative solution that is at most twice as big as the optimum (constrained to sets embracing C_l).

This allows us to conclude, using Fernau's result [7]:

Proposition 1. *There is a factor-2 approximation of* 3-HS *in time* $\mathcal{O}^*(1.477^K)$.

There is a technical difficulty when trying to apply Wahlström's result [12], namely the Measure & Conquer analysis that he employs. It is not completely clear if we can always choose an edge to worsen the approximation factor whose removal reduces the measure by the same amount as the previous branching did. As we will present considerably better running times in this paper, this question is of no major concern.

The same approach can be used for other variants of d-HS, as summarized in the following table. There, the first line lists the running time estimates that we obtained in [6] for the exact solution, ρ_d shows the intended approximation factor, where $\rho_d = (d+1)/2$, since each vertex that we put into the hitting set is accompanied by one worsening step, and $T_d^{\rho_d}$ shows the obtained running times; clearly, for the chosen approximation factors, the basis of $T_d^{\rho_d}$ is just the square root of the basis for T_d.

By letting grow H_n more slowly or more rapidly along the search tree, it is straightforward to obtain similar results for other (better or worse) approximation factors with accordingly changed (slower or faster) running times. We leave

Table 1. Approximation factors and running times obtained based on [7,6]

d	3	4	5	6	10	100
$T_d(K) \leq$	2.18^K	3.12^K	4.08^K	5.05^K	9.02^K	99.0002^K
$\rho_d =$	2	2.5	3	3.5	5.5	55.5
$T_d^{\rho_d}(K) \leq$	1.51^K	1.77^K	2.02^K	2.25^K	3.01^K	9.95^K

out the according details in this extended abstract, but rather refer to similar reasonings for VC in [3].[1]

In the following, we will present branching algorithms that are simpler and simpler to analyze than those from [7,6,12], but where a more elaborated use of interleaving nonetheless leads to better running times. For the ease of presentation, we will no longer differentiate between C_n and $V(H_n)$ in what follows, but just assume that a partial hitting set C_n is associated to each node n of the search tree. Whenever clear from the context, we omit the index n. Similarly abusing notation, we also allow $G = (V, E)$ to refer to the "current hypergraph."

It is worth noticing that the approach for approximating VC of [2] (as detailed in the PhD Thesis of N. Bourgeois) that consisted in first splitting the given instance into parts and then computing exact solutions for each of the parts is dependent on some kernelization results that are not available in this case.

3 A Simple Branching for Approximation

We are now elaborating on the interleaving idea further, looking at a very simplistic-looking general branching algorithm. Due to the fact that we observe a good approximation ratio when a certain vertex is not put into the partial hitting set, even this simple branching already improves on the idea presented in the previous section. In the beginning, Algorithm 1 is called with the parameters $(G, K, \emptyset)$, where (G, K) is the original 3-HS instance that we want to solve. This original parameter K is also occasionally used by Algorithm 1, while k refers to the value of the parameter in the current situation.

The algorithm uses the following ingredients:

- Several reduction rules are known for HS; we will list (some of) them below, including possibly rules that are only valid in an approximative sense.
- We still have to specify heuristic priorities that might improve our branching.
- Whenever a vertex is put into the hitting set, an edge is selected for worsening the solution. Also here, we might introduce some selection strategies to improve on the running time.
- It is well-known that (3)HS can be solved to optimality in polynomial time if each vertex belongs to at most two edges by invoking some Edge Cover algorithm on an auxiliary graph.

[1] Similar ideas have been presented at WorKer 2011 by H. Shachnai, joint work with M. Fellows and F. Rosamond.

Algorithm 1. `3HS-2-appr-general`: A 2-approximation algorithm for 3-HS

1: Input: Hypergraph $G = (V, E)$, parameter k, and a partial hitting set C
2: Output: Either NO or a hitting set C with $|C| \leq 2K$.
3: Apply all reduction rules exhaustively, possibly modifying C and k.
4: **if** $k < 0$ **then**
5: Return NO.
6: **else if** possible: choose a $v \in V$ such that $\deg(v) \geq 3$ according to heuristic priorities. **then**
7: Binary branch on v:
8: Case 1: Put $v \in C$, i.e., $C \leftarrow C \cup \{v\}$, $V \leftarrow V \setminus \{v\}$, $E \leftarrow \{e \in E \mid e \subseteq V\}$.
Select some $f \in E$ for worsening and put all vertices of f into C, i.e.,
$C \leftarrow C \cup f$, $V \leftarrow V \setminus f$, $E \leftarrow \{e \in E \mid e \subseteq V\}$, $k \leftarrow k - 2$
Recursively call `3HS-2-appr-general` with the modified parameters.
9: Case 2: Do not put v into C, i.e.,
$V \leftarrow V \setminus \{v\}$, $E \leftarrow \{e \setminus \{v\} \mid e \in E\}$.
Recursively call `3HS-2-appr-general` with the modified parameters.
10: **else**
11: Solve the remaining instance optimally using an Edge Cover algorithm.
12: Return either NO or a hitting set C with $|C| \leq 2K$.

Simple reduction rules. We first list the (well-known) reduction rules valid for (3)HS, as they can be found, e.g., in [7].

- (hyper)edge domination: A hyperedge e is *dominated* by another hyperedge f if $f \subset e$. In that case, delete e.
- tiny edges: Delete all hyperedges of size one and place the corresponding vertices into the hitting set.
- vertex domination: A vertex x is *dominated* by a vertex y if, whenever x belongs to some hyperedge e, then y also belongs to e. Then, we can simply delete x from the vertex set and from all edges it belongs to.

Notice that the tiny edge rule puts a vertex into the hitting set. Since this is an exact rule, we may worsen the solution in order to obtain a sufficiently approximated solution. One easy consequence of the vertex domination rule is that we can assume a minimum degree of two in an irreducible instance. Moreover, for each vertex pair (x, v) of an irreducible instance, there exists an edge pair (*irreducibility witness*) (e, f) with $x \in f, v \notin e$, but $x \notin f, v \in e$.

The following simple rule preserves the approximation factor of two.

- small edges: If e is a hyperedge of size two, i.e., $e = \{x, y\}$, then put both x and y into the hitting set.

In general, we will always first employ exact reduction rules before employing approximative reduction rules.

Analyzing a simplistic branching. In the first case of the branching, v is put into C, and afterwards, the solution is worsened by putting all vertices of an edge e

into C, as well. Since any optimum solution that contains v will also contain at least one vertex from e, while the algorithm will, altogether, put $e \cup \{v\}$ into C, with $|e \cup \{v\}| \leq 4$, this locally preserves the claimed 2-approximation factor.

In the second case of the branching, v is not put into C. Clearly, there is some edge $e \in E$ containing v. In this second case, v will be removed from e. In the recursive call, the small edges rule triggers and puts from e into C.

So, at this point of the analysis, we face one $(2, 1)$ branching vector, i.e., a branching number of 1.619, obviously worse than what we got by profiting from earlier analysis for exact algorithms. However, there is some hope that we might get better running time estimates. For instance, since we know that the vertex v is (at least) of degree three, we might find that there are edges e, f, g containing e such that $|e \cup f \cup g| = 7$. Now, in Case 2 of the branching, three small edges are produced. This alone gives already a $(2, 3)$ branch, i.e., $\mathcal{O}^*(1.325^K)$, for our factor-2 approximation. We deliver a detailed analysis in the next section.

4 A More Elaborated Analysis of a Factor-2 Approximation Algorithm for 3-HS

More approximation-preserving reduction rules.

- approximative vertex domination: Assume there is a hyperedge $e = \{x, y, z\}$ such that, whenever x belongs to some hyperedge h, then y or z also belong to h. Then, we put y and z together into the hitting set that we produce.

Lemma 1. *The approximative vertex domination rule is correct for an algorithm aiming at a factor-2 approximation.*

Proof. Namely, assume that an optimum solution contains x. Then, we can replace x in that solution by y and z, losing a factor of two, but still having a valid hitting set. If x is not in any optimum solution C, then, in order to hit e, y or z (or both) must be in C, so our rule loses again at most a factor of two. □

This rule alone is already very powerful. Consider any vertex v that Algorithm 1 chooses for branching. By approximative vertex domination, v must belong to two edges e, f with $|e \cup f| = 5$, as otherwise, fixing some e where v belongs to, all edges f that contain v would also contain some other vertex of e. In particular, this reasoning shows that there is for any vertex v of degree two, the two edges containing v will together host five vertices. This yields a branching vector of $(2, 2)$, which already improves on the much more sophisticated exact algorithms that were (ab)used to produce approximative solutions in Proposition 1.

- small triangle situation: Assume there are three small hyperedges $e = \{y, z\}$, $f = \{x, y\}$, $g = \{x, z\}$. This describes a triangle situation (e, f, g). Then, we put $\{x, y, z\}$ together into the hitting set, and we can even choose another hyperedge of size three to worsen the ratio.

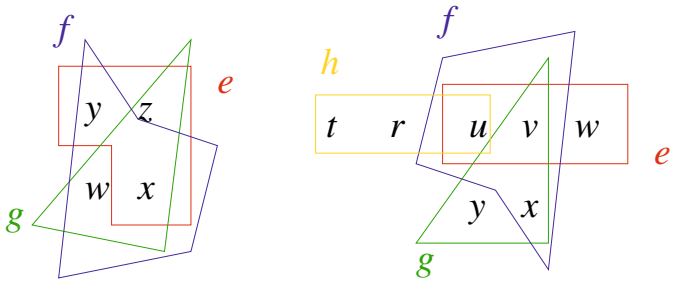


Fig. 1. Special situations for better branching

Notice that it is clear that two of the three vertices $\{x, y, z\}$ must be in any optimum solution. So, if we put $\{x, y, z\}$ together with a whole edge h into the hitting set, out of these six vertices, three must be in any optimum solution. This shows the validity of this rule. To improve our branching, we will always first look for small triangle situations before applying the small edges rule.

- approximative double vertex domination: Assume there is a hyperedge $e = \{x, y, a\}$ and another vertex b such that, whenever x or y belong to some hyperedge h, then a or b also belong to h. Then, we put a and b together into the hitting set that we produce.

Lemma 2. *The approximative double vertex domination rule is correct for an algorithm aiming at a factor-2 approximation.*

Proof. Assume that an optimum solution contains at least one of the vertices x and y. Then, we can replace The vertices x and y in that solution by a and b, losing at most a factor of two, but still having a valid hitting set. If neither x nor y is in any optimum solution C, then, in order to hit e, a must be in C, so our rule loses again at most a factor of two. □

We will now describe how this new reduction rule can help in certain branching scenarios. Assume there are three hyperedges $e = \{x, y, z\}$, $f = \{x, y, w\}$, $g = \{x, w, z\}$. This describes a potential triangle situation $(x; e, f, g)$ in the sense that, when branching at vertex x, in the branch that does not put x into the hitting set, a small triangle situation will be produced. Hence, we can reduce the parameter k by three in that branch.

We can experience a similar profit from vertex domination. We assume there are hyperedges $e = \{u, v, w\}$, $f = \{u, v, x\}$ and a further edge $g \neq \{v, w, x\}$ containing v but not u; we can further assume $\deg(u) \geq 3$ (as otherwise v dominates u), and that none of the further edges h_i containing u also contain v. Finally, we consider edge $h = \{u, r, t\}$ and assume that all edges h_i containing u, also contain r or t. We then consider a binary branch at v. In the branch when v is put into the hitting set, the approximative vertex domination rule triggers on vertex u. Hence, we can reduce the parameter by at least three in this branch

(including the worsening step). In the case when v is not put into the hitting set, then the small edges $e' = e \setminus \{v\} = \{u, w\}$, $f' = f \setminus \{v\} = \{u, x\}$ and $g' = g \setminus \{v\} \neq \{x, w\}$ (with $u \notin g'$) are produced, so that either $e' \cap g' = \emptyset$ or $f' \cap g' = \emptyset$. Hence, the small edge rule can be performed twice, yielding a parameter reduction of two. We will term $(v; e, f, g)$ a potential vertex domination situation.

Algorithm 2. `3HS-2-appr`: A more specific 2-approximation algorithm for 3-HS

```
1: Input: Hypergraph G = (V, E), parameter k, and a partial hitting set C
2: Output: Either NO or a hitting set C with |C| ≤ 2K.
3: Apply all reduction rules exhaustively, possibly modifying C and k.
4: if k < 0 then
5:    Return NO.
6: else if possible: choose (a) an internal or (b) [if (a) fails] an external branching
   pair (u, v) then
7:    Binary branch on {u, v} as follows:
8:    Case 1: Put u, v ∈ C, i.e.,
          C ← C ∪ {u, v}, V ← V \ {u, v}, E ← {e ∈ E | e ⊆ V}, k ← k − 1
      Recursively call 3HS-2-appr with the modified parameters.
9:    Case 2: Do not put u, v into C, i.e.,
          V ← V \ {u, v}, E ← {e \ {u, v} | e ∈ E}.
      Recursively call 3HS-2-appr with the modified parameters.
10: else
11:    Solve the remaining instance optimally using an Edge Cover algorithm.
12:    Return either NO or a hitting set C with |C| ≤ 2K.
```

Explaining a more refined branching. In Algorithm 2, we follow another branching strategy in a hypergraph $G = (V, E)$ by selecting branching pairs. More specifically, a pair (u, v) of vertices, where $\deg(u), \deg(v) \geq 3$, is called a *branching pair* if one of two conditions is met:

- If there are two hyperedges $e = \{u, v, w\}$, $f = \{u, v, x\}$, then (u, v) is an *internal* branching pair. We branch on this case with preference.
- If in the hypergraph $G' = (V', E')$ that is obtained from G by deleting u and all the edges u belongs to, v has still degree at least three, then (u, v) is an *external* branching pair.

In the penultimate line of the algorithm, it might still be that there are vertices of degree at least three. However, as we do not find any branching pair, these vertices will disappear in a single branch that will afterwards allow the use of an Edge Cover algorithm to solve the Hitting Set instance with maximum degree of two.

Clearly, an optimum solution either contains u or v, or neither u nor v. We are worsening this case distinction by a factor of two if we consider the case when u or v is in an optimum solution together, so taking $\{u, v\}$ into the solution. This is done in the first branch. If this is not the case, neither u nor v are in an

optimum solution. However, in the second branch, when u, v are removed from the instance, reduction rules will apply We are giving a more refined analysis in the following theorem. The following lemma is crucial to show the running time of `3HS-2-appr`.

Lemma 3. *If the maximum degree in an irreducible hypergraph G is at least three, then there must exist a vertex v with* $\deg(v) \geq 3$*, called* preferred vertex*, that satisfies one of the following cases.*

1st branching scenario *There is another vertex $u \in V$ and three edges e, f, g such that $\{u, v\} = e \cap f \cap g$.*

2nd branching scenario *There are 3 edges e, f, g s.t. (A) $\{v\} = e \cap f \cap g$ and $|e \cup f \cup g| = 7$ or $(v; e, f, g)$ describes (B) a potential triangle situation or (C) a potential vertex domination situation.*

Proof. Consider an irreducible hypergraph $G = (V, E)$ with maximum degree at least 3, and consider a vertex $v \in V$ with degree at least 3. Then one of the following three cases must be satisfied:

1. There are three edges e, f, g containing v such that $e \cap f \cap g = \{u, v\}$; then we have the first branching scenario.
2. There are no three edges e, f, g containing v such that $e \cap f \cap g = \{u, v\}$, but there are two edges containing v such that $e \cap f = \{u, v\}$, say $e = \{u, v, w\}$ and $f = \{u, v, x\}$.
   - If there exists an edge $h = \{v, w, x\}$ or $h = \{u, w, x\}$ then $\{v; e, f, h\}$ (or $\{u; e, f, h\}$) is a potential triangle situation (case (B) in the second branching scenario). In what follows we assume that such edge h does not exist.
   - Since the hypergraph G is irreducible, the vertex u also obeys $\deg(u) \geq 3$.

     Claim: There is an edge g such that $g \cap (e \cup f) = \{v\}$ or $g \cap (e \cup f) = \{u\}$. Namely, any edge h ($h \notin \{e, f\}$) containing v [or u, resp.] will not contain u [or v, resp.] to avoid the first branching scenario. To falsify the claim, $h \cap \{w, x\} \neq \emptyset$. To avoid the potential triangle situation described in the previous item, $|h \cap \{w, x\}| = 1$. Still, any hyperedge h containing u or v also contains w or x, which is not possible, as the approximative double vertex domination rule would have dealt with this situation. ◊

     Without loss of generality, assume that $g \cap (e \cup f) = \{v\}$, and let j be another edge containing u (but not v).
     - If there are no more edges containing v, then $\{u; e, f, j\}$ describes a potential vertex domination situation.
     - If there is another edge ℓ containing v, then in order to avoid the potential triangle situation $\ell \neq \{v, w, x\}$ and thus $|\ell \cap \{w, x\}| \leq 1$. Hence, $\{v; e, g, \ell\}$ or $\{v; f, g, \ell\}$ describe the second branching scenario, case (A).
3. There are no two edges e, f containing v such that $e \cap f = \{u, v\}$ for some vertex u; then we have the second branching scenario, case (A). □

Table 2. A list of branching vectors, keeping track of the worsening steps

Situation	Branching vector	subcubic case?	with $w = 1$	$w = 1.5$	$w = 2$
1st branching sc.	$(1, 3 + 3w)$	No	1.2852	1.2431	(No)
2nd b.s., Case (A)	$(1 + w, 3)$	Yes	1.3196	1.2600	1.2356
2nd b.s., Case (B)	$(1 + w, 2 + w)$	No	$\leq$ Case (A)	$\leq$ Case (A)	$\leq$ Case (A)
2nd b.s., Case (C)	$(2 + w, 2)$	Yes	1.3196	1.2600	1.255

Remark 1. As an aside, let us mention that there is a simple variant of Algorithm 1 that branches on preferred vertices, if possible internal branching pairs, and would obtain the branching numbers listed in Table 2, referring to the analysis of the previous lemma. The parameter w refers to a worsening step; usually, $w = 1$. Our new analysis profits from external branching pairs, as we will see.

Theorem 1. *`3HS-2-appr` can be used to find a 2-approximation* 3-HS *of a 3HS-instance $G = (V, E)$ in time $\mathcal{O}^*(1.2852^K)$.*

Proof. The correctness of the Algorithm 2 has been discussed before.

Now we turn to the running time analysis. The first branching scenario is encountered if the algorithm branches on an internal branching pair. It assumes the existence of two vertices u and v and three edges e, f, g such that $e \cap f \cap g = \{v, u\}$. Hence, the recursive call faces three tiny edges: $e' = e \setminus \{v, u\}$, $f' = f \setminus \{v, u\}$, and $g' = g \setminus \{v, u\}$. The tiny edges rule will then put three vertices into the hitting set, but since this is an exact reduction rule, three additional independent edges can be selected and put into the hitting set. Altogether, this yields a branching vector of $(1, 6)$ and a branching number of 1.2852.

If no internal branching pair exists, then we face the 2nd branching scenario described in Lemma 3. Let v be some preferred vertex. Let V_v and E_v collect all vertices and edges that are directly affected by branching at v. This means: (a) If $\{v\} = e \cap f \cap g$ and $|e \cup f \cup g| = 7$ or if $(v; e, f, g)$ describes a potential triangle situation, then $V_v = e \cup f \cup g$, $E_v = \{e, f, g\}$; (b) if $(v; e, f, g)$ describes a potential vertex domination situation, then $V_v = e \cup f \cup g \cup h$, where h is the edge where the (approximative) vertex domination rule will apply to, see the discussion leading to Fig. 1, and E_v collects all edges containing vertices from V_v. Consider $G' = (V', E')$, where $V' = V \setminus V_v$, $E' = E \setminus E_v$. If the maximum degree in G' is at most two, then by branching at v alone according to the 2nd branching scenario, possibly followed by branching at other vertices from E_v, we will produce a hypergraph of maximum degree two that can be solved in polynomial time. The finitely many branches indicated in the previous sentence do not affect the overall running time.

Note that there is no preferred vertex in G' that corresponds to the 1st branching scenario, as such vertex would have been previously selected by the algorithm instead of v as part of an internal branching pair. Hence, we will find a vertex v' that fits into one of the cases of the second branching scenario. For the purpose of analyzing this part of the algorithm, we assign parameter a to vertices v and v', where $a(v) = 1$ if $\{v; e, g, f\}$ describes a potential vertex domination situation, and $a(v) = 0$ otherwise; similarly, $a(v') = 1$ if $\{v'; e', g', f'\}$ describes a

potential vertex domination situation, and $a(v') = 0$ otherwise. Then we branch as follows:

Case 1. At least one of the vertices v and v' is in a minimum hitting set respecting previous choices; then we put $\{v, v'\}$ in C and have a parameter reduction of 1. Additionally, if $\{v; e, g, f\}$ and/or $\{v'; e', g', f'\}$ describes a potential vertex domination situation, we add additional vertices to C, as described above. Thus the total parameter reduction is $1 + a(v) + a(v')$.

Case 2. None of the vertices v and v' is in any minimum hitting set respecting previous choices; then we simply remove v and v' from the vertex set V and from all the edges that contain v or v'. The total parameter reduction is $6 - a(v) - a(v')$.

For the claimed parameter reductions to be true, it is crucial to observe that the branching at v and v' is done independently, as it is guaranteed by the construction of V_v and E_v. Hence, the reductions follow from what we collected for branching at a single vertex in Table 2. In total, we have a branching vector $(1 + a(v) + a(v'), 6 - a(v) - a(v')$. Depending on the values $a(v)$ and $a(v')$, the branching vector can be $(1, 6)$, $(2, 5)$ or $(3, 4)$. Out of the 3 corresponding branching numbers, the largest one is 1.2852, corresponding to $(1, 6)$. □

5 Approximating 3-HS with Degree Constraints

In the related case of Minimum VC, quite some research was undertaken to find better approximations for the degree-restricted case, e.g., for the case of cubic graphs. Surprisingly, to the best knowledge of the author, no such results are known for 3-HS. Also for the problem of finding smaller kernels, only relatively small progress was reported in [9], though that paper is far from trivial. Here, we are going to report on several results, focussing on consequences of running Algorithm 2 on subcubic instances, i.e., instances where each vertex belongs to at most three hyperedges. Proofs are omitted due to space restrictions.

Lemma 4. *If $G = (V, E)$ is a subcubic irreducible 3-HS instance, then we know: If there are two edges $e, f \in E$ with $|e \cap f| = 2$, then for any further edge g with $v \in g$, $e \cap g = e \cap f = \{v\}$.*

Lemma 5. *If $G = (V, E)$ is a subcubic irreducible 3-HS instance, then no potential triangle situation occurs.*

In the following, we assume (in addition), that hyperedge components with at most 27 vertices are solved (exactly) due to table look-up. This will be called the small component rule. This rule, as well as the tiny edge and the small triangle rule are exact rules that put a vertex into the hitting set, so that a worsening step triggers. Hence, these three rules are summarized as *trigger rules*. The following auxiliary results turns out to be useful for proving the crucial Lemma 8.

Lemma 6. *Whenever a hypergraph component completely disappears when applying reduction rules, then the last reduction rule applied was a trigger rule.*

Lemma 7. *Let $G = (V, E)$ be a subcubic irreducible 3-HS instance. The removal of $H \subseteq V$, $|H| \leq 3$, cannot destroy any hypergraph component.*

Lemma 8. *After branching on a subcubic instance or after performing a worsening step, i.e., after the corresponding vertices were put into the hitting set, we find a vertex of degree two or a yet unaccounted small edge, unless we have entered the final polynomial-time phase.*

Theorem 2. *Algorithm 2 can be implemented to find a 2-approximation for* 3-Hitting Set *on subcubic instances in time $\mathcal{O}^*(1.2555^K)$.*

Proof. Due to Lemmas 4 and 5, only the cases marked with "Yes" in Table 2 may occur when running Algorithm 1, modified towards branching on preferred vertices as indicated in Remark 1. A yet unaccounted small edge will first trigger the small edge or the small triangle rule, clearly allowing to add one to each component of the branching vector, yielding, in particular, $w \geq 2$. Otherwise, let v be a vertex of degree two, as it exists due to Lemma 8, pertaining to hyperedges e and f. The idea is to exploit the worsening step as follows. If any hyperedge h containing some vertex from $X = (e \cup f) \setminus \{v\})$, with $h \notin \{e, f\}$, has two vertices from X, then X induces a small hyperedge component, again allowing to reduce the parameter by at least one. So, there is a hyperedge h containing exactly one vertex from $X = (e \cup f) \setminus \{v\}$, with $h \notin \{e, f\}$. Slightly modifying our algorithm, we will always pick such a hyperedge h in the worsening step. This will put (at least) two more vertices in the approximative hitting set compared to what we already accounted for, due to the vertex domination and small edge rules. So, we can always rely on $w \geq 2$ in the branching vectors. Hence, we are facing as worst-case branching vectors: $(3, 3)$ and $(4, 2)$. □

We can make use of the same idea for subcubic instances of d-HS in general. However, we must be careful with the interplay between the intended approximation factor and the corresponding small edges rule. We give some details for the case $d = 4$ in the following. If we put a hyperedge of size 3 into the hitting set, then this gives us an approximation factor of three (only). We show two ways how to deal with this problem: either, we aim at an approximation factor of three only, or we have to set up recurrences that allow for an improved factor-2.5 approximation. The figures are based on Table 1.

Theorem 3. Minimum 4-HS *can be approximated in time $\mathcal{O}^*(1.4613^K)$ up to a factor of three in general and in time $\mathcal{O}^*(1.2556^K)$ on subcubic hypergraphs.*

Theorem 4. Minimum 4-HS *can be approximated in time $\mathcal{O}^*(1.7650^K)$ up to a factor of* 2.5 *in general and in time $\mathcal{O}^*(1.5754^K)$ on subcubic hypergraphs.*

Let us move back to the 3-HS case again, but now applying the idea mentioned last to hypergraphs of maximum degree four. Similar to Lemma 8, we can now assume that, before applying any worsening step, we might find a vertex of degree three in the graph. So, with the help of two subsequent worsening steps, we can produce a small hyperedge due to vertex domination. This means (again) that we can assume $w = 3/2$ in Table 2. Evaluations the corresponding branching vectors are also shown there prove:

Theorem 5. Minimum 3-Hitting Set *can be approximated in time* $\mathcal{O}^*(1.3196^K)$ *up to a factor of two in hypergraphs of maximum degree four.*

We conclude with a new polynomial-time approximation algorithm:

Algorithm 3. `3HS3-2.5-appr`: A 2.5-approximation algorithm for 3-HS3

```
1: Input: Hypergraph G = (V, E) of maximum degree three
2: Output: a hitting set C with |C| ≤ 2.5|C*|, where C* is an optimum solution
3: Initially branch on an arbitrary hyperedge (if it exists).
4: while E ≠ ∅ do
5:    Apply all reduction rules exhaustively.
6:    if the hypergraph is 3-regular then
7:       Pick a hyperedge h and put it into C.
8:    Pick a hyperedge e s.t. some neighbor x ∉ e of some vertex v ∈ e obeys deg(x) =
      2. // Irreducibility witness.
9:    Put e into the hitting set C.
10: end while
11: Return C.
```

Theorem 6. Minimum 3-Hitting Set *can be approximated in polynomial time up to a factor of* 2.5 *in hypergraphs of maximum degree three.*

Proof. 1. As long as Line 7 in Algorithm `3HS3-2.5-appr` is not executed, the algorithm works fine: It would put a hyperedge e into C, reduce the degree of x to one, so that vertex domination triggers, followed by the small edge rule on $f \setminus \{x\}$, putting in total 5 vertices in C. As at least one vertex of f and of e must be in C^*, by a local ratio argument, we achieve a factor of 2.5.
2. We can afford that the tiny edge, the small triangle and the small (non-trivial) component rule are followed by a worsening step, still staying within the promised approximation factor.
3. We claim that Line 10 either creates a new vertex of degree two (see 1.), or it triggers one of the three mentioned rules, so that Line 7 can be executed as a worsening step (see 2.). □

6 Further Questions

(1) Can we employ other forms of exact parameterized algorithms to obtain parameterized approximation algorithms, for instance, those relying on a Measure & Conquer analysis (see [12]) or on iterative compression, (see [8]). (2) Can we further improve on the running time analysis, for instance, by making use of a Measure & Conquer style analysis in the cubic case? (3) Can the techniques presented be extended to work for the weighted case, as known for moderately exponential-time approximation algorithms [5]? (4) The method that we employed for obtaining approximation algorithms is reminiscent of the well-known local ratio method [1]. This deserves further exploration. (5) Can other prominent techniques from polynomial-time approximation be employed for exponential-time approximation?

References

1. Bar-Yehuda, R.: One for the price of two: a unified approach for approximating covering problems. Algorithmica 27, 131–144 (2000)
2. Bourgeois, N., Escoffier, B., Paschos, V.T.: Efficient Approximation of Combinatorial Problems by Moderately Exponential Algorithms. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 507–518. Springer, Heidelberg (2009)
3. Brankovic, L., Fernau, H.: Combining Two Worlds: Parameterised Approximation for Vertex Cover. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part I. LNCS, vol. 6506, pp. 390–402. Springer, Heidelberg (2010)
4. Chen, Y., Grohe, M., Grüber, M.: On Parameterized Approximability. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 109–120. Springer, Heidelberg (2006)
5. Cygan, M., Kowalik, Ł., Wykurz, M.: Exponential-time approximation of weighted set cover. Information Processing Letters 109, 957–961 (2009)
6. Fernau, H.: Parameterized algorithmics for d-hitting set. International Journal of Computer Mathematics 87(14), 3157–3174 (2010)
7. Fernau, H.: A top-down approach to search trees: Improved algorithmics for 3-HITTING SET. Algorithmica 57, 97–118 (2010)
8. Fomin, F.V., Gaspers, S., Kratsch, D., Liedloff, M., Saurabh, S.: Iterative Compression and Exact Algorithms. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 335–346. Springer, Heidelberg (2008)
9. Kanj, I.A., Zhang, F.: 3-HITTING SET on Bounded Degree Hypergraphs: Upper and Lower Bounds on the Kernel Size. In: Marchetti-Spaccamela, A., Segal, M. (eds.) TAPAS 2011. LNCS, vol. 6595, pp. 163–174. Springer, Heidelberg (2011)
10. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within $2-\varepsilon$. Journal of Computer and System Sciences 74, 335–349 (2008)
11. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32, 57–95 (1987)
12. Wahlström, M.: Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems. PhD thesis, Department of Computer and Information Science, Linköpings universitet, Sweden (2007)

Approximation Algorithms for the Maximum Leaf Spanning Tree Problem on Acyclic Digraphs

Nadine Schwartges, Joachim Spoerhase, and Alexander Wolff

Chair of Computer Science I, University of Würzburg
http://www1.informatik.uni-wuerzburg.de/en/staff

Abstract. We consider the problem Maximum Leaf Spanning Tree (MLST) on digraphs, which is defined as follows. Given a digraph G, find a directed spanning tree of G that maximizes the number of leaves. MLST is NP-hard. Existing approximation algorithms for MLST have ratios of $O(\sqrt{\mathrm{OPT}})$ and 92.

We focus on the special case of acyclic digraphs and propose two linear-time approximation algorithms; one with ratio 4 that uses a result of Daligault and Thomassé and one with ratio 2 based on a 3-approximation algorithm of Lu and Ravi for the undirected version of the problem. We complement these positive results by observing that MLST is MaxSNP-hard on acyclic digraphs. Hence, this special case does not admit a PTAS (unless $\mathcal{P} = \mathcal{NP}$).

1 Introduction

Network design deals with the problem of optimally connecting a given set of network nodes by links. Network design problems arise for example in the planning of telecommunications networks, logistical networks or in circuit layout. Often, network design problems are modeled as graph optimization problems. Specifically, the goal is to find a subgraph G' of a given graph G so that G' meets certain connectivity requirements and optimizes a quality measure tailored to the respective application.

An important class of network design problems are *spanning tree* problems. Here, a solution of the problem has to satisfy only a minimum connectivity requirement. Specifically, there must be a node r—the root—such that every node is reachable from r by some path. Spanning trees use the minimum number of edges (links) among all subgraphs of G with this property.

A prominent spanning tree problem is Minimum Spanning Tree (MST), where every edge of the input graph G has an associated cost. The goal is to find a spanning tree whose total edge cost is minimum. A natural extension of this problem is Steiner Tree, where only a given subset T of so-called *terminal nodes* needs to be connected.

In contrast to MST, the quality measure of the spanning tree problem we investigate here is associated with the *nodes*, not the edges. This assumption is

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 77–88, 2012.

driven by applications in which the network nodes perform a certain function. We assume that nodes of higher degree have more sophisticated and thus also costlier functionality. Specifically, if we distinguish only between pure receivers (leaves of the tree) and routers (internal nodes), which are more expensive, we arrive at the Maximum Leaf Spanning Tree problem (MLST): given a graph G with root r, the task is to find an r-rooted spanning tree that maximizes the number of leaves. MLST is one of the classical NP-hard problems listed by Garey and Johnson [11].

We consider *digraphs*, that is, edges can only be traversed in one direction. Directed MLST is NP-hard, too, since it is a generalization of the undirected version. This motivates our interest in approximations. Although approximation algorithms are known for digraphs, their performance guarantees are not satisfactory. Therefore, we focus on the special case of *acyclic* digraphs, which is still NP-hard [2]. It turns out that we can exploit the special structure of acyclic digraphs to obtain guarantees that are significantly better than those known for general digraphs.

Previous results and related work. On undirected graphs, MLST is well-investigated. It is known that undirected MLST is NP-hard [11]. Galbiati et al. [10] showed that undirected MLST is even MaxSNP-complete, that is, there is no polynomial-time approximation scheme (PTAS) for this problem (if $\mathcal{P} \neq \mathcal{NP}$).

These negative results have stimulated the development of a series of approximation algorithms for MLST. Improving on their own earlier results, Lu and Ravi [13] developed a nearly-linear-time 3-approximation algorithm based on an *expansion* strategy. Basically, this strategy consists of growing a subforest F of the input graph by iteratively connecting nodes to a maximal set of edges so that F remains a forest. The expansion idea originally goes back to Kleitman and West [12] who considered graphs with bounded minimum degree and derived lower bounds for the maximum number of leaves of spanning trees in such graphs.

Solis-Oba [14] later proposed a linear-time algorithm based on the ideas of Kleitman and West [12] and Lu and Ravi [13]. By means of a clever analysis he showed that his algorithm is not only faster than the algorithm of Lu and Ravi but also gives a 2-approximation. So far, better results have been obtained only for special graph classes such as cubic graphs, the currently best being a 3/2-approximation algorithm [4].

Recently, there has been a lot of interest in the *directed* version of MLST. As often in network design, the directed case seems to be much harder than the undirected one. Drescher and Vetta [9] pointed out that the techniques that are successful for undirected graphs—namely, edge-swapping and expansion—fail for digraphs. They end up giving an algorithm for directed MLST with a ratio of $O(\sqrt{\mathrm{OPT}})$, which is considerably worse than the ratio 2 known for undirected graphs [14]. Daligault and Thomassé [8] improved upon this result by providing a 92-approximation algorithm. The techniques employed in both of the above algorithms differ completely from the approaches for the undirected case.

A large portion of the research on directed MLST has focused on the development of fixed-parameter tractable algorithms. The parameterized version

of MLST includes an additional parameter k. The goal is to decide whether a given graph has a spanning tree with at least k leaves. The currently fastest fixed-parameter tractable algorithm is due to Gutin et al. [7] and has a running time of $3.72^k \cdot n^{O(1)}$ where n is the number of nodes in the input graph. There are also specialized fixed-parameter results for *acyclic* digraphs [1,2], that is, for the graph class considered in this work.

The currently fastest (unparameterized) exact algorithm was given by Binkele-Raible and Fernau [3]. It runs in $O^*(1.9043^n)$ time, where the O^*-notation neglects polynomial factors.

Our contribution. In this paper, we give two linear-time approximation algorithms for MLST on acyclic digraphs.

Our first result is a 4-approximation algorithm that makes use of a result of Daligault and Thomassé [8] who gave a lower bound on the number of leaves in a special class of digraphs.

In our second and main result we investigate the expansion approach, which has already led to several positive results for undirected MLST [12,13,14]. Applying the expansion idea to acyclic digraphs we obtain a 2-approximation algorithm. So we improve significantly upon the 92-approximation algorithm known for general digraphs and close up to the undirected case.

Our positive results are complemented by the observation that MLST in acyclic digraphs is MaxSNP-hard, that is, there is no PTAS for this problem (unless $\mathcal{P} = \mathcal{NP}$). That justifies the development of constant-factor approximation algorithms for MLST in acyclic digraphs.

To stress the relevance of our main result, let us compare MLST to Steiner Tree (ST), which can be considered paradigmatic among the tree-based network design problems. The best known algorithm for undirected ST has a ratio of roughly 1.39 [5]. This has to be compared to the best known algorithm for the directed case, which has a performance guarantee of $O(n^\varepsilon)$ [6], for any $\varepsilon > 0$. There is a specialized approximation algorithm solving ST in acyclic digraphs but it yields the same result [15]. It can even be shown that for the acyclic case the approximation ratio is lower-bounded by $\Omega(\log n)$ [15]. To sum up, even the acyclic directed case of ST is significantly harder than the undirected one.

In terms of general graphs, ST and MLST behave similarly. In both cases, the results for undirected graphs are much better than the results for digraphs w.r.t. approximation. For *acyclic* digraphs, however, the problems exhibit significant difference. For ST, the acyclic case is provably harder than the undirected one and no improvement upon the general case has been obtained so far. In this paper, we provide an example of a tree-based network design problem (namely MLST) for which acyclicity can be exploited very well. It turns out that both algorithm *and* proof are a lot simpler in the acyclic than in the undirected case.

Since both our algorithms have the same (linear) asymptotic running time, the expansion algorithm supersedes the 4-approximation algorithm. Nevertheless, we think it is worth describing both algorithms since they are based on two conceptually different existing approaches that yield strong results. Finally,

the analysis of the 4-approximation algorithm is considerably simpler than the analysis of the expansion algorithm.

We use n and m as shorthand for the numbers of nodes and edges of the given acyclic digraph G with root r. We denote an optimum spanning tree of G by T^* and the number of its leaves by OPT. Given an arbitrary spanning tree T of G, we denote the set of leaves of T by $L(T)$.

2 Indegree-Based Algorithm

In this section, we develop a 4-approximation algorithm based on (an extension of) a lemma by Daligault and Thomassé [8]. Let $V_{=1}$ be the set of nodes of indegree 1 in the given digraph G, and let $V_{\geq 2}$ be the set of nodes of indegree at least 2 in G.

Lemma 1 ([8]). *Any rooted acyclic digraph G has a spanning tree with at least $|V_{\geq 2}|/3$ leaves. Such a spanning tree can be computed in $O(m)$ time.*

Proof (Sketch). Daligault and Thomassé [8] prove the existence of a spanning tree with at least $(|V_{\geq 2}| + \deg(r) + 2)/3 \geq |V_{\geq 2}|/3$ leaves.

The proof of Daligault and Thomassé is constructive, and it is not hard to verify that the construction can be carried out in linear time. □

Our approximation algorithm is based on the following observation. Lemma 1 gives us already a good approximation in the case that $|V_{\geq 2}|$ is large enough in comparison to OPT. On the other hand, if $|V_{\geq 2}|$ is small then $|V_{=1}|$ is large. Since each of the nodes in $V_{=1}$ has exactly one incoming edge, *every* spanning tree (including the optimum one) must use these incoming edges. In other words, a large fraction of the edges are fixed, which leaves less freedom for the choice of the remaining edges. Intuitively we expect that even an arbitrary spanning tree gives us a good approximation.

Theorem 1. *The algorithm of Lemma 1 is a 4-approximation algorithm for MLST on acyclic digraphs.*

Proof. Let $\alpha := |V_{\geq 2}|/\mathrm{OPT}$ and let T be the spanning tree output by the algorithm of Lemma 1. We now prove the following two bounds

$$|L(T)| \geq \frac{\alpha}{3}\mathrm{OPT} \tag{1}$$

$$|L(T)| \geq (1-\alpha)\mathrm{OPT}\,. \tag{2}$$

Bound (1) is an immediate consequence of the definition of α and Lemma 1.

For proving bound (2), we consider the graph $F = (V, E')$ with $E' = \{\,(u, v) \mid v \in V_{=1}\,\}$. The subgraph F of G is a forest containing only edges that are part of every spanning tree of G. Let L' be the set of leaves and isolated nodes of F, that is, the set of nodes with outdegree 0.

Consider an optimum spanning tree T^* of G. As argued above, F is a subforest of T^* and F has the same node set as T^*. Every leaf of T^* has outdegree 0 in T^*

and therefore also in F. Hence, L' contains the set of leaves of T^* as a subset. This yields $|L'| \geq \mathrm{OPT}$.

As F contains all edges ending in a node of $V_{=1}$, there are exactly $|V_{=1}|$ edges in F. The output tree T contains F as a subforest. Let us reconstruct T from F. To this end, we need $n - 1 - |V_{=1}| = |V_{\geq 2}|$ additional edges that are part of T but do not lie in F. By adding these edges to F, at most $|V_{\geq 2}|$ nodes in L' get connected with an outgoing edge. Hence, T contains at least $|L'| - |V_{\geq 2}|$ leaves. Using $|L'| \geq \mathrm{OPT}$ and the definition of α, we can conclude that

$$|L(T)| \geq |L'| - |V_{\geq 2}| \geq \mathrm{OPT} - \alpha\mathrm{OPT} = (1 - \alpha)\mathrm{OPT}\,.$$

This proves bound (2).

Now we balance bounds (1) and (2) to prove the approximation ratio 4. If $\alpha \geq 3/4$ then bound (1) yields $|L(T)| \geq \mathrm{OPT}/4$. On the other hand, if $\alpha \leq 3/4$ then bound (2) yields $|L(T)| \geq \mathrm{OPT}/4$. □

3 Expansion Algorithm

In this section, we present a linear-time 2-approximation algorithm for acyclic digraphs. Our algorithm and its analysis bear resemblances with the 3-approximation algorithm of Lu and Ravi [13] for undirected graphs. Therefore, we start with a brief outline of their algorithm.

3.1 Expansion Algorithm for Undirected Graphs

The algorithm of Lu an Ravi is based on a two-stage expansion strategy that works roughly as follows.

Stage I constructs a *leafy* subforest F of the input graph G. A forest is leafy if and only if any degree-2 node is adjacent to two nodes of degree at least 3. The leafy subforest F is constructed by *processing* the nodes of G iteratively in an arbitrary order. Processing a node u means to *expand* u if u has degree at least 3 after the expansion. The expansion of u adds to F a maximal set E_u of edges $(u, v) \in E(G)$ such that F remains a forest.

Stage II connects the subtrees created in stage I to a spanning tree of G in an arbitrary manner.

The total running time of the algorithm is $O(m\alpha(m, n))$, where $\alpha(\cdot\,, \cdot)$ is the inverse Ackermann function.

The 2-approximation algorithm of Solis-Oba [14] can be understood as a special case of the algorithm of Lu and Ravi, in which the nodes are processed in a particular order. More precisely, only leaves of F or singletons are expanded. Also, the connected components of F grow one by one rather than simultaneously. The particular node order does not only yield the better performance guarantee but also linear running time.

Our 2-approximation algorithm for acyclic digraphs closes up to the result of Solis-Oba for undirected graphs. Our analysis, however, is a lot simpler than that of Solis-Oba. In fact, our algorithm and its analysis are closer to the work of Lu

and Ravi. We remark that a (straightforward) adaption of Solis-Oba's algorithm to DAGs does not yield better results (confer Section 3.4).

3.2 Expansion Algorithm for Acyclic Digraphs

Similar to the algorithm of Lu and Ravi, our expansion algorithm for acyclic digraphs consists of an expansion stage in which a subforest F of G is created and a connection stage where this forest F is completed to a spanning tree.

A detailed description of our algorithm can be found in Algorithm 1 and in the procedures expansion and connection that implement the expansion and the connection stage. We use a node-marking technique. If a node is marked in these stages it indicates that the node already has an incoming edge belonging to F or is the root of F.

The connection stage is similar to the undirected case. Basically, the connected components of F are connected to each other in an arbitrary manner.

The expansion stage, however, has to be adapted to digraphs appropriately. Instead of requiring degree at least 3 as in the undirected case, we expand a node if it obtains outdegree at least 2. Also the implementation of the expansion operation simplifies. Whenever an edge (u, v) is added to F, we only have to make sure that v has indegree 0 in F. We accomplish this by means of node markings. The algorithm of Lu and Ravi has to check whether u and v lie in different connected components. This is why we can improve the running time from $O(m\alpha(m, n))$ to $O(m)$.

Algorithm 1. MaxLeafTwoApprox(G)

Input: acyclic digraph G with root r
Output: spanning tree T
mark r
$F \leftarrow$ expansion(G)
$T \leftarrow$ connection(G, F)
return T

Lemma 2. *Given an acyclic digraph G,* MaxLeafTwoApprox(G) *computes, in $O(m)$ time, a spanning tree of G.*

Proof. Recall that a node u is marked if and only if it has (exactly) one incoming edge or if $u = r$. No marked node can get further incoming edges. Hence, when the algorithm terminates, each node has either indegree 0 or 1 depending on whether it is marked or not. Since the connection stage marks all yet unmarked nodes, the result of the algorithm, F, is a subgraph of G that is acyclic (because G is) and in which every node except r has exactly one incoming edge. Thus, F is a spanning tree of G.

The linear running time can be achieved if the graph is represented by an adjacency list. Determining, for every $v \in V$, the set U_v of unmarked neighbors in procedure expansion takes $O(\sum_v \text{outdeg}(v)) = O(m)$ time in total.

Procedure expansion(G)

```
F ← ∅                                   { empty forest }
foreach node v in G do
    if v ∉ F then
        F ← F + v
    U_v ← unmarked endpoints of outgoing edges of v in G
    if |U_v| ≥ 2 then
        F ← F + U_v
        foreach u ∈ U_v do
            F ← F + (v, u)
            mark u
return F
```

Procedure connection(G, F)

```
foreach unmarked node v do
    choose an arbitrary incoming edge e of v in G
    F ← F + e
    mark v
return F
```

In procedure connection, connecting all yet unmarked nodes with an arbitrary incoming edge takes $O(n)$ time. □

3.3 Performance Guarantee

The expansion stage (procedure expansion) of our algorithm creates a forest F that possibly contains isolated nodes. Let $\bar{F}$ be the forest obtained by removing all isolated nodes from F. The forest $\bar{F}$ consists of a set $\{T_0, \ldots, T_k\}$ of node disjoint, non-trivial subtrees $T_i = (V_i, E_i)$, $i = 0, \ldots, k$. Let r_i be the root of subtree T_i.

Since procedure expansion expands only nodes of outdegree at least 2, none of the trees T_i, $i = 0, \ldots, k$ contains an interior node of outdegree 1. In other words, $\bar{F}$ contains only leaves and nodes of outdegree at least 2. This implies that at least half of the nodes in $\bar{F}$ are leaves as we show now.

Lemma 3. *For $i = 0, \ldots, k$, any subtree $T_i \in \bar{F}$ has at least $(|V_i| + 1)/2$ leaves.*

Proof. It is well known that a binary tree on n nodes has at least $(n + 1)/2$ leaves. Internal nodes of outdegree greater than 2 can only increase the number of leaves. □

We first consider only the leaves of an optimal spanning tree T^* that lie in $V(\bar{F})$. A trivial upper bound on the number of these leaves is $|V(\bar{F})|$. The forest $\bar{F}$, in turn, has at least $(|V(\bar{F})| + k + 1)/2$ leaves (because of Lemma 3) and is thus a good intermediate step in obtaining our desired 2-approximation algorithm for MLST.

To prove the overall performance guarantee we face, however, the following two problems. The first problem is that the procedure connection may connect leaves of $\bar{F}$ with outgoing edges thereby "killing" those leaves. The second problem is that the optimum T^* may well have additional leaves outside of $\bar{F}$. Concerning the first problem, we now show that connection kills at most k leaves of $\bar{F}$.

Lemma 4. *Procedure* connection *creates a tree with at least* $|L(\bar{F})| - k$ *leaves.*

Proof. Let n_0 denote the number of outdegree-0 nodes in F (that is, leaves and isolated nodes) at the beginning of an iteration of procedure connection, and let n_{cc} denote the current number of (possibly trivial) connected components of F. Note that n_{cc} drops by 1 and n_0 increases by 1 in each iteration of connection. This means that the value of $n_0 - n_{\mathrm{cc}}$ remains constant during the execution of the procedure.

This implies the claim since $n_0 - n_{\mathrm{cc}} = |L(\bar{F})| - (k+1)$ holds at the beginning of the procedure and, hence, also at the end when we have that n_0 equals the number of tree leaves and $n_{\mathrm{cc}} = 1$. □

The following lemma resolves the second above-mentioned problem—leaves outside of $\bar{F}$ cannot effectively increase the total number of leaves—and shows that the optimum kills *at least* k leaves in $V(\bar{F})$.

Lemma 5. *It holds that* $OPT \leq |V(\bar{F})| - k$.

Proof. Let T^* be an optimum spanning tree, and let R be the set of all roots $r_0, \ldots, r_k$ of $\bar{F}$ that are different from the "global" root r. Our proof works as follows. We identify a unique node for each root $r_i \in R$, its *witness* $q(r_i)$. We will make sure that each witness is an *internal* node of T^* that lies in $\bar{V} := V(\bar{F}) \cup \{r\}$. This shows that T^* has at most $|V(\bar{F})| - k$ leaves in $\bar{V}$. It does not rule out, however, that T^* has additional leaves outside of $\bar{V}$. To this end, we will additionally identify, for each leaf l of T^* outside of $\bar{V}$, a witness $q(l)$, that is, a unique internal node in T^* that lies in $\bar{V}$. We will then show that the map q is injective. This proves the claim: if T^* has ℓ leaves outside of $\bar{V}$, then T^* can have at most $|V(\bar{F})| - k - \ell$ leaves inside of $\bar{V}$.

To define the map q, consider a node z that is either a leaf of T^* not contained in $\bar{V}$ or a root in R. We define the node $q(z)$ to be the closest ancestor of z in T^* (excluding z itself) that lies in $\bar{V}$. Since the root r lies in $\bar{V}$ such a witness $q(z)$ always exists.

Let z and z' be distinct nodes in the domain of q. It remains to show that $q(z) \neq q(z')$. Assume to the contrary that $q(z) = q(z')$. Let P and P' be the paths in T^* from $s := q(z) = q(z')$ to z and to z', respectively. We distinguish two cases.

First, we consider the case that s, z and z' lie on a common path in T^*. Then we can assume without loss of generality that z is an internal node on the path P', which implies that z is not a leaf in T^*. Since z lies in the domain of q, z must be the root of some subtree T_i in $\bar{F}$. In particular, $z \in \bar{V}$. Thus, z is an ancestor of z' in T^* that lies in $\bar{V}$ and is closer to z' than $q(z') = s$. This contradicts the choice of $q(z')$.

Now, we consider the case that s, z and z' do *not* lie on a common path. Then there is a node u at which the paths P and P' split; see Fig. 1. Let v and v' be the successors of u on paths P and P', respectively. Either v or v' is marked by procedure expansion. For, if v and v' are still unmarked when node u is processed then u will be expanded thereby marking v and v'. We assume without loss of generality that v is the node marked by procedure expansion.

We claim that $z \neq v$. If z is a leaf of T^* that lies outside of $\bar{V}$, then $z \neq v$ because v—being marked—lies in $\bar{V}$. If z is the root of a subtree T_i for any $i \in \{0, \ldots, k\}$, then the claim follows because v has an incoming edge belonging to $\bar{F}$. Therefore, v is an ancestor of z in T^* that lies in $\bar{V}$ and is closer to z than $q(z)$. Again, this is a contradiction.

Both cases yield the desired contradiction. This completes the proof. □

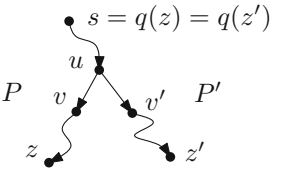


Fig. 1. Illustration of the case where s, z and z' do not lie on a common path

From Lemmas 2 to 5, we can deduce the main result of this paper.

Theorem 2. *The expansion algorithm for acyclic digraphs is a 2-approximation algorithm. It runs in linear time.*

Proof. Let T be the tree created by the expansion algorithm. Then we have

$$\frac{\mathrm{OPT}}{|L(T)|} \leq \frac{|V(\bar{F})| - k}{|L(\bar{F})| - k} \leq \frac{|V(\bar{F})| - k}{(\sum_{i=0}^{k}(|V_i| + 1)/2) - k} = \frac{2(|V(\bar{F})| - k)}{|V(\bar{F})| - k + 1} \leq 2,$$

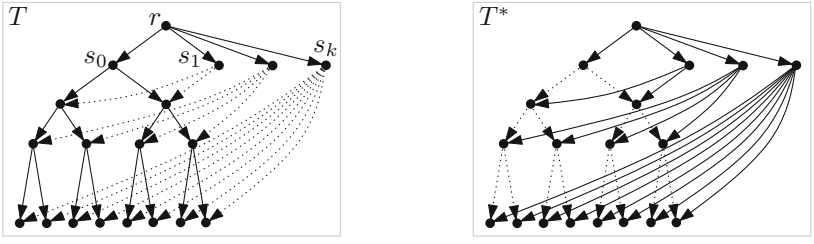
where the first inequality is due to Lemmas 4 and 5, the second inequality is due to Lemma 3, and the equality follows from the fact that $\sum_{i=0}^{k} |V_i| = |V(\bar{F})|$. □

3.4 Tight Example

We construct an infinite sequence $G_1, G_2, \ldots$ of rooted acyclic digraphs such that the performance ratio of Algorithm 1 on this sequence tends to 2.

For any positive integer k, let the root r of G_k have $k+1$ successors $s_0, \ldots, s_k$, see Fig. 2. The node s_0 is the root of a perfect binary tree B_k with $k+1$ levels $L_0 = \{s_0\}, L_1, \ldots, L_k$. For $i = 1, \ldots, k$, there is an edge from s_i to each node in level L_i. This completes the description of G_k.

Since the order in which our algorithm expands the nodes is not specified, we can assume that the algorithm first expands the root r and then the perfect



(a) result T of our algorithm applied to G_k (b) optimum spanning tree T^* of G_k

Fig. 2. Tight example G_k (drawn for $k = 3$) with two different spanning trees; solid edges represent tree edges, dotted edges represent non-tree edges

binary tree B_k. Then the spanning tree that our algorithm outputs has $2^k + k$ leaves in total; 2^k leaves in B_k plus the k leaves $s_1, \ldots, s_k$.

On the other hand, in the optimum solution T^* every node of B_k is a leaf. Thus, $\mathrm{OPT} = 2^{k+1} - 1$. Clearly, the performance ratio $(2^{k+1} - 1)/(2^k + k)$ of our algorithm approaches 2.

Note that the above suboptimal spanning tree can also be obtained when we apply (a straightforward adaption of) Solis-Oba's algorithm [14] to G_k, that is, if we expand always at the leaves of the current subtree. This demonstrates that Solis-Oba's algorithm, too, does not yield better results for DAGs. Finally, this example remains valid even if the algorithm expands the nodes in *topological* order (which appears most natural).

4 MaxSNP-Hardness

Galbiati et al. [10] prove that the undirected MLST problem is MaxSNP-hard, which implies that there is no PTAS for undirected MLST (unless $\mathcal{P} = \mathcal{NP}$). Their hardness proof consists of a so-called *L-reduction* in which they use a special class of instances for undirected MLST. We now show that, for this special class, the undirected and the acyclic directed case are equivalent.

The undirected graphs that Galbiati et al. use in their proof have the structure depicted in Fig. 3 (a). These graphs consist of three levels of nodes. Each level has the same cardinality. Each node in level 1 is connected to the root r and each node in level 3 is connected to its counterpart in level 2. Additional edges connect only nodes between level 1 and level 2. Let G be an undirected graph with such a structure. Galbiati et al. show, that for any spanning tree T in G there is a spanning tree T' with the same number of leaves such that any node in level 1 is directly connected to the root r. We call the tree T' *valid*. Confer Fig. 3 (b) for a valid spanning tree. Galbiati et al. only use valid spanning trees in their L-reduction.

Given G, we construct an acyclic digraph D by orienting the edges of G so that r is the root of D and each edge starting in level i ends in level $i + 1$, where $i = 1, 2$. The remaining edges emanate from r; see Fig. 3 (c).

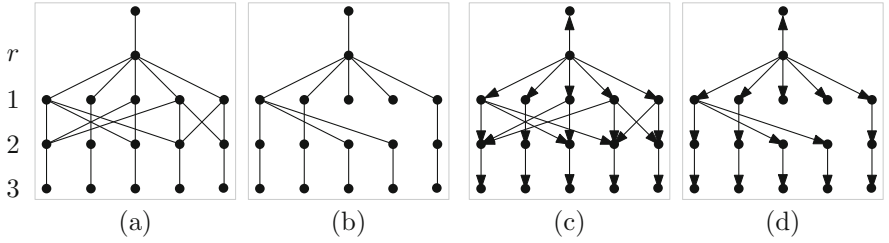


Fig. 3. Example of a graph used in the MaxSNP-hardness proof of Galbiati et al. [10]

We observe that there is a one-to-one correspondence between valid spanning trees of G and spanning trees of D. The unique orientation of any valid spanning tree of G yields a spanning tree of D with the same number of leaves. Conversely, each spanning tree T of D must contain all edges between r and level 1 and all edges between level 2 and level 3 since nodes in levels 1 and 3 have indegree one. Hence, the undirected tree corresponding to T is a valid spanning tree of G with the same number of leaves.

To sum up, the above equivalence shows that MLST on acyclic digraphs is MaxSNP-hard.

5 Concluding Remarks

Summarizing, we have given two linear-time approximation algorithms for solving the acyclic directed MLST problem with ratios of 4 and 2, respectively. The 4-approximation algorithm uses a result of Daligault and Thomassé [8] for MLST in acyclic digraphs. The 2-approximation algorithm is inspired by Lu and Ravi's 3-approximation algorithm for MLST in undirected graphs. Our result provides an example of a tree-based network design problem where acyclicity in digraphs can be exploited very well. Finally, we observed that MLST in acyclic digraphs is MaxSNP-hard.

References

1. Alon, N., Fomin, F.V., Gutin, G., Krivelevich, M., Saurabh, S.: Parameterized Algorithms for Directed Maximum Leaf Problems. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 352–362. Springer, Heidelberg (2007)
2. Alon, N., Fomin, F.V., Gutin, G., Krivelevich, M., Saurabh, S.: Spanning directed trees with many leaves. SIAM J. Discrete Math. 23(1), 466–476 (2009)
3. Binkele-Raible, D., Fernau, H.: A Faster Exact Algorithm for the Directed Maximum Leaf Spanning Tree Problem. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 328–339. Springer, Heidelberg (2010)
4. Bonsma, P.S., Zickfeld, F.: A 3/2-Approximation Algorithm for Finding Spanning Trees with Many Leaves in Cubic Graphs. SIAM J. Disc. Math. 25(4), 1652–1666 (2011)

5. Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: An improved LP-based approximation for Steiner tree. In: Proc. 42nd ACM Symp. Theory Comput. (STOC), pp. 583–592 (2010)
6. Charikar, M., Chekuri, C., Cheung, T.Y., Dai, Z., Goel, A., Guha, S., Li, M.: Approximation algorithms for directed Steiner problems. In: Proc. 9th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA), pp. 192–200 (1998)
7. Daligault, J., Gutin, G., Kim, E.J., Yeo, A.: FPT algorithms and kernels for the directed k-leaf problem. J. Comput. Syst. Sci. 76(2), 144–152 (2010)
8. Daligault, J., Thomassé, S.: On Finding Directed Trees with Many Leaves. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 86–97. Springer, Heidelberg (2009)
9. Drescher, M., Vetta, A.: An approximation algorithm for the maximum leaf spanning arborescence problem. ACM Trans. Algorithms 6(3), 1–18 (2010)
10. Galbiati, G., Maffioli, F., Morzenti, A.: A short note on the approximability of the maximum leaves spanning tree problem. Inform. Process. Lett. 52(1), 45–49 (1994)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., New York (1979)
12. Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM J. Discrete Math. 4(1), 99–106 (1991)
13. Lu, H.I., Ravi, R.: Approximating maximum leaf spanning trees in almost linear time. J. Algorithms 29(1), 132–141 (1998)
14. Solis-Oba, R.: 2-Approximation Algorithm for Finding a Spanning Tree with Maximum Number of Leaves. In: Bilardi, G., Italiano, G.F., Pietracaprina, A., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 441–452. Springer, Heidelberg (1998)
15. Zelikovsky, A.: A series of approximation algorithms for the acyclic directed Steiner tree problem. Algorithmica 18(1), 99–110 (1997)

Optimization over Integers with Robustness in Cost and Few Constraints

Kai-Simon Goetzmann[1,⋆], Sebastian Stiller[2,⋆⋆], and Claudio Telha[3]

[1] Institut für Mathematik, TU Berlin
goetzmann@math.tu-berlin.de
[2] Sloan School of Management, MIT
sebastia@mit.edu
[3] Operations Research Center, MIT
ctelha@mit.edu

Abstract. We consider robust counterparts of integer programs and combinatorial optimization problems (summarized as *integer problems* in the following), i.e., seek solutions that stay feasible if at most Γ-many parameters change within a given range. While there is an elaborate machinery for continuous robust optimization problems, results on robust integer problems are still rare and hardly general.

We show several optimization and approximation results for the robust (with respect to cost, or few constraints) counterpart of an integer problem under the condition that one can optimize or approximate the original integer problem with respect to a piecewise linear objective (respectively piecewise linear constraints).

For example, if there is a ρ-approximation for a minimization problem with non-negative costs and non-negative and bounded variables for piecewise linear objectives, then the cost robust counterpart can be $\rho(1+\varepsilon)$-approximated.

We demonstrate the applicability of our approach on two classes of integer programs, namely, totally unimodular integer programs and integer programs with two variables per inequality. Further, for combinatorial optimization problems our method yields polynomial time approximations and pseudopolynomial, exact algorithms for Robust Unbounded Knapsack Problems.

Keywords: Robust Optimization, Integer Programming, Total Unimodularity, Unbounded Knapsack, Integer Programs with two variables per inequality.

1 Introduction

A solution to an optimization problem often has to be good not just for one instance but for a set of scenarios. This can either be due to uncertainty as to which of the

⋆ Supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

⋆⋆ Marie-Curie Fellow of the European Commission under the ROSES-Project (FP7-PEOPLE-2009-IOF 254402).

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 89–101, 2011.

scenarios will eventually occur, or because the solution shall be used several times in different scenarios.

One solution concept for optimization over scenarios is Robust Optimization. In the robust paradigm feasibility and cost of a solution are measured by those scenarios in which the solution performs worst. This worst case approach contrasts to stochastic programming, where the cost of a solution is typically a weighted average over all scenarios, good ones and bad ones.

As an illustration, suppose we choose a route for regularly driving to work. We want to be on time no matter what happens, thus we have to evaluate each route by the travel time in its worst case scenario.

Robust optimization has thriven in the past decade, partly because its applicability became apparent, and partly because the resulting mathematical models allow for strong solution methods. For continuous problems a cohesive body of quite general methods has been developed. For combinatorial problems and integer linear programs (IPs) the picture is a lot more scattered. Typically, the results cover a specific combinatorial problem. This is of course a consequence of the richness of combinatorial optimization and integer linear programming. General results for all of these problems as in the continuous case are unlikely. Therefore the following result by Bertsimas and Sim is even more remarkable:

In [7], they show that for uncertain cost coefficients, where at most Γ of them can deviate from the nominal setting at the same time, solvability or approximability of any problem with binary decision variables extends to the robust case, as it suffices to solve a linear number of instances of the deterministic problem.

Bertsimas and Sim explicitly note that this result is intrinsically limited to binary variables. With the help of a new technique we get a corresponding result for integer, not necessarily binary cost robust problems[1]. Further, we can extend our method to general robust integer problems with uncertainty in one (or few) linear constraint(s). Restricting the latter result again to binary problems gives the exact sibling of the cost robust result in [7] for robustness in constraints. However, new insights were needed to translate the methods from [7] to the constraint robust setting.

Our Contribution: The main results of this paper are the following:

- The cost robust counterpart (in the same sense as in [7]) of an integer problem can be solved or approximated if the original problem can be solved for piecewise linear convex cost functions with at most two bends.
- To solve integer problems with uncertainty in a constant number of linear constraints, one has to solve a modified problem where the left hand sides of the constraints are replaced by piecewise linear convex functions.
- For *binary* problems with uncertainty in a constant number of linear constraints, it suffices to solve a polynomial number of instances of the original problem with slightly modified coefficients in the constraints.

[1] We speak of integer *problems* to integrate IPs and combinatorial problems, where the feasibility sets need not be given explicitly by linear constraints.

At first sight, the requirement of solvability for piecewise linear functions seems clumsy, and not likely to be useful. To the contrary, we exemplify its usefulness by a number of quite different and broad applications of our results. Those general results allow us to develop methods for cost robust counterparts of entire classes of integer linear programs, notably, totally unimodular programs (TUM) and integer programs with two variables per constraint (IP2). Both classes have been studied intensely in the deterministic case, but we are not aware of any general results on their robust counterparts. Our general result on cost robust TUM problems broadly extends results on Robust Min Cost Flows in [7]. Further, we apply our general results to a combinatorial problem, namely, Unbounded Knapsack. In this case we derive an algorithm that handles cost robustness and robustness in the constraints at the same time.

We believe that this paper will motivate the consideration of piecewise linear cost functions and constraints for further classes of integer problems.

Let us remark that although many optimization problems with a natural non-binary IP description can be reformulated as binary IPs, this does usually not yield a workaround to apply results for robust binary IPs to the naturally non-binary problem – even granted the incurred blow-up of the instance. The hindrance is usually that the scenario sets of the robust counterparts make no sense once the problem is transformed into an unnatural binary program.

Related Work: Modern continuous robust optimization started with [20] for convex uncertainty sets and [4,5,6] for ellipsoidal uncertainty sets. Still a good overview for the state of the art is [1]. The Γ-scenario setting from [7,8] has found frequent application, e.g. in [9,10,16].

Robust Knapsack has so far only been considered in the binary setting. While for Γ-scenarios for uncertain costs the result from [7] applies, for the case of general scenarios there is no approximation algorithm at all [2,21]. Klopfenstein and Nace [17,18] considered polyhedral aspects of the robust Knapsack Problem, and in the context of the chance-constraint version also a weight robust Knapsack Problem. For the latter problem they derive a pseudo-polynomial algorithm, a result that also follows from our general result on constraint robust problems.

For integer linear programs with two variables per inequality (IP2), Hochbaum et al. [13] and Bar-Yehuda and Rawitz [3] provide a pseudopolynomial time 2-approximation algorithm. In case the inequalities are restricted to be monotone, there are pseudopolynomial time exact algorithms [3,14]. All algorithms explicitly assume that the variables are bounded.

Our general result generalizes a result for cost robust binary IPs [7], and our results for totally unimodular integer programs generalize results on specific totally unimodular problems, e.g., Min Cost Flows [7].

Structure of the paper: In Section 2 and 3 we present the general results on cost robust and constraint robust integer problems, respectively. In Section 4 we apply them to problems with a totally unimodular description and integer programs with two variables per inequality, enabling us to solve the cost robust

counterparts, as well as to the Unbounded Knapsack Problem, where we can solve a robust version that features both, uncertainty in cost and weights.

2 Uncertainty in the Objective

We start with the general result on cost robust, not necessarily binary problems. We will use $[n]$ for the set $\{1,\dots,n\}$ here and throughout. Further, let us note that in our notation the set $\mathbb{N}$ includes the zero. By $\mathrm{T}_{\mathcal{A}}(I)$ we denote the running time of an algorithm $\mathcal{A}$ on an instance I.

The formal definition for the considered class of problems reads as follows:

Definition 1 (Cost Robust Optimization Problem). *For the optimization problems* $\min_{x\in X}\{c^{\mathrm{T}}x\}$ *and* $\max_{x\in X}\{c^{\mathrm{T}}x\}$*, given by* $P=(c,X)$ *where* $X\subseteq\mathbb{Z}^n$ *and* $c\in\mathbb{R}^n$*, and a non-negative integer vector* $d\in\mathbb{N}^n$ *together with* $\Gamma\in[n]$*, the* minimization (maximization) (d,Γ)-Cost Robust Counterpart (CRC) of P *is defined by*

$$\min_{x\in X}\left\{c^{\mathrm{T}}x+\max_{\substack{S\subseteq[n]\\|S|\le\Gamma}}\sum_{j\in S}|d_jx_j|\right\}\quad\text{and}\quad\max_{x\in X}\left\{c^{\mathrm{T}}x-\max_{\substack{S\subseteq[n]\\|S|\le\Gamma}}\sum_{j\in S}|d_jx_j|\right\},\tag{1}$$

respectively.

Our main goal is to show that one can solve or approximate the CRC of P, if one can solve or approximate the following variant of P:

Definition 2 (Modified Optimization Problem). *For the minimization (maximization) problem given by* $P=(X,c)$*,* $c'\in\mathbb{R}^n_{\ge0}$ *and* $\alpha\ge0$*, the* (c',α)-Modified Minimization (Maximization) Problem (MMin, MMax) of P *is*

$$\min_{x\in X}\Big\{\sum\nolimits_{j\in[n]}\widetilde{c}_j(x_j)\Big\}\qquad\text{and}\qquad\max_{x\in X}\Big\{\sum\nolimits_{j\in[n]}\widetilde{c}_j(x_j)\Big\},\tag{2}$$

where $\widetilde{c}_j(x_j):=c_jx_j\pm\max\{c'_jx_j-\alpha,0\}\pm\max\{-c'_jx_j-\alpha,0\}$ *for minimization ("+") and maximization ("−"), respectively.*

At this point minimization and maximization are fully symmetric. At a later stage it will come in handy to have them defined separately.

Theorem 3. *Consider the optimization problem of* $P=(c,X)$ *with* $X\subseteq\mathbb{Z}^n$ *and* $c\in\mathbb{R}^n$*. Suppose for some* $\rho\ge1$ *there is a* ρ*-approximation algorithm* $\mathcal{A}_1$ *for the* (c',α)*-MMin (MMax) of* P *and arbitrary* c' *and* α*. Further suppose there is an algorithm* $\mathcal{A}_2$ *that, for given* $d\in\mathbb{N}^n$ *and* $\Gamma\in[n]$*, computes upper bounds* u_j *on the absolute value of each variable* x_j *in the optimal solution of the* (d,Γ)*-CRC of* P*. Then there is a* ρ*-approximation algorithm* $\mathcal{A}$ *for the* (d,Γ)*-CRC of* P *with running time* $\mathrm{T}_{\mathcal{A}}(P,d,\Gamma)\in\mathcal{O}\big(\mathrm{T}_{\mathcal{A}_2}(P,d,\Gamma)+\overline{\vartheta}\cdot\mathrm{T}_{\mathcal{A}_1}(P,d,\overline{\vartheta})\big)$*, where* $\overline{\vartheta}:=\max_j\{u_jd_j\}$.

We will use $\overline{\vartheta}:=\max_j\{u_jd_j\}$ throughout the remainder of this paper without defining it again.

Note that for $\rho = 1$, i.e., if we have an exact algorithm for the modified problem, we can solve the CRC exactly.

Proof. We only consider minimization problems, since we can transform any maximization problem into a minimization problem by taking the negative of the costs. Along the lines of the binary result from [7], we formulate the inner maximization problem of (1) as an IP, dualize and eliminate all but one dual variable to get the following reformulation of (1):

$$\min_{x\in X,\vartheta\geq 0}\left\{c^{\mathrm{T}}x+\Gamma\vartheta+\sum_{j\in[n]}\left(\max\{d_jx_j-\vartheta,0\}+\max\{-d_jx_j-\vartheta,0\}\right)\right\}. \quad (3)$$

From this point on, the methods from [7] no longer apply because the variables are non-binary. We thus use our new technique, which utilizes the notion of the Modified Optimization Problem: For a fixed ϑ, (3) is equivalent to the (d,ϑ)-MMin of P. By the conditions of the Theorem, we can compute a ρ-approximate solution to this problem.

Let (x^*,ϑ^*) be an optimal solution to (3). We know that $|x_j^*| \leq u_j$, so if $\vartheta \geq \max_j u_jd_j = \overline{\vartheta}$, for all j both maxima in (3) vanish. Hence, if we increase ϑ beyond this number, the objective value increases. It follows that $\vartheta^* \leq \overline{\vartheta}$.

Also, we can assume that ϑ^* is integral: Denote by

$$C^*(\vartheta) := \Gamma\vartheta+\min_{x\in X}\left\{\sum_{j\in[n]}\left(c_jx_j+\max\{d_jx_j-\vartheta,0\}+\max\{-d_jx_j-\vartheta,0\}\right)\right\} \quad (4)$$

the optimal cost for a fixed ϑ. Since x and d are integral, this function is linear in ϑ within each interval $[k,k+1], k \in \mathbb{N}$. In such an interval the local maximum is obtained for $\vartheta = k$ or for $\vartheta = k+1$, and thus the global maximum is obtained for some integral ϑ.

We can thus compute all ρ-approximate solutions corresponding to integral values of ϑ in $[0,\overline{\vartheta}]$, and choose the best among them, resulting in the claimed running time. □

Remark. Our model of robustness limits to deviation in at most Γ cost coefficients. The resulting inner maximization problem, which we dualized in the previous proof, is totally unimodular. Therefore a standard argument originating from [7] gives that this model is equivalent to protecting against any cost function $c+\delta d$ with δ in the set $\{\delta \in \mathbb{R}^n : \sum_{j\in[n]} |\delta_j| \leq \Gamma\}$.

Unless $\max_j u_jd_j$ is polynomial in the input, in Theorem 3 one ends up with a pseudopolynomial algorithm for the CRC, even if a polynomial algorithm for the modified optimization problem is given. This can be overcome if $\rho = 1$ and C^* as defined in (4) is convex as a function of ϑ, in which case ϑ^* can be found via a carefully constructed binary search (similar to the one in proof of Theorem 7 in [7]):

Theorem 4. *Consider the minimization problem of $P = (c, X)$ with $X \subseteq \mathbb{Z}^n$ and $c \in \mathbb{R}^n$. If the conditions of Theorem 3 hold, and if $\rho = 1$ and C^* is a*

convex function, then there is an exact algorithm $\mathcal{A}$ for the (d,Γ)-CRC of P with running time $\mathrm{T}_{\mathcal{A}}(P,d,\Gamma) \in \mathcal{O}\big(\mathrm{T}_{\mathcal{A}_2}(P,d,\Gamma) + \log(\overline{\vartheta}) \cdot \mathrm{T}_{\mathcal{A}_1}(P,d,\overline{\vartheta})\big)$.

For an application of this result we refer the reader to the part on problems with totally unimodular description in Section 4.

When $\rho > 1$ or C^* is not convex, we can still restrict the number of calls of the oracle $\mathcal{A}_1$ to $\mathcal{O}(\log(\overline{\vartheta}))$ in exchange for a slightly weaker approximation guarantee. But for this result we have to consider minimization and maximization separately and restrict to combinatorial problems with non-negative cost coefficients and variables. Note that in this case the second maximum in both the definition of MMin and MMax vanishes.

It requires some additional non-trivial insights to prove that if ϑ^* is approximated, also the value of the solution will not deviate too much from the optimal value. We present these ideas in the following two proofs.

Theorem 5 (Minimization Problem). *Consider the minimization problem of $P = (c, X)$ with $X \subseteq \mathbb{N}^n$ and $c \in \mathbb{R}^n_{\geq 0}$. Under the conditions of Theorem 3, for all $\varepsilon > 0$ there is a $\rho(1+\varepsilon)$-approximation algorithm $\mathcal{A}$ for the (d,Γ)-CRC of P with running time* $\mathrm{T}_{\mathcal{A}}(P,d,\Gamma) \in \mathcal{O}\big(\mathrm{T}_{\mathcal{A}_2}(P,d,\Gamma) + \frac{1}{\varepsilon} \cdot \log(\overline{\vartheta}) \cdot \mathrm{T}_{\mathcal{A}_1}(P,d,\overline{\vartheta})\big)$.

Proof. We start as in the proof of Theorem 3. To attain the claimed running time, however, for any given $\varepsilon > 0$, we now solve (4) approximately for all $\vartheta \in \{0\} \cup \{(1+\varepsilon)^k : k \in \mathbb{N}, (1+\varepsilon)^{k-1} \leq \overline{\vartheta}\}$, and return the best of all these solutions. This yields a $\rho(1+\varepsilon)$-approximation for the CRC:

Let (x^*, ϑ^*) be an optimal solution to (3), w.l.o.g. $\vartheta^* \leq \overline{\vartheta}$ and $\vartheta^* \in \mathbb{N}$. In case $\vartheta^* \in \{0,1\}$, our solution is within a factor of ρ of the optimum, since these two values for ϑ are checked. Otherwise, let $k_0 \in \mathbb{N} \setminus \{0\}$ be such that $(1+\varepsilon)^{k_0-1} < \vartheta^* \leq (1+\varepsilon)^{k_0} =: \vartheta_0$. Since Γ, ϑ^*, c, and $x \geq 0$, we get

$$\frac{C^*(\vartheta_0)}{C^*(\vartheta^*)} \leq \max\left\{ \frac{\Gamma\vartheta_0}{\Gamma\vartheta^*}\,,\ \frac{\min_{x\in X}\left\{\sum_j c_j x_j + \max\{d_j x_j - \vartheta_0, 0\}\right\}}{\min_{x\in X}\left\{\sum_j c_j x_j + \max\{d_j x_j - \vartheta^*, 0\}\right\}} \right\} \leq 1+\varepsilon\,.$$

Since we can compute ρ-approximations to $C^*(\vartheta)$, the best solution we find is a $\rho(1+\varepsilon)$-approximation for the CRC. Further, the oracle $\mathcal{A}_1$ is called $\mathcal{O}(\log_{(1+\varepsilon)} \overline{\vartheta}) = \mathcal{O}\big(\frac{1}{\varepsilon} \cdot \log(\overline{\vartheta})\big)$ times , resulting in the claimed running time. □

For maximization, the perturbed cost in a worst scenario can be relatively close to zero, while all numbers involved are rather large. This, roughly speaking, spoils an approximation result for maximization similar to Theorem 5 – unless we impose a further condition:

Theorem 6 (Maximization Problems). *Consider the maximization problem of $P = (c, X)$ with $X \subseteq \mathbb{N}^n$ and $c \in \mathbb{R}^n_{\geq 0}$. Suppose the conditions of Theorem 3 hold, and suppose that the relative cost decrease in the (d,Γ)-CRC of P is bounded from above by a constant $\beta < 1$, i.e.:*

$$\exists\, \beta < 1 : \qquad \frac{d_j}{c_j} \leq \beta \quad \forall\, j \in [n]\,.$$

Then there is a 2ρ-approximation algorithm $\mathcal{A}$ for the (d,Γ)-CRC of P with running time $\mathrm{T}_{\mathcal{A}}(P,d,\Gamma) \in \mathcal{O}\big(\mathrm{T}_{\mathcal{A}_2}(P,d,\Gamma) + \log(\overline{\vartheta}) \cdot \mathrm{T}_{\mathcal{A}_1}(P,d,\overline{\vartheta})\big)$.

Proof. As in the proof of Theorem 5, we solve the MMax of P for $\vartheta = (1+\varepsilon)^k$ for some $k \in \mathbb{N}$ and a particular $\varepsilon > 0$. For the choice of ε, consider an optimal solution (x^*, ϑ^*) with value OPT. We get that

$$\Gamma\vartheta^* \leq \Gamma\vartheta^* + \sum_{j\in[n]} \max\{d_j x_j^* - \vartheta^*, 0\} = \underbrace{c^{\mathrm{T}}x^* - \mathrm{OPT}}_{(1)} \overset{(*)}{\leq} d^{\mathrm{T}}x^* \leq \beta c^{\mathrm{T}}x^* \ ,$$

where $(*)$ holds because (1) is the cost we lose due to the decrease of some of the coefficients, and this cost is bounded by $d^{\mathrm{T}}x^*$.

We now set $\varepsilon := {}^{(1-\beta)}/_{2\beta}$ (w.l.o.g. $\beta > 0$). Then

$$\mathrm{OPT} \geq (c-d)^{\mathrm{T}}x^* \geq (1-\beta)c^{\mathrm{T}}x^* = 2\varepsilon\beta c^{\mathrm{T}}x^* \geq 2\varepsilon\Gamma\vartheta^* \ .$$

With this, we can bound the error that arises from approximating ϑ^*:

Denote by $C^*(\vartheta) := -\Gamma\vartheta + \max_{x\in X}\left\{\sum_{j\in[n]} c_j x_j - \max\{d_j x_j - \vartheta, 0\}\right\}$ the optimal cost for a fixed ϑ. With ϑ_0 as in the proof of Theorem 5 we then get

$$\begin{aligned}\frac{\mathrm{OPT}}{C^*(\vartheta_0)} &\leq \frac{\mathrm{OPT}}{-\Gamma\vartheta_0 + \max_{x\in X}\left\{\sum_{j\in[n]}\left(c_j x_j - \max\{d_j x_j - \vartheta^*, 0\}\right)\right\}} \\ &= \frac{\mathrm{OPT}}{-\Gamma\vartheta_0 + \mathrm{OPT} + \Gamma\vartheta^*} \leq \frac{\mathrm{OPT} - \varepsilon\Gamma\vartheta^* + \varepsilon\Gamma\vartheta^*}{-\Gamma(1+\varepsilon)\vartheta^* + \mathrm{OPT} + \Gamma\vartheta^*} \\ &= 1 + \frac{\varepsilon\Gamma\vartheta^*}{\mathrm{OPT} - \varepsilon\Gamma\vartheta^*} \leq 2 \ .\end{aligned}$$

Since we are able to approximate the optimal solution to the MMax of P within a factor of ρ, the considerations above prove that our algorithm yields a 2ρ-approximation. The number of calls of $\mathcal{A}_1$ is the same as in the proof of Theorem 5. Since ε is constant, we get the claimed overall running time. □

3 Uncertainty in Constraints

We now turn to the case where the coefficients of a single linear constraint (or those of a constant number of them) are uncertain. In the setting considered here minimization and maximization are equivalent, so we restrict to one of the two. The formal definition of the considered class of problems is as follows:

Definition 7 (Constraint Robust Maximization Problem). *Consider the problem* $\max_{x\in X}\{c^{\mathrm{T}}x\}$*, given by* $P = (c, X)$ *where* $c \in \mathbb{R}^n$ *and* $X = \{x \in X' : a^{\mathrm{T}}x \leq r\}$ *for some* $X' \subseteq \mathbb{Z}^n, a \in \mathbb{R}^n, r \in \mathbb{R}$*. For a non-negative integer vector* $b \in \mathbb{N}^n$ *together with* $\Gamma \in [n]$*, the* (b, Γ)-Constraint Robust Counterpart (ConsRC) of P *is defined by*

$$\max c^{\mathrm{T}}x \qquad s.t. \quad x \in X', \quad a^{\mathrm{T}}x + \max_{\substack{S\subseteq[n]\\|S|\leq\Gamma}} \sum_{j\in S} |b_j x_j| \leq r. \tag{5}$$

As in the cost robust setting, the left hand side of the constraint with uncertain coefficients can be transformed into a sum of piecewise linear convex function with two bends:

Lemma 8. *The (b,Γ)-Constraint Robust Counterpart of the maximization problem $P=(c,X)$ as defined in Definition 7 is equivalent to*

$$\max_{\xi\geq 0}\max_{x\in X(\xi)} c^{\mathrm{T}}x, \qquad \text{with} \tag{6}$$

$$X(\xi) := \Big\{x\in X' \colon \Gamma\xi + \sum_{j\in[n]} \big(a_jx_j + \max\{b_jx_j-\xi,0\} + \max\{-b_jx_j-\xi,0\}\big) \leq r\Big\}.$$

Proof. With the same transformations as in the cost robust setting, we get that (5) is equivalent to

$$\max c^{\mathrm{T}}x \qquad \text{s.t.} \quad x\in X' \quad \text{and}$$
$$\min_{\xi\geq 0}\Big\{\Gamma\xi + \sum\nolimits_j a_jx_j + \max\{b_jx_j-\xi,0\} + \max\{-b_jx_j-\xi,0\}\Big\} \leq r.$$

Thus, for all feasible solutions x of (5) there exists some $\xi(x)\geq 0$ such that $x\in X(\xi(x))$. Consequently, (5) is equivalent to $\max_{\xi\geq 0}\max_{x\in X(\xi)} c^{\mathrm{T}}x$. □

For the non-binary case, the optimal ξ^* can be found by enumeration, since it is integral and bounded by the maximum deviation in the constraint coefficients:

Corollary 9. *Consider the (b,Γ)-ConsRC of the maximization problem $P=(c,X)$ as defined in Definition 7. Suppose there is an algorithm $\mathcal{A}_1$ computing a ρ-approximation for $\max_{x\in X(\xi)} c^{\mathrm{T}}x$ for any $\xi\geq 0$, and an algorithm $\mathcal{A}_2$ that computes upper bounds u_j on the absolute value of each variable x_j in the optimal solution of (5). Then there is a ρ-approximation algorithm $\mathcal{A}$ for the (b,Γ)-ConsRC of P with running time $\mathrm{T}_{\mathcal{A}}(P,b,\Gamma) = \mathcal{O}\big(\mathrm{T}_{\mathcal{A}_2}(P,b,\Gamma) + \overline{\xi}\cdot\mathrm{T}_{\mathcal{A}_1}(P,b,\Gamma,\overline{\xi})\big)$, where $\overline{\xi} := \max_j\{u_jb_j\}$.*

If all variables are binary, i.e. $X'\subseteq\{0,1\}^n$, there are only $n+1$ possibilities for ξ^*, and for a fixed ξ the constraint of problem (6) becomes linear again. Hence, to solve the (b,Γ)-ConsRC of $P=(c,X)$, it suffices to solve $n+1$ problems of the type of P for slightly different coefficients in the linear constraint.

This result is an exact sibling of the result on cost robust binary problems in [7], but it requires some new insights to translate the methods from [7] to the constraint robust setting.

Theorem 10. *If $X'\subseteq\{0,1\}^n$, the (b,Γ)-ConsRC of the maximization problem $P=(c,X)$ as defined in Definition 7 is equivalent to*

$$\max_{\ell=1,\dots,n+1}\left(\max c^{\mathrm{T}}x \qquad \text{s.t.} \quad x\in X', \quad \Gamma b_\ell + a^{\mathrm{T}}x + \sum_{j=1}^{\ell-1}(b_j-b_\ell)x_j \leq r\right),$$

whereby w.l.o.g. we assume $b_n\leq b_{n-1}\leq\ldots\leq b_1$ and define $b_{n+1}:=0$.

Proof. We know that $\xi^*\in[0,b_1]$. We split up this interval at $b_\ell, \ell=n,\dots,2$, and maximize over each subinterval, i.e. we reformulate (6) to get

$$\max_{\ell=1,\dots,n}\Big(\max_{\xi\in[b_{\ell+1},b_\ell]}\Big(\max_{x\in X(\xi)} c^{\mathrm{T}}x\Big)\Big). \tag{7}$$

For $x \in \{0,1\}^n$ we have $\max\{b_j x_j - \xi, 0\} = \max\{b_j - \xi, 0\} x_j$, and thus for $\xi \in [b_{\ell+1}, b_\ell]$

$$\begin{aligned} X(\xi) &= \Big\{ x \in X' : \Gamma\xi + \sum_{j\in[n]} \big(a_j x_j + \max\{b_j - \xi, 0\} x_j\big) \le r \Big\} \\ &= \Big\{ x \in X' : \Gamma\xi + a^{\mathrm{T}} x + \sum_{j=1}^{\ell} (b_j - \xi) x_j \le r \Big\}. \end{aligned} \tag{8}$$

For any fixed x, the left hand side of the constraint in (8) is a linear function in ξ that has to be no greater than r somewhere in $[b_{\ell+1}, b_\ell]$ for x to be feasible. Thus, if the constraint is satisfied for any ξ in this interval, because of linearity it will be satisfied for at least one of the values $\xi = b_{\ell+1}$ or $\xi = b_\ell$. As a consequence,

$$\max_{\xi\in[b_{\ell+1},b_\ell]} \Big(\max_{x\in X(\xi)} c^{\mathrm{T}} x \Big) = \max_{\xi=b_{\ell+1},b_\ell} \Big(\max_{x\in X(\xi)} c^{\mathrm{T}} x \Big). \tag{9}$$

Combining (7)–(9) yields the claimed result. □

As a corollary from Theorem 10 we get the existence of a pseudopolynomial exact algorithm as well as an FPTAS for the weight robust counterpart of the binary Knapsack Problem, generalizing a result from [18].

All the results from this section hold as well if there is a constant number k of constraints with uncertain coefficients. The problem $\max_{x\in X(\xi)} c^{\mathrm{T}} x$ would then have to be solved $(\max_j\{u_j b_j\})^k$ times in the setting of Corollary 9 and $(n+1)^k$ times in the binary case.

4 Applications

The final section is devoted to applications of the general results presented above. We first consider the cost robust setting for problems with a totally unimodular description and IPs with two variables per inequality, and then study the Unbounded Knapsack Problem, both with uncertain weights and cost, integrating our general results.

Problems with Totally Unimodular Description. The concept of totally unimodular matrices is arguably the most successful concept for solving a large class of integer programs. In general, robust counterparts need not inherit total unimodularity. We show that in our setting, however, the CRC of P can be solved exactly for those problems where the solution space of P can be described by a totally unimodular matrix of size polynomial in the size of the input of P.

This generalizes results on specific totally unimodular problems. In particular, it broadly generalizes the results on Robust Network Flows in [7], since the Min Cost Flow Problem is totally unimodular.

In this section we do not require non-negativity of the cost vector, so the minimization results we show can be used for maximization problems as well. We do require non-negative variables. This condition can be lifted, but this yields much less readable results that rest on similar arguments.

Definition 11. *A minimization problem $P = (c, X)$ is said to have a* bounded TUM description (A, b, u) *if the set of feasible solutions $X \subseteq \mathbb{N}^n$ is described by a totally unimodular matrix $A \in \mathbb{R}^{m \times n}$, an integral right-hand-side b, and an integral vector of upper bounds u, i.e.*

$$\operatorname{conv}(X) = \{x \in \mathbb{R}^n : Ax \leq b, x \leq u\}, \qquad A \text{ TUM}, b \in \mathbb{Z}^m, u \in \mathbb{Z}^n.$$

To apply Theorem 4 to solve problems of this kind, we need to establish the following two lemmas:

Lemma 12. *If the minimization problem $P = (c, X)$ is given by a bounded TUM description (A, b, u), then the Modified Minimization Problem can be solved in polynomial time.*

Lemma 13. *Let $C^*(\vartheta)$ be defined as in (4). Then for a minimization problem $P = (c, X)$ with a bounded TUM description (A, b, u), C^* is convex for any $c \in \mathbb{R}^n, d \in \mathbb{N}^n, \Gamma \in [n]$.*

The key idea is to split up each variable into three to model the piecewise linear cost function, and to observe that the resulting LP is still totally unimodular. For details we refer the reader to the technical report [11].

With Theorem 4 and the two lemmas, we get that we can solve the CRC of any problem with a bounded totally unimodular description in polynomial time:

Theorem 14. *If the minimization problem $P = (c, X)$ is given by a bounded TUM description (A, b, u), then for any $c \in \mathbb{R}^n, d \in \mathbb{N}^n$ and $\Gamma \in [n]$, there is an exact algorithm for the (d, Γ)-CRC of P that runs in polynomial time.*

Integer Programs with Two Variables per Inequality. We now apply our main results to a second, large, and intensely studied class of integer programs, namely integer programs with two variables per inequality (IP2).

Definition 15 (Integer Programs with Two Variables per Inequality). *A* bounded integer program with two variables per inequality *(bounded IP2) is a system of the form*

$$\min \{c^{\mathrm{T}} x \colon a_i^{\mathrm{T}} x \geq b_i \text{ for } i = 1, \ldots, m,\ \ell \leq x \leq u,\ x \text{ integer}\},$$

where $b \in \mathbb{Z}^m$, $\ell, u \in \mathbb{Z}^n$, $c \in \mathbb{Q}^n$ and each vector $a_i \in \mathbb{Z}^n$ has two non-zero components.

A bounded IP2 is called monotone *if the non-zero coefficients of a_i have opposite signs.*

The conditions required in Section 2 allow to intensely use the existing techniques for non-robust IP2, in particular [13], [14] and [3]. We obtain a pseudopolynomial time 2-approximation for the CRC of bounded IP2s and an exact pseudopolynomial time algorithm for the CRC of bounded, monotone IP2s.

Theorem 16. *The cost robust counterpart of a bounded monotone IP2 can be solved in pseudopolynomial time.*

Remark. Theorem 16 can be established by extending (to handle piecewise linear functions) the pseudopolynomial time algorithm of Hochbaum and Naor [14] for bounded monotone IP2, cf. [11]. In [3], Bar-Yehuda and Rawitz give an exact pseudopolynomial algorithm for monotone cost functions but non-negative lower bounds.

Theorem 17. *There is a pseudopolynomial time 2-approximation algorithm for the cost robust counterpart of a bounded IP2 with non-negative coefficients in the objective function and non-negative lower bounds.*

Remark. Theorem 17 is proven by extending (to handle piecewise linear functions) the pseudopolynomial 2-approximation algorithm of Hochbaum et al. [13], cf. [11]. This result is also shown in [3].

Robust Unbounded Knapsack Problems. To demonstrate how versatile our main results are for combinatorial problems, we apply them to the *Unbounded Knapsack Problem*, the non-binary extension of the classical Knapsack Problem (KP). For this problem we will be able to handle counterparts that feature both, cost robustness and robustness in the constraint.

Definition 18 (Unbounded Knapsack Problem). *An instance of the* Unbounded Knapsack Problem *(UKP) is given by a knapsack capacity $W \geq 0$ and n types of items with weights $w_j \in \mathbb{N}$ and costs $c_j \in \mathbb{R}_{\geq 0}$, $j \in [n]$. The task is to find a vector $x \in \mathbb{N}^n$ with $\sum_j w_j x_j \leq W$ maximizing the cost $\sum_j c_j x_j$.*

UKP and its extensions, in particular its robust counterparts, are *NP*-hard. Intuitively, UKP seems to be more complex than the binary KP, since the input is more compact. Still, as for KP, there is both a pseudopolynomial Dynamic Program (DP) and an FPTAS [19,15].

We now consider the robust versions of the Unbounded Knapsack Problem.

Cost Robust UKP (CRUKP). While the result for binary cost robust programs [7] can be applied to the standard Knapsack Problem, the CRC of UKP surpasses the reach of [7]. As argued earlier, a reformulation as a binary integer program does not only cause a blow-up in size, but it also renders the scenario set meaningless. Thus, to solve CRUKP, we need to be able to solve UKP for piecewise linear concave cost functions. In [12], Hochbaum presented an FPTAS for this problem. We give an alternative FPTAS based on a dynamic program (DP) in our technical report [11]. With these results, by Theorem 6 it follows that for all $\varepsilon > 0$, there is a $(2 + \varepsilon)$-approximation algorithm for CRUKP, if the relative cost decrease is bounded away from 1 by a constant. On the other hand, using Theorem 3 with the DP from [11], we get an exact algorithm with pseudopolynomial running time.

Weight Robust UKP (WRUKP). Next we turn to the Unbounded Knapsack Problem where weights instead of costs are uncertain. In terms of Section 3 we have uncertainty in the only constraint. We consider the $(\Delta w, \Gamma)$-ConsRC of UKP, where Δw_j denotes the possible increase in weight of items of type j. From Corollary 9 we learn that we have to solve $\max_{x \in X(\xi)} c^{\mathrm{T}} x$ in order to get a pseudopolynomial algorithm for WRUKP. The FPTAS from [12] could be used for this. Alternatively, we can compute an exact solution in pseudopolynomial time by the DP described in [11]. With $u_j = \frac{W}{w_j}$, this yields an exact algorithm for WRUKP with running time $\mathcal{O}(\max_j \frac{\Delta w_j}{w_j} \cdot n^2 W^2)$.

General Robust UKP (RUKP). Finally, we consider a version of UKP where both weights and costs are uncertain. At most Γ_w types of items can increase their weight, and at most Γ_c cost coefficients decrease. This is the $(\Delta w, \Gamma_w)$-ConsRC of CRUKP. Since the DP from [11] works for concave cost and convex weight functions, by Theorem 3 we get an exact algorithm $\mathcal{A}_1$ for CRUKP on the modified solution space $X(\xi)$ with a running time of $\mathcal{O}(\max_j \frac{d_j}{w_j} \cdot n^2 W^2)$, and can thus solve RUKP exactly in a running time of $\mathcal{O}(\max_j \frac{\Delta w_j}{w_j} \cdot \max_j \frac{d_j}{w_j} \cdot n^2 W^3)$.

Acknowledgement. We are grateful to Martin Skutella and Günter Rote for discussions that substantially enhanced this paper.

References

1. Special issue on robust optimization. Math. Program. 107(1-2) (2006)
2. Aissi, H., Bazgan, C., Vanderpooten, D.: Approximation of min-max and min-max regret versions of some combinatorial optimization problems. Europ. J. of Oper. Res. 179(2), 281–290 (2007)
3. Bar-Yehuda, R., Rawitz, D.: Efficient algorithms for integer programs with two variables per constraint 1. Algorithmica 29(4), 595–609 (2001)
4. Ben-Tal, A., Nemirovski, A.: Robust convex optimization. Math. Oper. Res. 23(4), 769–805 (1998)
5. Ben-Tal, A., Nemirovski, A.: Robust solutions to uncertain linear programs. Oper. Res. Letters 25(1), 1–13 (1999)
6. Ben-Tal, A., Nemirovski, A.: Robust solutions of linear programming problems contaminated with uncertain data. Math. Program. 88(3), 411–424 (2000)
7. Bertsimas, D., Sim, M.: Robust discrete optimization and network flows. Math. Program. 98(1-3), 49–71 (2003)
8. Bertsimas, D., Sim, M.: The price of robustness. Oper. Res. 52(1), 35–53 (2004)
9. Feige, U., Jain, K., Mahdian, M., Mirrokni, V.: Robust Combinatorial Optimization with Exponential Scenarios. In: Fischetti, M., Williamson, D.P. (eds.) IPCO 2007. LNCS, vol. 4513, pp. 439–453. Springer, Heidelberg (2007)
10. Fischetti, M., Monaci, M.: Light Robustness. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 61–84. Springer, Heidelberg (2009)
11. Goetzmann, K.-S., Stiller, S., Telha, C.: Optimization over integers with robustness in cost and few constraints. Technical Report 009-2011, Technische Universität Berlin (2011)

12. Hochbaum, D.: A nonlinear knapsack problem. Oper. Res. Lett. 17, 103–110 (1995)
13. Hochbaum, D., Megiddo, N., Naor, J., Tamir, A.: Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. Math. Program. 62(1), 69–83 (1993)
14. Hochbaum, D., Naor, J.: Simple and fast algorithms for linear and integer programs with two variables per inequality. SIAM J. Comput. 23, 1179–1192 (1994)
15. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM 22, 463–468 (1975)
16. Khandekar, R., Kortsarz, G., Mirrokni, V., Salavatipour, M.R.: Two-Stage Robust Network Design with Exponential Scenarios. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 589–600. Springer, Heidelberg (2008)
17. Klopfenstein, O., Nace, D.: A note on polyhedral aspects of a robust knapsack problem (2007), http://www.optimization-online.org
18. Klopfenstein, O., Nace, D.: A robust approach to the chance-constrained knapsack problem. Oper. Res. Letters 36(5), 628–632 (2008)
19. Martello, S., Toth, P.: Knapsack Problems. Algorithms and Computer Implementations. John Wiley and Sons (1990)
20. Soyster, A.L.: Convex programming with set-inclusive constraints and applications to inexact linear programming. Oper. Res. 21(5), 1154–1157 (1973)
21. Yu, G.: On the max-min 0-1 knapsack problem with robust optimization applications. Oper. Res. 44(2), 407–415 (1996)

A Lower Bound on Deterministic Online Algorithms for Scheduling on Related Machines without Preemption

Tomáš Ebenlendr[1] and Jiří Sgall[2]

[1] Institute of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic
ebik@math.cas.cz
[2] Dept. of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, CZ-11800 Praha 1, Czech Republic
sgall@kam.mff.cuni.cz

Abstract. We prove a new lower bound of 2.564 on deterministic online algorithms for makespan scheduling on related machines (without preemptions). Previous lower bound was 2.438 by Berman et al. We use an analytical bound on maximal frequency of scheduling jobs instead of the combinatorial bound obtained by computer based search through the graph of possible states of an algorithm in the previous work.

1 Introduction

We consider *one-by-one* online scheduling on uniformly related machines. The speed of machine M_i is denoted s_i. Each job is characterized by its processing time p_j takes p_j/s_i time to process on M_i. No preemptions are allowed, i.e., once the job is started it cannot be interrupted and the machine is busy with this job until the job is processed. The objective is to minimize the *makespan* (also called the length of the schedule, or the maximal completion time). The online algorithm sees only the next job from the input sequence and it has to schedule this job before it is given the following job. Note that in this model it is not necessary to specify the starting times of jobs, as any schedule can be trivially converted to the schedule without idle time (gaps), while not increasing the makespan. (Accordingly, this model is often considered as a variant of load balancing.)

We prove a new lower bound of 2.564 for the above-described problem, i.e., for deterministic online algorithms for makespan scheduling on related machines without preemptions. The previous lower bound was 2.438 by Berman et al [5]. They use combinatorial approach with computer based search through the graph of possible states of an algorithm. In contrast, we use an analytical bound on maximal frequency of scheduling jobs.

Our lower bound is based on an instance where both the machine speeds and the processing times are a geometric sequence of machines, with both sequences having the same common ratio, similarly as in [5,10]. In the previous bounds for similar problems one usually argues about the total amount of work done by

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 102–108, 2012.

the machines. In contrast, our bound is based on reasoning about the number of jobs scheduled and the frequency of scheduling jobs on every machine. First, we consider how the algorithm behaves on one of the machines and we upper bound the frequency of scheduling a job on this machine. This bound is a function of the competitive ratio, the common ratio of the geometric sequence, and the speed of the machine. Then we take the sum of these bounds on frequencies over all machines. Any online algorithm has to schedule one job in one step, thus this sum has to be at least 1. Finally, we let the common ratio of the geometric sequence to approach 1, and obtain our lower bound. This yields a certain inequality for the competitive ratio which we solve numerically.

Related Work

Naturally, the lower bounds need to be compared to the existing algorithms. The first constant-competitive algorithm for non-preemptive scheduling on related machines was developed in [1]. The currently best algorithms are $3 + \sqrt{8} \approx 5.828$ competitive deterministic and 4.311 competitive randomized one [5]. For an alternative very nice presentation see [3]. All these algorithms use doubling, i.e., strategies that estimate the optimal makespan by a geometric sequence. While this is a standard technique for obtaining a constant competitive ratio, it would be surprising if it led to optimal algorithms. The lower bound for randomized algorithms is 2, see [10]. Thus, both in the deterministic and randomized cases, significant gaps remain.

For a small number of machines the best known algorithm is the greedy List Scheduling (even though for many machines it is not even constant-competitive). Here List Scheduling is defined so that the next job is always scheduled so that it will finish as early as possible. The exact competitive ratio for $m = 2$ is ϕ and for $3 \leq m \leq 6$ it is equal to $1 + \sqrt{(m-1)/2}$ [6]; moreover for $m = 2, 3$ it can be checked easily that there is no better deterministic algorithm. For $m = 2$ it is possible even to give the exact optimal ratio for any speed combination, see [9]. The previous lower bound of 2.438 works for $m = 9$; for a smaller number of machines, no lower bound is known, except for the bound of 2 that follows from the analysis of List Scheduling for $m = 3$.

Interestingly, for the related problem where preemptions are allowed, we are able to provide an optimal online algorithm for any combination of the speeds and its competitive ratio is between 2.112 and $e \approx 2.718$, see [8,7]. Similar results seem to be out of reach for non-preemptive scheduling, as the combinatorial structure is much more difficult and the value of the optimum is NP-hard to compute, while for the preemptive scheduling it is computable, in fact given by an easy formula.

The problem of non-preemptive scheduling can be formulated in the language of online load balancing as the case where the jobs are permanent and the load is their only parameter corresponding to our processing time. Consequently, there are many results on load balancing that extend the basic results on online scheduling in a different direction, see e.g. [2].

Notations

We number the machines as well as the jobs from 0 (to obtain simpler formulas). Thus we have machines $M_0, M_1, \ldots, M_{m-1}$ and jobs $\mathcal{J} = (J_0, J_1, \ldots, J_{n-1})$. We use $\mathcal{J}[j] = (J_0, J_1, \ldots, J_j)$ to denote the input sequence of jobs cut off after J_j.

Let $\mathcal{J}_i$ be the set of jobs scheduled on machine M_i. The completion time of the machine is then simply the sum of processing times of the jobs scheduled to the machine divided by its speed: $C_i = \frac{1}{s_i} \sum_{j:J_j \in \mathcal{J}_i} p_j$. We compare the maximum completion time in the output of the algorithm with maximum completion time of the optimal schedule.

2 Lower Bound

Our lower bound is proved by an instance with a geometric sequence of machines, $s_i = \alpha^{-i}$, and a geometric sequence of jobs, $p_i = \alpha^i$, for some $\alpha > 1$. Both sequences have the same length, i.e., $n = m$. The optimal schedule after step t is to schedule the jobs on the machines in the reverse order, i.e., the J_j on machine M_{t-j}. The optimal makespan is thus equal to the size of the largest job, $C^*_{\max}(\mathcal{J}[t]) = p_t = \alpha^t$.

To achieve the competitive ratio of R, the algorithm has to complete the job t before time $R \cdot C^*_{\max}(\mathcal{J}[t])$. It follows immediately that it cannot schedule any job at any machine with speed below $1/R$. Furthermore, if the speed of a machine is only slightly above $1/R$, the jobs cannot be scheduled on it very often. Intuitively, the faster machines can schedule a job more frequently. We calculate the maximal frequency of scheduling a job for each machine separately, depending on its speed and R. The lower bound will follow from the fact that the sum of the frequencies has to be at least 1 so that the algorithm schedules all the jobs.

Following this scheme of the proof has some technical difficulties. In particular, the notion of frequency is not clear: The algorithm may schedule nothing on a machine for some time and then several jobs in a row. We need to think in a certain amortized way. Instead of formalizing the notion of amortized frequency, we formulate the bounds in terms of the number of jobs.

The following main lemma gives the bound for a single machine. The number t_i can be interpreted as the highest possible amortized frequency of scheduling a job to machine M_i with respect to the claimed competitive ratio R.

Lemma 1. *Let* A *be an R-competitive algorithm. Consider the instance described above. Let*

$$t_i = \log_\alpha \frac{R}{R - \alpha^i} \qquad \text{for } s_i = \alpha^{-i} > R^{-1} \tag{1}$$

Then, for any fixed $\alpha > 1$ and n, the algorithm A *schedules at most $\frac{n}{t_i} + R + 1$ jobs from the input sequence on machine M_i. Moreover the algorithm schedules at most one job on the machine with speed equal to $1/R$ (if there is any) and no job on any slower machine.*

Proof. If $s_i < 1/R$, then no job can be scheduled the machine M_i, since if the sequence would end now, the optimal makespan would be equal to the size of the last job on input, i.e, $C^*_{\max}(\mathcal{J}[t]) = p_t$. Moreover if $s_i = 1/R$ then only one job can be scheduled on M_i: The same argument now shows that no job is scheduled on M_i before scheduling any job. Thus we assume $s_i > 1/R$ from now on.

Let $p'_1, p'_2, \ldots, p'_{n_i}$ be (the processing times of) the jobs in $\mathcal{J}_i$ (i.e., those scheduled on the machine M_i). Intuitively, to schedule as many jobs as possible, it is best to schedule greedily the smallest possible jobs. We proceed to obtain a lower bound q_i on p'_i, the size of the $i$the job on the machine.

We can bound p'_j by $\sum_{k=1}^{j} p'_k \le Rs_i p'_j$ because the algorithm is R-competitive and the optimal makespan is p'_j after scheduling this job as the last one. In addition, $p'_j \ge 1$ for all jobs. This yields:

$$p'_j \;\ge\; \max\left\{\frac{\sum_{k=1}^{j-1} p'_k}{Rs_i - 1}, 1\right\} \quad \text{for } j = 1, 2, \ldots, n_i\,. \tag{2}$$

We define a sequence $(q_j)_{j=1}^{n_i}$ of lower bounds on the processing times from $\mathcal{J}_i$ recursively by taking equality in the expression above:

$$q_j \;=\; \max\left\{\frac{\sum_{k=1}^{j-1} q_k}{Rs_i - 1}, 1\right\} \quad \text{for } j = 1, 2, \ldots, n_i\,. \tag{3}$$

We can show that $q_j \le p'_j$ by induction on j: We have $q_1 = 1 \le p'_1$ by definition. To bound p'_{j+1}, note that $\sum_{k=1}^{j} q_k \le \sum_{k=1}^{j} p'_k$ using inductive assumption and plug in (2) and (3) for $j+1$:

$$q_{j+1} = \max\left\{\frac{\sum_{k=1}^{j} q_k}{Rs_i - 1}, 1\right\} \le \max\left\{\frac{\sum_{k=1}^{j} p'_k}{Rs_i - 1}, 1\right\} \le p'_{j+1}.$$

If $q_j > 1$ then by the definition of q_j we have

$$q_{j+1} = \frac{q_j + \sum_{k=1}^{j-1} q_k}{Rs_i - 1} = \frac{q_j}{Rs_i - 1} + \frac{\sum_{k=1}^{j-1} q_k}{Rs_i - 1} = \frac{q_j}{Rs_i - 1} + q_j = q_j \frac{Rs_i}{Rs_i - 1} \tag{4}$$

and thus

$$\log_\alpha \frac{q_{j+1}}{q_j} \;=\; \log_\alpha \frac{Rs_i}{Rs_i - 1} \;=\; t_i\,. \tag{5}$$

Now we bound n_i. We know that $q_{n_i} \le p'_{n_i} \le p_n = \alpha^n$ and thus $\log_\alpha q_{n_i} \le n$. Using (5) we have at most $n/t_i + 1$ numbers of size $q_j > 1$ in $q_1, \ldots, q_{n_i}$. We also have that $\sum_{j=1}^{\lfloor R \rfloor} q_j \ge \lfloor R \rfloor > R - 1 \ge Rs_i - 1$, thus $q_j > 1$ for any $j > R$. This gives that there are no more than $n/t_i + R + 1$ jobs scheduled to the machine M_i by an R-competitive algorithm. □

Theorem 1. *For any R-competitive deterministic algorithm for nonpreemptive scheduling on related machines, the following inequality holds:*

$$1 \le \int_0^1 \frac{\ln(R)}{-\ln(1 - R^{-x})}\, dx\,. \tag{6}$$

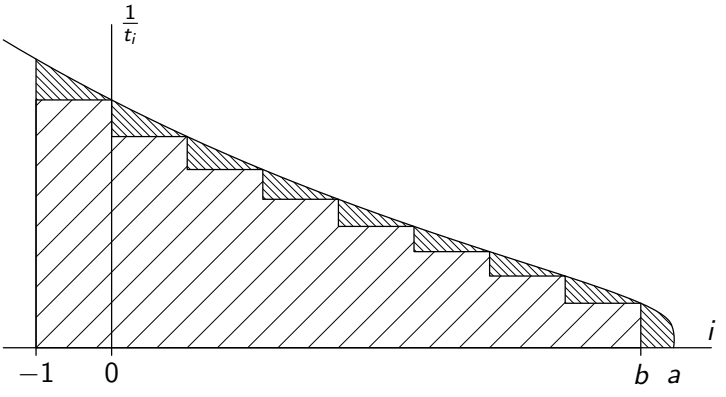


Fig. 1. The labels on the horizontal axis are $a = \log_\alpha R$ and $b = \lceil \log_\alpha R \rceil - 1$. The sparsely hatched region shows the area of the sum in (7). The densely hatched region shows the additional area of the integral in (8).

This gives $R > 2.564$.

Proof. Let n_i be the number of jobs scheduled on the machine M_i at the end of the sequence. The algorithm has to schedule all jobs, thus Lemma 1 implies

$$n = \sum_{i=0}^{n-1} n_i \quad = \quad \sum_{i=0}^{\lfloor \log_\alpha R \rfloor} n_i \quad \leq \quad (R+1)\lfloor \log_\alpha R \rfloor + \sum_{i=0}^{\lceil \log_\alpha R \rceil - 1} \frac{n}{t_i} \ .$$

(The change of the summation bound is an artifact of the subtlety of $i = \log_\alpha R$: There t_i is not defined but $n_i \leq 1$.) We can set n arbitrarily large, so that the term $(R+1)\lfloor \log_\alpha R \rfloor$ is negligible. Thus, for any $\varepsilon > 0$, we get:

$$1 - \varepsilon \quad \leq \quad \sum_{i=0}^{\lfloor \log_\alpha R \rfloor} \frac{1}{t_i} \quad = \quad \sum_{i=0}^{\lfloor \frac{\ln R}{\ln \alpha} \rfloor} \frac{\ln \alpha}{-\ln(1-\alpha^i R^{-1})} \tag{7}$$

$$\leq \quad \int_{-1}^{\frac{\ln R}{\ln \alpha}} \frac{\ln \alpha}{-\ln(1-\alpha^i R^{-1})} di \tag{8}$$

$$= \quad \int_{-\frac{\ln \alpha}{\ln R}}^{1} \frac{\ln R}{-\ln(1-R^{y-1})} dy \tag{9}$$

$$\xrightarrow{\alpha \to 1} \int_0^1 \frac{\ln R}{-\ln(1-R^{y-1})} dy \ . \tag{10}$$

In (8) we simply bound the sum by the appropriate integral. We use the fact that the function in the sum can be viewed as a continuous and decreasing function of i, see Figure 1. We substitute $i = y\frac{\ln R}{\ln \alpha}$ to get (9).

The inner function of the integral in (6) is a bounded monotone function of R and x. So we can solve the integration numerically and get the threshold of $R \approx 2.5649877$. □

Our bound with the limit argument gives little intuition about the size of the instance we need. In the rest of this section we give a bound of $O(1/\varepsilon^2)$ on the size of the instance for proving the lower bound of $R' = R - \varepsilon$. The two limits ($\varepsilon \to 0$ and $\alpha \to 1$) may need many machines and jobs for an accurate bound and we need to bound both of these errors.

Let us denote the expression (9) by $F_\alpha(R)$:

$$F_\alpha(R) \;=\; \int_{-\frac{\ln \alpha}{\ln R}}^{1} \frac{\ln R}{-\ln(1 - R^{y-1})} dy$$

Let R_α be the bound that is obtained from the inequality $1 \le F_\alpha(R_\alpha)$, that is $R_\alpha = \inf\{R \mid F_\alpha(R) \ge 1\}$.

Suppose that we have $\varepsilon' > 0$ such that $R' + \varepsilon' < R$, and $\alpha > 1$ such that $R' + \varepsilon' \le R_\alpha$. Then, to obtain a lower bound of R', we need to choose n such that

$$F_\alpha(R_\alpha - \varepsilon') \le 1 - (R' + 1)\lfloor \log_\alpha R' \rfloor / n.$$

We can prove asymptotic bounds analytically or more precise bounds by numerical checking on the computer. Let $c_1 = \frac{\partial}{\partial \alpha} F_\alpha(R')$ and $c_2 = \frac{\partial}{\partial R'} F_\alpha(R')$. For α close to 0 and R' between 2.56 and 2.57, c_1 and c_2 are positive, bounded away from 0 and bounded. Then, for a small ε', we obtain $F_\alpha(R_\alpha - \varepsilon') \approx 1 - \varepsilon' c_2$ and $R_\alpha \approx R - (\alpha - 1)\, c_1/c_2$. Using a computer we have checked numerically that $R_\alpha > R - 3(\alpha - 1)$ and $F_\alpha(R_\alpha - \varepsilon') < 1 - \frac{\varepsilon'}{2}$. The first inequality says that $\alpha = 1 + \varepsilon/4$ and $\varepsilon' = \varepsilon/4$ suffices to provide $R' + \varepsilon' \le R_\alpha$. The second inequality implies that it is sufficient to choose n so that $(R+1)\lfloor \log_\alpha R \rfloor \le n\frac{\varepsilon'}{2}$. A sufficiently large n is $n = \Theta\left(\frac{1}{\varepsilon' \ln(\alpha)}\right) = \Theta\left(\frac{1}{\varepsilon \ln(1+\varepsilon)}\right) = \Theta(\varepsilon^{-2})$.

3 Conclusions

We have been able to improve the lower bound for non-preemptive online scheduling on related machines. The advantage of the new lower bound is that it provides a clean analytical argument. On the other hand, it seems that the limit case with many machines may not be the hardest one. For a fixed small number of machines, we assume that the combinatorial structure of the problem could lead to new lower bounds. This would probably need some combination of our analytical approach and the enumerative techniques from [5].

Our techniques cannot be used for lower bounds against the randomized algorithms, the best lower bound in this case remains at 2.

Of course, a challenge in this area is to design new algorithms, perhaps not based on the doubling techniques used so far.

Acknowledgments. Partially supported by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR), grant IAA100190902 of GA AV ČR, and grant 166610 of GA UK. We are grateful to anonymous reviewers for helpful comments.

References

1. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line load balancing with applications to machine scheduling and virtual circuit routing. J. ACM 44, 486–504 (1997)
2. Azar, Y.: On-line Load Balancing. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms: The State of the Art. LNCS, vol. 1442, pp. 178–195. Springer, Heidelberg (1998)
3. Bar-Noy, A., Freund, A., Naor, J.: New algorithms for related machines with temporary jobs. J. Sched. 3, 259–272 (2000)
4. Berman, P., Charikar, M., Karpinski, M.: On-line Load Balancing for Related Machines. In: Rau-Chaplin, A., Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 1997. LNCS, vol. 1272, pp. 116–125. Springer, Heidelberg (1997)
5. Berman, P., Charikar, M., Karpinski, M.: On-line load balancing for related machines. J. Algorithms 35, 108–121 (2000)
6. Cho, Y., Sahni, S.: Bounds for list schedules on uniform processors. SIAM J. Comput. 9, 91–103 (1980)
7. Ebenlendr, T.: Combinatorial algorithms for online problems: Semi-online scheduling on related machines. PhD thesis, Charles University, Prague (2011)
8. Ebenlendr, T., Jawor, W., Sgall, J.: Preemptive online scheduling: Optimal algorithms for all speeds. Algorithmica 53, 504–522 (2009)
9. Epstein, L., Noga, J., Seiden, S.S., Sgall, J., Woeginger, G.J.: Randomized on-line scheduling for two uniform machines. J. Sched. 4, 71–92 (2001)
10. Epstein, L., Sgall, J.: A lower bound for on-line scheduling on uniformly related machines. Oper. Res. Lett. 26, 17–22 (2000)

Scheduling Jobs on Identical and Uniform Processors Revisited*

Klaus Jansen and Christina Robenek

Department of Computer Science
Christian-Albrechts-University Kiel
Christian-Albrechts-Platz 4, 24098 Kiel, Germany
{kj,cot}@informatik.uni-kiel.de

Abstract. We study the problem of scheduling jobs on uniform processors with the objective to minimize the makespan. In scheduling theory this problem is known as $Q||C_{\max}$. We present an EPTAS for scheduling on uniform machines avoiding the use of an MILP or ILP solver. Instead of solving (M)ILPs we solve the LP-relaxation and use structural information about the "closest" ILP solution. For a given LP-solution x we consider the distance to the closest ILP solution y in the infinity norm, i.e. $\|x-y\|_\infty$. We call this distance max-gap(A_δ), where A_δ is the constraint matrix of the considered (I)LP. For identical machines and $\delta = \Theta(\varepsilon)$ the matrix A_δ has integral entries in $\{0,\dots,(1+\delta)/\delta\}$ and $O(1/\delta \log(1/\delta))$ rows representing job sizes and $2^{O(1/\delta \log^2(1/\delta))}$ columns representing configurations of jobs, so that the column sums are bounded by $(1+\delta)/\delta$. The running-time of our algorithm is $2^{O(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta))} + O(n \log n)$ where $C(A_\delta)$ denotes an upper bound for max-gap(A_δ). Furthermore, we can generalize the algorithm for uniform machines and obtain a running-time of $2^{O(1/\varepsilon \log(1/\varepsilon) \log(C(\tilde{A}_\delta))} + poly(n)$, where $\tilde{A}_\delta$ is the constraint matrix for a sub-problem considered in this case. In both cases we show that $C(A_\delta), C(\tilde{A}_\delta) \leq 2^{O(1/\varepsilon \log^2(1/\varepsilon))}$. Consequently, our algorithm has running-time at most $2^{O(1/\varepsilon^2 \log^3(1/\varepsilon))} + O(n \log n)$ for identical machines and $2^{O(1/\varepsilon^2 \log^3(1/\varepsilon))} + poly(n)$ for uniform machines, the same as in [11]. But, to our best knowledge, no instance is known to take on the value $2^{O(1/\varepsilon \log^2(1/\varepsilon))}$ for max-gap(A_δ) or max-gap$(\tilde{A}_\delta)$. If $C(\tilde{A}_\delta), C(A_\delta) \leq poly(1/\varepsilon)$, the running-time of the algorithm would be $2^{O(1/\varepsilon \log^2(1/\varepsilon))} + poly(n)$ and thus improve the result in [11].

Keywords: scheduling on uniform processors, bin packing, EPTAS.

1 Introduction

In this paper we study the problem of scheduling jobs on uniform processors with the objective to minimize the makespan. In scheduling theory this problem

* Research supported by German Research Foundation (DFG) project JA 612/14-1, "Design and analysis of efficient polynomial approximation schemes for scheduling and related optimization problems".

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 109–122, 2012.

is known as $Q||C_{\max}$ and is formally described as follows. We are given a set $\mathcal{J}$ of n Jobs J_j with processing times p_j and a set $\mathcal{P}$ of m processors P_i, each of them running with a certain speed s_i. A job J_j needs p_j/s_i time units to be finished, if it is executed on P_i. Without loss of generality we assume that the number m of processors is bounded by the number n of jobs and that the processors are sorted by decreasing speed, i.e. $s_1 \geq s_2 \geq \ldots \geq s_m$. For an instance $\mathcal{I}$ let $OPT(\mathcal{I})$ denote the length of an optimum schedule.

A *polynomial time approximation scheme* (PTAS) for $Q||C_{\max}$ is a family of polynomial-time approximation algorithms $(A_\varepsilon)_{\varepsilon>0}$, where for an instance $\mathcal{I}$ the output of each algorithm A_ε is a schedule of length $(1+\varepsilon)OPT(\mathcal{I})$ and the running-time of A_ε is bounded by a polynomial in the input length $|\mathcal{I}|$. The running-time of every A_ε is allowed to be exponential in $1/\varepsilon$, which can lead to very large running-times if ε is very small. Therefore we distinguish furthermore *efficient polynomial-time approximation schemes* (EPTAS) that have running-time bounded by $f(1/\varepsilon)poly(|\mathcal{I}|)$ for a function f, and *fully polynomial-time approximation schemes* (FPTAS) with running-time bounded by a polynomial in both, $1/\varepsilon$ and $|\mathcal{I}|$.

Known Results. In [4] and [5] the problem was shown to be NP-hard even for identical processors. In 1976 Horowitz and Sahni [10] presented an approximation scheme for a constant number m of uniform processors. Later Gonzales et al. [4] showed for the same problem that the LPT list algorithm (using largest processing time first policy) has output in $[1.5\,OPT(\mathcal{I}), 2\,OPT(\mathcal{I})]$. Hochbaum and Shmoys presented a PTAS for $Q||C_{\max}$ with running-time $(n/\varepsilon)^{O(1/\varepsilon^2)}$ [8], [9]. For identical processors the complexity was improved to $(n/\varepsilon)^{O(1/\varepsilon \log(1/\varepsilon))}$ by Leung [17]. Since the problem was shown to be NP-hard in the strong sense [4], no FPTAS exists. But, for identical processors Hochbaum [7] and Alon et al. [1] developed an EPTAS with running-time $f(1/\varepsilon) + O(n)$, where f is a function doubly exponential in $1/\varepsilon$. In [11] Jansen gave an EPTAS for scheduling jobs on uniform processors using an MILP relaxation with running-time $2^{O(1/\varepsilon^2 \log^3(1/\varepsilon))} + poly(n)$. Sanders et al. obtained a robust online algorithm for scheduling on identical machines with competitive ratio $(1+\varepsilon)$ and migration factor $\beta(\varepsilon) = 2^{O(1/\varepsilon \log^2(1/\varepsilon))}$ so that the running-time for incorporating a newly arrived job is constant. It maintains and updates a data structure in time doubly exponential in $1/\varepsilon$, namely $2^{2^{O(1/\varepsilon \log^2(1/\varepsilon))}}$, in each iteration. This is done by comparing the distance between solutions for ILPs with different right hand sides. The general case for uniform processors is not considered.

Our Results. In this work we present an EPTAS for scheduling on uniform machines avoiding the use of an MILP or ILP solver. In our new approach instead of solving (M)ILPs we solve the LP-relaxation and use structural information about the "closest" ILP solution. For a given LP-solution x we consider the distance to the closest ILP solution y in the infinity norm, i.e. $\|x-y\|_\infty$. For the constraint matrix A_δ of the considered LP we call this distance

$$\text{max-gap}(A_\delta) := \max\{\min\{\|y^\star - x^\star\|_\infty : y^\star \text{solution of ILP}\} : x^\star \text{solution of LP}\}.$$

Let $C(A_\delta)$ denote an upper bound for max-gap(A_δ). The running-time of our algorithm is $2^{O(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta))} + poly(n)$. We show that $C(A_\delta) \leq 2^{O(1/\varepsilon \log^2(1/\varepsilon))}$. Consequently, our algorithm has running-time at most $2^{O(1/\varepsilon^2 \log^3(1/\varepsilon))} + poly(n)$, the same as in [11]. But, to our best knowledge, no instance is known to take on the value $2^{O(1/\varepsilon \log^2(1/\varepsilon))}$ for max- gap(A_δ). We conjecture $C(A_\delta) \leq poly(1/\varepsilon)$. If that holds, the running-time of the algorithm would be $2^{O(1/\varepsilon \log^2(1/\varepsilon))} + poly(n)$ and thus improve the result in [11].

Methods. Assume that we are given an instance $\bar{\mathcal{I}}$ of m identical processors and n jobs with only d different processing times p_j, such that there are n_j jobs of each size. We use the dual approximation method by Hochbaum and Shmoys [9] to find a value T for the optimum makespan and transform the scheduling problem into a bin packing problem with bin size T. Then the problem can be described via the following configuration ILP for d different item sizes:

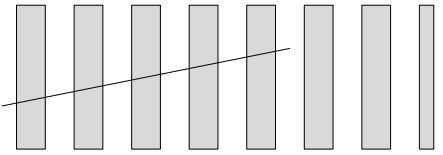
$$\begin{aligned} \sum_i x_i &\leq m \\ \sum_i a(j,i) x_i &\geq n_j \text{ for } j = 1, \ldots, d \qquad (ILP(d)) \\ x_i &\in \mathbb{Z}_{\geq 0}. \end{aligned}$$

A configuration C_i is a multiset of processing times p_j so that their total sum is bounded by T. The integer $a(j,i)$ denotes the number of jobs of processing time p_j in C_i. In $ILP(d)$ the variable x_i is the number of bins in which jobs are packed according to configuration C_i.

Solving an ILP is always difficult [14,15], so what kind of information about the structure of the ILP-solution can we get from a solution of the LP-relaxation? For the constraint matrix $A := (a(j,i))_{ji}$ of the above $ILP(d)$ we consider max-gap(A). Having an upper bound $C(A)$ for max-gap(A) and having an optimum fractional solution $x^\star$ we conclude that there exists an optimum solution $y^\star$ of $ILP(d)$ so that $y_i^\star \geq \lceil x_i^\star - C(A) \rceil$ for $x_i^\star \geq C(A)$. So we know how a subset of the bins $\mathcal{B}' \subset \mathcal{B}$ has to be filled with jobs in the optimum solution $y^\star$. We can reduce the instance to an instance $\bar{\mathcal{I}}_{red}$ by taking out the bins in $\mathcal{B}'$ and those jobs that are packed in $\mathcal{B}'$:

$$\begin{aligned} \tilde{m} &:= m - \sum_{x_i^\star > C(A)} \lceil x_i^\star - C(A) \rceil \text{ processors} \\ \tilde{n}_j &:= n_j - \sum_{x_i^\star > C(A)} a(j,i) \lceil x_i^\star - C(A) \rceil \text{ for all processing times } p_j. \end{aligned} \qquad (1)$$

In Figure 1 for example we have $C(A) = 3$. Given an optimum fractional solution $x^\star$ we conclude that there exists an optimum solution $y^\star$ of the ILP with $\|x^\star - y^\star\|_\infty \leq 3$. Thus, if $x_i^\star = 7.5$ we have $y_i^\star \geq 5$. Therefore we know that there is an integral solution of ILP (1) where at least 5 bins are occupied with configuration C_i. We take out these 5 bins and the corresponding



$C(A_\delta) = 3$ and $x_i^\star = 7.5 \Rightarrow y_i^\star \geq 5$

Fig. 1. Reducing the instance

jobs. Keep in mind that if the number of different job sizes and the number of jobs per bin is bounded by a constant, the total number of remaining jobs in $\bar{\mathcal{I}}_{red}$ can be bounded by a function in the value $C(A)$ namely by #(non-zero configurations in LP-solution) $*$ #(jobs per bin) $* C(A)$. Cook et al. [2] showed for general (I)LPs that max-gap(A) is bounded by Δ times the number of variables, where Δ is the maximum absolute value of a subdeterminant of the constraint matrix A.

So for an instance $\mathcal{I}$ with identical machines our algorithm first chooses $\delta \in \Theta(\varepsilon)$ and finds by binary search a candidate T for the makespan with $OPT(\mathcal{I}) \leq T \leq (1+\delta)OPT(\mathcal{I})$. By scaling we can assume that $T = 1$ and round the processing times p_j to values $\bar{p}_j = \delta(1+\delta)^{k_j}$ with $k_j \in \mathbb{Z}$ such that $p_j \leq \bar{p}_j \leq (1+\delta)p_j$. Consequently, we have to enlarge the bin capacities slightly to $(1+\delta)T = (1+\delta)$. With $\bar{\mathcal{I}}$ we denote the instance of rounded jobs, that are large, i.e. $\bar{p}_j > \delta$. We set up a configuration ILP for $\bar{\mathcal{I}}$ with $2^{O(1/\delta \log^2(1/\delta))}$ variables and constraint matrix A_δ as described above and solve the LP-relaxation or decide that no solution exists. In the latter case we increase the value T and restart. Notice that we have at most $O(1/\delta)$ large jobs per bin and by the rounding we have $R \in O(1/\delta \log(1/\delta))$ different large job sizes. Solving the LP-relaxation can be done in time $poly(1/\delta, \log n)$ [6]. Using the theorem by Cook et al. [2] we show that $C(A_\delta)$ is at most $2^{O(1/\delta \log^2(1/\delta))}$. Having a solution of the LP-relaxation we can reduce the instance as described in equation 1. The number of remaining large jobs in $\bar{\mathcal{I}}_{red}$ is bounded by $2^{O(1/\delta \log(1/\delta) \log(C(A_\delta)/\delta))}$. We allocate them by a dynamic programming approach. If this fails, we increase T and restart. In the end the small jobs are added greedily. The running-time is composed as follows "sorting the items by size"+"binary search on T"$*$"solving the LP"$*$"dynamic program"+"adding small jobs". This gives total running-time $O(n \log n) + O(\log(1/\varepsilon))poly(1/\varepsilon, \log n)2^{O(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta)))} + O(n) \leq O(n \log n) + poly(\log n)2^{O(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta)))} \leq 2^{O(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta)))} + O(n \log n)$.

The algorithm for uniform processors is much more complex. Since we have different bin capacities for uniform machines, we cannot directly apply the techniques used for identical machines. Therefore, we distinguish between three different scenarios for the shape of the bin sizes. For each scenario we give an algorithm that solves the problem in time $2^{O(1/\delta \log(1/\delta) \log(C(\tilde{A}_\delta)))} + poly(n)$ that applies our new technique to a subset of the instance. Furthermore, we use a new technique to round LP solutions for fractional bin packing producing

only few extra bins. In all cases the running time depends on $C(\tilde{A}_\delta)$ and is $2^{O(1/\varepsilon^2 \log^3(1/\varepsilon))} + poly(n)$ in the worst case for $C(\tilde{A}_\delta)$. Here, the matrix $\tilde{A}_\delta$ describes the constraints appearing in an ILP-approach that characterizes a more general scheduling problem as the one for identical machines: the jobs have a bounded number of different sizes and the machines run group-wise with the same speed, so we have configurations for each group. The entries of $\tilde{A}_\delta$ are integers in $\{0, 1, \ldots, (1+\delta)g(1/\delta)/\delta\}$ and the column sums are bounded by $(1+\delta)g(1/\delta)/\delta + 1$ for a function $g(1/\delta) = poly(1/\delta)$ that will be specified later. The value $C(\tilde{A}_\delta)$ is an upper bound for max-gap$(\tilde{A}_\delta)$.

We found out that the value Δ for matrices describing scheduling problems can be exponential in the number of different item sizes (see Lemma 2). But, no instance is known to take on the upper bound for max-gap(A_δ) or max-gap$(\tilde{A}_\delta)$. Therefore, an open question is to find a better bound for max-gap(A_δ) and max-gap$(\tilde{A}_\delta)$. One can also think of a robust online algorithm for identical processors or even for uniform processors with improved running-time using similar techniques.

Organization of the paper. We show that max-gap(A_δ) is bounded from above and that Δ is bounded from below in Section 2. In Section 3 we present an efficient algorithm for uniform processors that avoids to solve (M)ILPs and uses an upper bound for max-gap$(\tilde{A}_\delta)$ instead. Here, we proceed by case distinction and consider three different scenarios for the bin sizes. Due to space limitation we left out some proofs that can be found in the full version of the paper [13].

2 Bounds for max-Gap(A_δ) and the Running-Time

To obtain an upper bound $C(A_\delta)$ for max-gap(A_δ) we use an interesting result by Cook et al. [2]. They proved that the maximum distance between an optimum solution of the LP and a closest optimum solution of the ILP (and vice versa) is bounded by a function in the dimension and the coefficients of the underlying matrix.

Theorem 1. *[2] Let A be an integral $(M \times N)$ matrix, such that each subdeterminant is at most Δ in absolute value, and let b and c be vectors. Suppose that both objective values (i) $\min\{c^T x | Ax \geq b\}$ and (ii) $\min\{c^T x | Ax \geq b;\ x \in \mathbb{Z}^N\}$ are finite. Then:*

(a) for each optimum solution y of (i) there exists an optimum solution z of (ii) with $\|y - z\|_\infty \leq N\Delta$ and
(b) for each optimum solution z of (ii) there exists an optimum solution y of (i) with $\|y - z\|_\infty \leq N\Delta$.

Note that the theorem above also holds, if we have additional inequalities of the form $x_i \geq 0$. Furthermore, we can use $c^T x = \sum_i x_i$ as objective function instead of the inequality $\sum_i x_i \leq m$ in *ILP(d)*. For scheduling on identical processors the objective values of the ILP formulation for the rounded large jobs $\bar{\mathcal{I}}$ and its LP relaxation both are finite. Consequently, max-gap(A_δ) is bounded by $N\Delta$. In the following we give bounds for the parameters N and Δ.

Lemma 1. *The number of variables N in the modified ILP, the maximum absolute value Δ over all subdeterminants corresponding to the matrix $A_\delta = (a(j,i))$ and* max-gap(A_δ) *are at most* $2^{O(1/\delta \log^2(1/\delta))}$.

Since $\delta = \Theta(\varepsilon)$, the algorithm described informal in the introduction fufills the following theorem.

Theorem 2. *There is an algorithm with running-time*

$$2^{O(1/\varepsilon \log(1/\varepsilon) \log(C(A_\delta)))} + O(n \log n) \leq 2^{O(1/\varepsilon^2 \log^3(1/\varepsilon))} + O(n \log n)$$

that schedules n jobs on m identical processors producing a schedule with makespan at most $(1+\varepsilon)OPT(\mathcal{I})$.

If $C(A_\delta) = poly(1/\varepsilon)$, the running-time improves to $2^{O(1/\varepsilon \log^2(1/\varepsilon))} + O(n \log n)$. On the other hand, the value Δ can be quite large.

Lemma 2. *The maximum value Δ over all subdeterminants of the coefficient matrix $A_\delta = (a(j,i))$ is at least $2^{\Omega(1/\delta \log^2(1/\delta))}$.*

3 Scheduling on Uniform Processors

For uniform processors we can compute a 2 - approximation using the LPT algorithm studied by Gonzales et al. [4]. Here $LPT(\mathcal{I}) \leq 2OPT(\mathcal{I})$ where $LPT(\mathcal{I})$ is the schedule length generated by the LPT algorithm. Similar to identical processors, we can split the interval $[LPT(\mathcal{I})/2, LPT(\mathcal{I})]$ into $1/\delta$ subintervals of length $(\delta/2)LPT(\mathcal{I}) \leq \delta OPT(\mathcal{I})$ and transform the scheduling problem with makespan T into a bin packing problem with bin sizes $c_1 \geq \ldots \geq c_m$ (where $c_i = T \cdot s_i$). By scaling we assume $c_m = 1$. As for identical machines we round the job sizes p_j to values $\bar{p}_j = \delta(1+\delta)^{k_j} \leq (1+\delta)p_j$. Additionally we round and increase slightly the bin capacities c_i to values $\bar{c}_i = (1+\delta)^{\ell_i} \leq c_i(1+\delta)^2$. Let the instance of rounded jobs and bin capacities be denoted with $\bar{\mathcal{I}}$. For a set of bins $\mathcal{B}$ let $c_{\min}(\mathcal{B}) := \min\{c_i | b_i \in \mathcal{B}\}$. Analogously we define $c_{\max}(\mathcal{B})$.

Lemma 3. *[11] If there is a feasible packing of n jobs with processing times p_j into m bins with capacities $c_1, \ldots, c_m$, then there is also a packing of the n jobs with rounded processing times $\bar{p}_j = \delta(1+\delta)^{k_j} \leq (1+\delta)p_j$ into the m bins with rounded bin capacities $\bar{c}_i = (1+\delta)^{\ell_i} \leq c_i(1+\delta)^2$.*

In the general case with different bin sizes, we distinguish between three different scenarios depending on the structure of the set of bins in the instance. Let $g : \mathbb{N} \longrightarrow \mathbb{N}$ and $f : \mathbb{N} \longrightarrow \mathbb{N}$ be functions so that $g(1/\delta) \geq 1/\delta \log^2(1/\delta)$ with $g = poly(1/\delta)$ and $f(1/\delta) = \max\{\lceil (1+\delta+\log^2(1/\delta))/\delta \rceil, 1/\delta^4 \log(g(1/\delta)/\delta) C(\tilde{A}_\delta)\}$. The constant $C(\tilde{A}_\delta)$ is still an upper bound for max-gap$(\tilde{A}_\delta)$. Here $\tilde{A}_\delta$ is a matrix corresponding to a more general scheduling problem with $O(1/\delta \log(1/\delta))$ rows (different job sizes) and $2^{O(1/\delta \log^2(1/\delta))}$ columns (configurations) with integral entries in $\{0, 1, \ldots, (1+\delta)g(1/\delta)/\delta\}$ and column sums bounded by $(1+$

$\delta)g(1/\delta)/\delta+1$ similar to the constraint matrix of the configuration ILP used for identical processors. We consider the following three scenarios:

Case 1: For all $i \in \{1,\dots,m\}$ we have $\bar{c}_1/\bar{c}_m \leq g(1/\delta)$.
Case 2: There exists an index $K+1 \leq f(1/\delta)$ with $\bar{c}_1/\bar{c}_i < g(1/\delta)$ for $1 \leq i \leq K$ and $\bar{c}_1/\bar{c}_i \geq g(1/\delta)$ for $K+1 \leq i \leq m$.
Case 3: There exists an index $K+1 > f(1/\delta)$ with $\bar{c}_1/\bar{c}_i < g(1/\delta)$ for $1 \leq i \leq K$ and $\bar{c}_1/\bar{c}_i \geq g(1/\delta)$ for $K+1 \leq i \leq m$.

In the first scenario all bins have similar capacities. More precisely the capacity of every bin is bounded from above by $g(1/\delta)$ (Keep in mind that $c_{\min} := \min_i c_i = 1$). This scenario can be solved very similar to the problem with identical machines. Due to space limitations we omit that case here.

In the second scenario we consider two different bin groups $\mathcal{B}_0 := \{b_1,\dots,b_K\}$ and $\mathcal{B}_1 := \{b_{K+1},\dots,b_m\}$. For $\mathcal{B}_0$ we preprocess an assignment of large jobs ($\bar{p}_j > \delta c_{\min}(\mathcal{B}_0)$) via a dynamic program and obtain a set of assignment vectors V. If the dynamic program does not find a feasible solution for $\mathcal{B}_0$, we increase T. For $v \in V$ we allocate large jobs fractionally into $\mathcal{B}_1$ via an LP. If the LP does not have a feasible solution we compute a different vector v. If we still do not find an LP solution, we increase T. Then we round the solution of the LP with a novel rounding technique using a subroutine for bin packing with different bin sizes that produces only few extra bins. In the end the small jobs are scheduled behind the large ones. The complete algorithm can be found in the full version [13].

The third scenario is the most complicated case. Here we have three bin groups $\mathcal{B}_0 = \{b_1,\dots,b_K\}$, $\mathcal{B}_1 = \{b_i | i > K, \bar{c}_i \geq \delta c_{\min}(\mathcal{B}_0)\}$ and the remaining bins $\mathcal{B}_2 = \mathcal{B} \setminus (\mathcal{B}_0 \cup \mathcal{B}_1)$. If $\mathcal{B}_1 \neq \emptyset$ we distinguish large, medium and small jobs, else we only have large and small jobs:

A job is called large if $\bar{p}_j > \delta c_{\min}(\mathcal{B}_0)$ and medium if $\bar{p}_j \in (\delta c_{\min}(\mathcal{B}_1), \delta c_{\min}(\mathcal{B}_0)]$; other jobs are called small. We first allocate a subset of the large jobs into $\mathcal{B}_0$ and $\mathcal{B}_1$ using a linear program. As in the case for identical machines for a given solution x of the LP we reduce the instance by the number of large jobs surely packed in the closest integral solution. If the LP has no feasible solution we have to increase T and restart. Via dynamic programming our algorithm obtains an assignment of the remaining large jobs into $\mathcal{B}_0$ and $\mathcal{B}_1$ (if there is none, increase T). The medium jobs are packed with a bin packing subroutine into $\mathcal{B}_1$. Finally, the allocated medium and large jobs are fit together with the remaining jobs and the small jobs are added. An overview of the algorithm for this case is given in Figure 1. In the next section we describe this algorithm fully detailed.

3.1 Algorithm for Case 3

In this case we have two or three bin groups depending on the shape of bin sizes as depicted in Figure 2. Let $\mathcal{B}_0 = \{b_1,\dots,b_K\}$ be the set of the largest bins. Then, we define $\mathcal{B}_1 = \{b_i | i > K, \bar{c}_i \geq \delta c_{min}(\mathcal{B}_0)\}$ and $\mathcal{B}_2 = \mathcal{B} \setminus (\mathcal{B}_0 \cup \mathcal{B}_1)$ as the remaining bins. If $\mathcal{B}_1 \neq \emptyset$ we distinguish large, medium and small jobs. A job is called *large* if $\bar{p}_j > \delta c_{min}(\mathcal{B}_0)$ and *medium* if $\bar{p}_j \in (\delta c_{min}(\mathcal{B}_1), \delta c_{min}(\mathcal{B}_0)]$; other jobs are

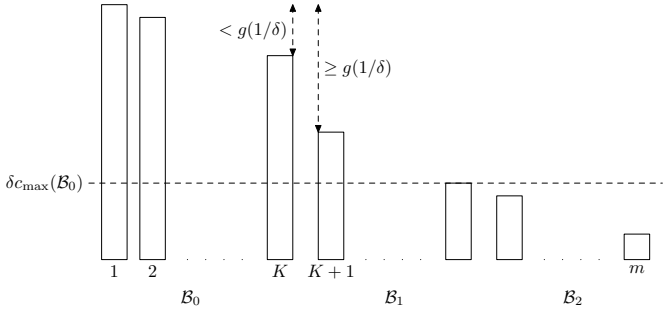


Fig. 2. Shape of bins for case 3

called *small*. Note that for a medium job we have $\bar{p}_j \leq \delta c_{\min}(\mathcal{B}_0) \leq c_{\max}(\mathcal{B}_1)$ by construction. If $\mathcal{B}_1 = \emptyset$ we do not have medium jobs. In this case we have for all $i > K$ that $\bar{c}_i \leq \delta c_{\min}(B_0)$. Thus, we have an additional gap between $\bar{c}_K$ and $\bar{c}_{K+1}$, i.e. $\frac{\bar{c}_K}{\bar{c}_{K+1}} = \frac{c_{\min}(\mathcal{B}_0)}{c_{\max}(\mathcal{B}_1)} > \frac{c_{\min}(\mathcal{B}_0)}{\delta c_{\min}(\mathcal{B}_0)} = \frac{1}{\delta}$.

Lemma 4. *Let A be a set $\{a(1+\delta)^x, \ldots, a(1+\delta)^y\}$ with $x, y \in \mathbb{Z}^+$, $x < y$ and $a \in \mathbb{R}^+$. Then $|A| \geq \log(\max(A)/\min(A))/\delta + 1$ and $|A| \leq 2\log(\max(A)/\min(A))/\delta + 1$ for any $\delta \in (0, 1/2]$.*

The above Lemma implies that the number of different rounded bin sizes and large and medium job sizes corresponding to $\mathcal{B}_0 \cup \mathcal{B}_1$ is bounded by $O(1/\delta \log(g(1/\delta)/\delta))$ and $O(1/\delta \log(g(1/\delta)/\delta^2))$, respectively. Notice that both numbers are at most $O(1/\delta \log(1/\delta))$ since $g(1/\delta) = poly(1/\delta)$.

Now we divide the set $\mathcal{B} = \mathcal{B}_0 \cup \mathcal{B}_1 \cup \mathcal{B}_2$ into N groups B_ℓ with m_ℓ bins with the same rounded bin size $\bar{c}(\ell)$ for $\ell = 1, \ldots, N$ and set up a linear program. Later we consider a reduced LP for the first two bin groups separately. In the LP below we use a variable $x_i^{(\ell)}$ to indicate the fractional length of a multiset $C_i^{(\ell)}$ of large processing times $\bar{p}_j \in [\delta\bar{c}(\ell), \bar{c}(\ell)]$ packed into bins of size $\bar{c}(\ell)$. Let $a(j, i^{(\ell)})$ be the number of the occurrences of $\bar{p}_j$ in $C_i^{(\ell)}$ and let $size(C_i^{(\ell)}) = \sum_j a(j, i^{(\ell)})\bar{p}_j$. Furthermore, let n_j be the number of jobs with processing time $\bar{p}_j = \delta(1+\delta)^j$ for $j = 0, \ldots, R$ (where $\delta(1+\delta)^R$ is the largest jobs size). Finally, we use a variable $y_{j,\ell}$ to indicate the fractional number of jobs of size $\delta \leq \bar{p}_j < \delta\bar{c}(\ell)$ packed as a small job in B_ℓ.

$$
\begin{array}{lll}
\sum_i x_i^{(\ell)} & \leq m_\ell & \text{for } \ell = 1, \ldots, N \\
\sum_{\ell,i} a(j, i^{(\ell)}) x_i^{(\ell)} + \sum_\ell y_{j,\ell} & = n_j & \text{for } j = 0, \ldots, R \\
\sum_i size(C_i^{(\ell)}) x_i^{(\ell)} + \sum_j y_{j,\ell}\delta(1+\delta)^j & \leq m_\ell \bar{c}(\ell) & \text{for } \ell = 1, \ldots, N
\end{array}
$$

$$
\begin{array}{lll}
x_i^{(\ell)} & \geq 0 & \text{for } \ell = 1, \ldots, N \text{ and } i = 1, \ldots, h_\ell \\
y_{j,\ell} & \geq 0 & \text{for } j = 0, \ldots, R \text{ and } \ell = 1, \ldots, N
\end{array}
$$

Algorithm 1. Algorithm for case 3

1: Obtain 2 - approximation using the LPT algorithm.
2: Compute a value $T \in [LS(I)/2, LS(I)]$.
3: Round the processing times of the jobs and distinguish small, medium and large jobs
4: Allocate a subset of the large jobs into $\mathcal{B}_0$ and $\mathcal{B}_1$ using a linear program and with Theorem by Cook et al. [2] reduce the instance.
5: **if** the linear program does not have a feasible solution **then**
6: increase T and go to step 2.
7: **end if**
8: Via dynamic programming obtain an assignment of the remaining large jobs into $\mathcal{B}_0$ and $\mathcal{B}_1$.
9: **if** the dynamic program for I_{red} does not find a feasible solution **then**
10: increase T and go to step 2.
11: **end if**
12: Allocate medium jobs into $\mathcal{B}_1$ via a bin packing subroutine.
13: Fit the allocated large and medium jobs together with the remaining jobs.
14: Schedule the small jobs behind the large ones. .

We suppose that all jobs fit into the bins, i.e. $\delta(1+\delta)^R \leq c_{max}(\mathcal{B}_0)$; otherwise there is no schedule with the corresponding makespan in the binary search. Suppose that $\mathcal{B}_0$ consists of L bin groups B_ℓ and $\mathcal{B}_1$ consists of P bin groups, see also Fig. 3.

Allocating large jobs. Suppose that the entire LP and the corresponding ILP have a solution. Consider now the corresponding $(x_i^{(\ell)})$ variables and constraints for the first $L+P$ bin groups. Let $\delta(1+\delta)^{R_m}$ be the smallest medium job size and let $\delta(1+\delta)^{R_\ell}$ be the smallest large job size.

$$\begin{array}{lll} \sum_i x_i^{(\ell)} & \leq \bar{m}_\ell \leq m_\ell & \text{for } \ell = 1, \ldots, L+P \\ \sum_{\ell,i} a(j, i^{(\ell)}) x_i^{(\ell)} & \geq \bar{n}_j & \text{for } j = R_m, \ldots, R \end{array}$$

$$x_i^{(\ell)} \geq 0 \qquad \text{for } \ell = 1, \ldots, L+P \text{ and } i = 1, \ldots, h_\ell$$

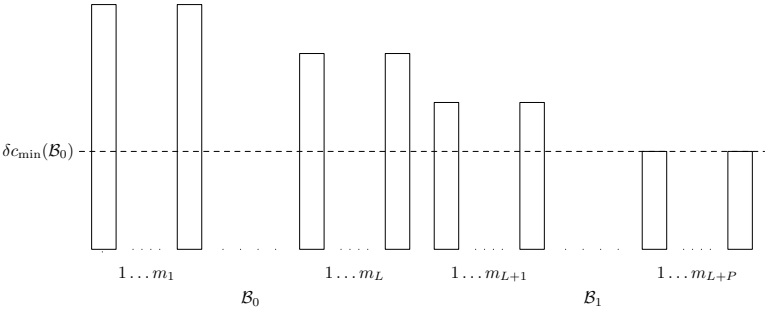


Fig. 3. Groups of similar capacities in $\mathcal{B}_0 \cup \mathcal{B}_1$

For a large job size we have $\bar{p}_j > \delta c_{min}(\mathcal{B}_0) > c_{\max}(\mathcal{B}_2)$. Hence, the large jobs have to be scheduled in $\mathcal{B}_0 \cup \mathcal{B}_1$. Consequently, we describe them by configuration variables only in the original LP and so the number $\bar{n}_j$ for large jobs covered by the LP above is integral and satisfies $\bar{n}_j = n_j$.

For medium job sizes, there are $y_{j,\ell}$ variables in the initial LP and we have in general fractional variables $\bar{n}_j \leq n_j$. Note that a configuration $C_i^{(\ell)}$ in $\mathcal{B}_0$ contains only large job sizes by construction and a configuration $C_i^{(\ell)}$ in $\mathcal{B}_1$ may contain both, large and medium job sizes.

Let $\bar{C}_k^{(\ell)}$ be a configuration with only large job sizes in bin group B_ℓ in $\mathcal{B}_1$ and let $z_k^{(\ell)}$ be a variable that indicates the total length of $\bar{C}_k^{(\ell)}$. For the rest of the paper we call $\bar{C}_k^{(\ell)}$ a big configuration. Then, the original configurations with both, medium and large job sizes, can be partitioned into groups with the same arrangement of large jobs according to configuration $\bar{C}_k^{(\ell)}$ (containing only large jobs). Let $Index(k,\ell)$ be the set of all indices i such that $C_i^{(\ell)}$ coincides with $\bar{C}_k^{(\ell)}$ for the large job sizes. Then, $z_k^{(\ell)} = \sum_{i \in Index(k,\ell)} x_i^{(\ell)}$ and the following modified LP for the large job sizes has a feasible solution.

$$
\begin{array}{lll}
\mathrm{LP}_{large} & & \\
\sum_i x_i^{(\ell)} & \leq m_\ell \text{ for } & \ell = 1, \ldots, L \\
\sum_k z_k^{(\ell)} & \leq m_\ell \text{ for } & \ell = L+1, \ldots, L+P \\
\sum_{\ell,i} a(j, i^{(\ell)}) x_i^{(\ell)} + \sum_{\ell,k} a(j, k^{(\ell)}) z_k^{(\ell)} & \geq n_j \text{ for } & j = R_\ell, \ldots, R \\
\\
x_i^{(\ell)} \quad \geq 0 & \text{for } \ell = 1, \ldots, L+P \text{ and } & i = 1, \ldots, h_\ell
\end{array}
$$

Since all large job sizes have to be placed into the first $L+P$ bin groups and using the assumption that the entire ILP has a solution, the modified ILP for the large job sizes has a feasible solution, too. Using the Theorem by Cook et al. [2], there is an ILP solution $(\hat{x}_i^{(\ell)}, \hat{z}_k^{(\ell)})$ with distances $\|\hat{x}_i^{(\ell)} - x_i^{(\ell)}\|_\infty$ and $\|\hat{z}_k^{(\ell)} - z_k^{(\ell)}\|_\infty$ bounded by $N\Delta \leq 2^{O(1/\delta \log^2(g(1/\delta)/\delta+1))}$. Notice that the column sum of a column of the constraint matrix $\sum_j a(j, i^{(\ell)})+1$ corresponding to a configuration is at most $c_{max}(\mathcal{B}_0)/(\delta c_{min}(\mathcal{B}_0))+1 \leq g(1/\delta)/\delta+1$. Since $g(1/\delta) = poly(1/\delta)$, the distances above are at most $C(\tilde{A}_\delta) = 2^{O(1/\delta \log^2(1/\delta))}$. If $x_i^{(\ell)}$ or $z_k^{(\ell)}$ is larger than $C(\tilde{A}_\delta)$, then we know that there is an integer solution with $\hat{x}_i^{(\ell)} \geq \lceil x_i^{(\ell)} - C(\tilde{A}_\delta) \rceil$ or $\hat{z}_k^{(\ell)} \geq \lceil z_k^{(\ell)} - C(\tilde{A}_\delta) \rceil$. Then we can reduce our instance $\bar{I}$ to a reduced instance I_{red} with $\tilde{n}$ large jobs and $\tilde{m}_\ell \leq m_\ell$ bins per block B_ℓ as described in the introduction.

The values of the coefficients of the constraint matrix are bounded by the number of large jobs per configuration which is at most $c_{max}(\mathcal{B}_0)/(\delta c_{min}(\mathcal{B}_0)) \leq g(1/\delta)/\delta$. The number of strict positive variables of a basic solution of the modified LP is at most $O(1/\delta \log(g(1/\delta)/\delta))$. Since each reduced variable has value at most $C(\tilde{A}_\delta)$, $\tilde{n}$ can be bounded by $O(1/\delta^2 g(1/\delta) \log(g(1/\delta)/\delta)) C(\tilde{A}_\delta)$. Since $g(1/\delta) = poly(1/\delta)$, the number of remaining large jobs is at most $\tilde{n} \leq poly(1/\delta) C(\tilde{A}_\delta) 2^{O(\log(1/\delta))} C(\tilde{A}_\delta)$. Moreover, this implies that we need at most $\tilde{M} \leq \tilde{n} \leq 2^{O(1/\delta \log^2(1/\delta))}$ machines for the large jobs in I_{red}. Since the modified

ILP for the large sizes has a feasible solution, we can find a solution for I_{red} by dynamic programming. Simply go over the machines in $\mathcal{B}_0 \cup \mathcal{B}_1$ and place the $\tilde{n}$ jobs onto the machines. This can be done by computing feasible vectors $(x_{R_\ell}, \ldots, x_R)$ that correspond to a packing of x_i large jobs of size $\bar{p}_i$ into the first k bins for $k = 1, \ldots, \sum_{\ell=1}^{L+P} \tilde{m}_\ell$. In this way we can find a feasible packing in time $\tilde{n}^{O(1/\delta \log(g(1/\delta)/\delta))} \leq 2^{O(1/\delta \log(1/\delta) \log \tilde{n})} \leq 2^{O(1/\delta \log(1/\delta) \log(C(\tilde{A}_\delta)))}$.

Allocating medium jobs. The main difficulty now is to handle the medium jobs. Consider the LP for the medium and large jobs corresponding to bin group $\mathcal{B}_1$. Take out for a moment the large jobs $I_{large,dp}$ placed by the dynamic program into $\mathcal{B}_1$. Notice that these large jobs have occupied a subset $M_{large,dp}$ of only $\tilde{M} \leq \tilde{n}$ machines in $\mathcal{B}_1$. Furthermore, notice that there are still large jobs preassigned via the big configurations $\bar{C}_k^{(\ell)}$ of length $\lceil z_k^{(\ell)} - C(\tilde{A}_\delta) \rceil = \lceil \sum_{i \in Index(k,\ell)} x_i^{(\ell)} - C(\tilde{A}_\delta) \rceil$ in $\mathcal{B}_1$. Since we have a feasible LP solution for all jobs, the residual configurations $C_i^{(\ell)}$ (restricted to medium job sizes) with fractional lengths $x_i^{(\ell)}$ fit into the gaps either besides their corresponding big configurations of lengths $\lceil z_k^{(\ell)} - C(\tilde{A}_\delta) \rceil$ or after them. The placement of medium jobs can be seen as a fractional bin packing problem with different bin sizes. We round the $x_i^{(\ell)}$ variables for $\mathcal{B}_1$ and use a bin packing subroutine:

New rounding technique. In our new approach we subdivide $\mathcal{B}_1$ into groups of bins $D_1, \ldots, D_H$ with similar bin sizes. These groups are not necessary equal to the groups $B_1, \ldots, B_N$ we considered to set up the above LP-relaxation. Then we use the solution of the LP relaxation above to pack the jobs or items via a bin packing algorithm. For each group D_k the bin packing algorithm packs the selected items into the group D_k of bins with different bin sizes plus few additional bins of maximum capacity $c_{\max}(D_k)$. Based on the subdivision the number of medium item sizes can be bounded by $d = O(1/\delta \log(1/\delta))$ for each group D_k. Using a recent result [12], we are able to pack the selected items into D_k plus $O(\log^2(d)) = O(\log^2(1/\delta))$ bins of capacity $c_{\max}(D_k)$. The overall goal is to obtain a packing of almost all jobs into $\mathcal{B}_1$ plus at most $O(\log^2(1/\delta))$ bins of capacity $c_{\max}(\mathcal{B}_1)$. In the following we explain in detail how this rounding works.

Suppose that $\mathcal{B}_1$ has a bin b_{i_1} with $\bar{c}_{i_1} < c_{\max}(\mathcal{B}_1)/h(1/\delta)$, where $h : \mathbb{R}^+ \to \mathbb{R}^+$ is a function with $poly(1/\delta) \geq h(1/\delta) \geq 1/\delta$. W.l.o.g. let $i_1 \in \{K+1, \ldots, m\}$ be minimal with that property. In this case we build $D_1 = \{b_{K+1}, \ldots, b_{i_1-1}\}$ and construct the other groups $D_2, \ldots, D_H$ iteratively in the same way. The next group $D_2 = \{b_{i_1}, \ldots, b_{i_2-1}\}$ fulfills the properties $c_{\min}(D_2) = \bar{c}_{i_2-1} \geq c_{\max}(D_2)/h(1/\delta)$ and $\bar{c}_{i_2} < c_{\max}(D_2)/h(1/\delta)$, see Figure 4. If all bins have capacity larger than or equal $c_{\max}(\mathcal{B}_1)/h(1/\delta)$, we have only one group $D_1 = \mathcal{B}_1$. With Lemma 4 we conclude that the number of different bin sizes in each group D_k is at most $O(1/\delta \log(h(1/\delta)))$ and the number of medium job sizes in D_k is at most $O(1/\delta \log(h(1/\delta)/\delta))$. Since $h(1/\delta) \leq poly(1/\delta)$, both numbers are bounded by $O(1/\delta \log(1/\delta))$. Consider now a linear program solution $x_i^{(\ell)}$ and

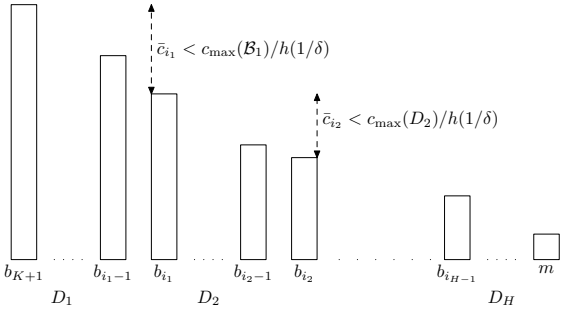


Fig. 4. Grouping $\mathcal{B}_1$ into D_1 to D_H

consider the reduced linear program LP_k with corresponding constraints for the bin group D_k.

$$
\begin{array}{lll}
\mathrm{LP}_k: & & \\
\sum_i x_i^{(\ell)} & = \bar{m}_\ell & \text{for bins of capacity } c(\ell) \text{ in } D_k \\
\sum_{\ell,i} a(j, i^{(\ell)}) x_i^{(\ell)} & \geq n_j^{(k)} & \text{for each medium job size in } D_k \\
x_i^{(\ell)} & \geq 0 & \text{for } \ell = 1, \ldots, L,
\end{array}
$$

The value $\bar{m}_\ell \leq m_\ell$ is the fractional number of bins of size $\bar{c}(\ell)$ in D_k and $n_j^{(k)}$ is the fractional number of medium job sizes $\delta(1+\delta)^j$ placed into D_k according to the solution of $\mathrm{LP}_{\mathrm{large}}$. If in LP_k we replace the right hand sides $n_j^{(k)}$ by $\lfloor n_j^{(k)} \rfloor$ for each medium job size, we have to cover an integral number of jobs. Thus, the total execution time $\sum_{\bar{p}_j \text{medium in } D_k} \delta(1+\delta)^j$ of the non-covered medium jobs in D_k for can be bounded by $c_{\max}(D_k) \sum_{j=0}^{\infty} (1+\delta)^{-j} = c_{\max}(D_k)(1+\delta)/\delta$ (using the geometric sum over the job sizes). Since medium jobs have processing time $\bar{p}_j \in (\delta c_{\min}(\mathcal{B}_1), \delta c_{\min}(c_{\min}\mathcal{B}_0))$ the additional execution time of non-covered jobs in D_1 is bounded by $\delta c_{\min}(\mathcal{B}_0) \sum_{j=0}^{\infty} (1+\delta)^{-j} = (1+\delta)\mathcal{B}_0$.

Now a (fractional) solution of the modified LP_k can be transformed into an integral solution. That means $\lfloor n_j^{(k)} \rfloor$ jobs of size $\delta(1+\delta)^j$ can be packed into the bins in D_k plus $O(\log^2(d))$ additional bins of size $c_{\max}(D_k)$ [12] (where $d = O(1/\delta \log(1/\delta))$ is the number of different medium job sizes). Notice that it is allowed to use m_ℓ bins instead of the fractional number $\bar{m}_\ell$ of bins in each group B_ℓ. This is sufficient, since the overall area $\sum_{\ell: B_\ell \subset D_k} Area(large, \ell)$ of the medium jobs packed into D_k plus the extra bins remains the same.

Lemma 5. *The total execution time of the medium jobs in the additional bins for $D_1, \ldots, D_H$ is at most* $(1+\delta)\min\{c_{\min}(\mathcal{B}_0), (1/\delta)c_{\max}(D_1)\}$ $+ O(\log^2(1/\delta))c_{\max}(D_1)$.

Since medium jobs are small corresponding to $\mathcal{B}_0$ and since $K \geq f(1/\delta) \geq \lceil (1+\delta+\log^2(1/\delta))/\delta \rceil$ we can distribute medium jobs corresponding to the additional term among the first K bins. Here we use a greedy algorithm that allocates a

load of at least $\delta c_{\min}(\mathcal{B}_0)$ and at most $2\delta c_{\min}(\mathcal{B}_0)$ on the first bins. This increases the makespan by at most $2\delta c_{\min}(\mathcal{B}_0)$. It is also possible that the total area of large jobs pre-assigned via the LP to $\mathcal{B}_0$ is smaller than the total area of large jobs placed via the pre-assignment with configuration lengths $\lceil x_i^{(\ell)} - C(\tilde{A}_\delta)\rceil$ and the dynamic program into $\mathcal{B}_0$. This implies that this additional occupied area in $\mathcal{B}_0$ can not be used for medium and small jobs. Then some medium jobs cannot be placed correctly onto the machines. We show below how to place these jobs into $\mathcal{B}_0$. Furthermore, some small jobs have to be placed into $\mathcal{B}_1$. But this is easier and possible, since these jobs are small corresponding to the bins in $\mathcal{B}_1$ and the total area of large, medium and small jobs corresponding to the variable values $x_i^{(\ell)}$ and $y_{j,\ell}$ for $\ell = 1, \ldots, K + L$ fits into $\mathcal{B}_0 \cup \mathcal{B}_1$.

Lemma 6. *The medium jobs, that do not fit into $\mathcal{B}_0$ because of additional large jobs placed by the dynamic program into $\mathcal{B}_0$, can be distributed among the machines in $\mathcal{B}_0$, so that the makespan is bounded by $(1 + O(\delta))OPT(\mathcal{I})$.*

Repacking process. Packing the allocated large and medium jobs together with the remaining jobs into the bins requires an extensive repacking process described in the following steps.

Step 1: Remove the set A_{medium} of medium jobs placed onto machines belonging to $M_{large,dp} \subset \mathcal{B}_1$. Reinsert the large jobs from $I_{large,dp}$ onto these machines and place fractionally a subset of A_{medium} into the remaining gaps.

Lemma 7. *The schedule produced in Step 1 can be made integral and has makespan at most $(1 + O(\delta))OPT(\mathcal{I})$.*

Step 2: Round the $(x_i^{(\ell)})$ variables corresponding to $\mathcal{B}_2$ and place the jobs via a bin packing subroutine into $\mathcal{B}_2$ plus some additional bins of size $c_{max}(\mathcal{B}_2) \leq \delta c_{\max}(\mathcal{B}_0)$. This can be done via our new rounding technique similar as the medium jobs are placed in $\mathcal{B}_1$. The additional bins can be distributed among the first K bins.

Step 3: As in [11] we round the $(y_{j,\ell})$ variables over the bin groups B_ℓ using a result of Lenstra et al. [16] and place the corresponding jobs greedily onto the machines. Thereby we have to place in addition one fractional job per bin group on one machine. Since the jobs corresponding to $y_{j,\ell}$ are small in $\mathcal{B}_\ell$ we only have to increase the bin sizes slightly.

This implies that the algorithm for case 3 produces a schedule of length $(1 + O(\delta))OPT(\mathcal{I})$ in time $2^{O(1/\varepsilon \log(1/\varepsilon) \log(C(\tilde{A}_\delta)))} + poly(n)$. In the full version [13] we consider the remaining cases in detail and prove.

Theorem 3. *There is an EPTAS for scheduling jobs on uniform machines with running-time*

$$2^{O(1/\varepsilon \log(1/\varepsilon) \log(C(\tilde{A}_\delta)))} + poly(n) = 2^{O(1/\varepsilon^2 \log^3(1/\varepsilon))} + poly(n).$$

If $C(\tilde{A}_\delta) = poly(1/\varepsilon)$, the running-time improves to $2^{O(1/\varepsilon \log^2(1/\varepsilon))} + poly(n)$.

References

1. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation schemes for scheduling on parallel machines. Journal on Scheduling 1, 55–66 (1998)
2. Cook, W., Gerards, A.M.H., Schrijver, A., Tardos, É.: Sensitivity theorems in integer linear programming. Mathematical Programming 34, 251–264 (1986)
3. Eisenbrand, F., Shmonin, G.: Caratheodory bounds for integer cones. Operations Research Letters 34, 564–568 (2006)
4. Gonzales, T., Ibarra, O.H., Sahni, S.: Bounds for LPT schedules on uniform processors. SIAM Journal on Computing 6, 155–166 (1977)
5. Graham, R.J., 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)
6. Grötschel, M., Lovasz, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Heidelberg (1987)
7. Hochbaum, D.S.: Various notions of approximations: good, better, best, and more. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-Hard Problems, ch. 9, pp. 346–398. Prentice Hall (1997)
8. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: practical and theoretical results. Journal of the ACM 34, 144–162 (1987)
9. Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM Journal on Computing 17, 539–551 (1988)
10. Horowitz, R., Sahni, S.: Exact and approximate algorithms for scheduling nonidentical processors. Journal of the ACM 23, 317–327 (1976)
11. Jansen, K.: An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. SIAM J. Discrete Math. 24(2), 457–485 (2010)
12. Jansen, K.: A fast approximation scheme for the multiple knapsack problem. To appear in: International Conference on Current Trends in Theory and Practise of Computer Science, SOFSEM 2012 (2012)
13. Jansen, K., Robenek, C.: Scheduling on uniform processors revisited, Technical Report, University of Kiel
14. Kannan, R.: Minkowski's convex body theorem and integer programming. Mathematics of Operations Research 12, 415–440 (1987)
15. Lenstra, H.W.: Integer programming with a fixed number of variables. Mathematics of Operations Research 8, 538–548 (1983)
16. Lenstra, J.K., Shmoys, D.B., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Mathematical Programming 24, 259–272 (1990)
17. Leung, J.: Bin packing with restricted piece sizes. Information Processing Letters 31, 145–149 (1989)

Approximation Algorithms for Fragmenting a Graph against a Stochastically-Located Threat

David B. Shmoys[1,⋆] and Gwen Spencer[2,⋆⋆]

[1] School of ORIE and Dept. of Computer Science, Cornell University, Ithaca, NY
shmoys@cs.cornell.edu
[2] School of ORIE, Cornell University, Ithaca, NY
gms39@cornell.edu

Abstract. Motivated by issues in allocating limited preventative resources to protect a landscape against the spread of a wildfire from a stochastic ignition point, we give approximation algorithms for a new family of stochastic optimization problems.

1 Introduction

Increasing frequency of catastrophically-damaging wildfire events has stimulated interest among foresters and land managers in effective use of preventative fuel reductions. Traditional fire suppression policy has focused almost exclusively on realtime firefighting (once the fire has broken out), but preventative fuel reductions such as dead-brush removal, small-scale controlled burns, and crown raising can be applied in advance to slow or stop the spread of wildfires. Recent wildfire modeling literature has used historical and scientific information to estimate a distribution of wildfire occurrence in which both the ignition site and the wind direction can vary [3],[8].

The planning problem of how to allocate limited resources across preventative and realtime stages, and where to distribute preventative resources using probabilistic information motivates a natural new family of budgeted stochastic optimization problems that fragment (or cut) a landscape graph to isolate a stochastically occurring ignition point. A key feature is the tradeoff between spending preventively when only distributional knowledge is available and spending at increased cost once a fire has broken out. We explore a number of model variants. Studying this family of problems through the lens of efficient approximation, we give constant bicriteria approximations in trees, and a budget-balanced constant approximation for the limiting case in which real-time actions become prohibitively expensive. Our techniques also yield new approximation results for multistage stochastic extensions of the budgeted Maximum Coverage problem. The theme of our models (protecting a network from the spread of a stochastic

⋆ Work supported under grants no. CCR-0635121, DMS-0732196, CCF-0832782, CCF-1017688.

⋆⋆ Supported in part by NSF under a Graduate Research Fellowship and under grants no. CCR-0635121, DMS-0732196, CCF-0832782, CCF-1017688.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 123–136, 2012.

outbreak of a harmful diffusive process) has other important environmental applications (e.g., containing invasive species over land or through water systems).

Results. In trees, the problem is (weakly) NP hard even when there is a single ignition point that is known deterministically [4] (the Knapsack Problem is a special case). An existing PTAS in graphs of bounded treewidth for the deterministic ignition-point case extends immediately to a PTAS in graphs of bounded treewidth for the deterministic ignition-set case. Applying some careful partial-enumeration then allows a PTAS in trees for the stochastic case in which the number of scenarios is constant.

The Graph Protection Problem: Summary of Main Results		
	restricted graph classes	general graphs (via Räcke) [5]
2-stage		
stochastic, single source	trees: $(1-(1-1/2\delta)^{2\delta}, 2)$ Via pipage rounding. *Alternative:* (0.387, 1)	constant number of scenarios $\Rightarrow$ $(1-(1-1/2n)^{2n}, O(\log n))$
stochastic, single source with (B_1, B_2)	trees: $(1-(1-1/2\delta)^{2\delta}, 1, 2)$ Via pipage rounding.	constant number of scenarios $\Rightarrow$ $(1-(1-1/2n)^{2n}, O(\log n), O(\log n))$
k-stage		
stochastic, single source	trees, restricted partition hierarchy: $(1-(1-1/k\delta)^{k\delta}, 2+\epsilon)$ Via pipage rounding.	constant number of scenarios and restricted partition hierarchy $\Rightarrow$ $(1-(1-1/kn)^{kn}, O(\log n))$
1-stage		
stochastic, single source with probabilistic edges	trees: $(1-1/e, 1)$ Due to submodularity.	open
stochastic, single source	trees: $(1-(1-1/\delta)^{\delta}, 1)$ Reduce to MCKP, apply [1].	$(1-(1-1/n)^{n}, O(\log n))$
stochastic with constant support and constant source size	trees: $(1+\epsilon, 1)$	$(1+\epsilon, O(\log n))$
deterministic with arbitrary source size	bounded tree width: $(1+\epsilon, 1)$	$(1+\epsilon, O(\log n))$
deterministic with single source	bounded tree width: $(1+\epsilon, 1)$ [4]	$(1+\epsilon, O(\log n))$ [4]

For the 2-stage stochastic model in which actions may either be taken in advance of the ignition based on probabilistic information, or after the single ignition point is known at inflated cost, we give a $(1-(1-1/2\delta)^{2\delta})$-approximation in trees which violates the budget by a factor of at most 2 (δ is the tree diameter). Notably, the inflation in the second stage can vary across scenarios and edges. For the limiting stochastic case in which no realtime action is possible, we give a $(1-(1-1/\delta)^{\delta})$-approximation algorithm in trees for the case of probabilistic ignition from a single source. We also give a 0.387-approximation which is budget-balanced for the 2-stage stochastic model, and some results for a k-stage extension. In some cases we can extend to general graphs with an

additional $O(\log n)$ loss in budget-balancedness via the probabilistic cut-capacity approximation result of Räcke [5] as in Engelberg, et al. [2].

For an extension in which transmission on edges is probabilistic and depends on the level of investment in removing the edge (assuming independence of edge realizations), we give a $(1 - 1/e)$-approximation algorithm in trees.

Our multistage and probabilistic-transmission results in trees also hold for analogous generalizations of the Maximum Coverage with Knapsack Constraint problem (MCKP) in which elements may *fail* independently with probability that depends on the level at which we invest in them, and the objective is to maximize the expected weight of the sets covered by the realized elements. For probabilistic element-failure MCKP, our guarantee matches the asymptotic guarantee for the deterministic element case from Ageev & Sviridenko [1].

Related Literature. The placement of preventative fuel treatments has been addressed in the recent forestry literature. Finney [3] prioritizes spatial fire spread dynamics, limits probabilistic model components, and aims to reduce the rate of spread of the head of fire. Wei et al. [8] considers the objective of reducing expected value lost across a grid-cell landscape by reducing burn probabilities (probabilities computed through simulation); however their IP-based approach is based on a questionable linearity assumption. These approaches produce divergent solution forms: the development of additional mathematical tools and techniques that simultaneously address stochastic and spatial aspects would be useful to decision-makers faced with this important planning problem.

The problems we study have ties to the existing computer science literature. The special case in which the ignition point is known deterministically and there is a single decision stage has been studied as the Minimum-Size Bounded-Capacity Cut problem by Hayrapetyan et al. [4]. They show that the problem is weakly NP-hard in trees by reduction from the Knapsack problem. In general graphs they give two different $(\frac{1}{1-\lambda}, \frac{1}{\lambda})$ bicriteria-approximations for the (expected value burned, budget), and they give a PTAS in graphs of bounded tree width. Engelberg, et al. [2] study a number of budgeted cut problems in graphs including the weighted Budgeted Separating Multiway Cut Problem (wBSMC), which the single-stage (aka, no realtime action) stochastic version of our problem reduces to. They apply Räcke's probabilistic cut-capacity-preserving approximation to reduce to the case of trees, then observe submodularity in trees, and apply [7] to get a $((1 - 1/e),\ O(\log n))$ bicriteria result. Our LP-based result for the single-stage stochastic version of our problem in trees generalizes to wBSMC in trees giving a slightly stronger $(1 - (1 - 1/n)^n,\ O(\log n))$ bicriteria result.

Techniques. For the deterministic case, a psuedopolynomial-time exact dynamic programming method is converted to an efficient scheme by rounding the input (as in [4]): our extension to general ignition sets is by demonstrating bounded treewidth of a modified input. For the extension with probabilistic-edge transmission, proving submodularity in tree graphs allows application of Sviridenko's [7] result on budgeted maximization of submodular functions. In the multistage-stochastic case, we solve a natural LP with a more complex

feasible region than that considered by Ageev & Sviridenko [1], but we are able to extend their pipage-rounding analysis to reduce the number of fractional variables: this requires additional specifications about which pairs of fractional decision variables may be rounded against each other and a careful treatment of the larger number of fractional variables that remain at the end of the pipage stage. All extensions from trees to general graphs employ the probabilistic capacity-preserving mapping of Räcke in the standard way (see [2]): approximate the costs by a distribution over trees, solve a suitably modified instance in each tree, translate solutions back to the original graph, select the best solution. Our techniques also yield similar results for stochastic multistage and probabilistic item-failure extensions of the constrained Maximum Coverage problem.

2 2-Stage Stochastic Graph Protection Problem in Trees

The spread of wild fires can be prevented both through advance fuel treatments and through real-time fire-fighting. Our model captures the tradeoff between using resources in advance vs. waiting until the realization of the ignition point is known but operations are more costly.

The input is a connected tree $T = (V, E)$, a non-negative *value* function $v : V \to \mathbb{Z}$, a non-negative *cost* function $c : E \to \mathbb{Z}$, and a *budget* B. A distribution Π over *source nodes* i is specified. In the first stage Π is known, and it costs c_e to remove edge e from T, in the second phase a realization from Π is specified (say the source is i), and edge e may be removed from T at cost $M^{ie}c_e$. That is: edges purchased in the second stage, once the source is known, have increased cost by a multiplicative *inflation factor* that may depend both on the scenario realized and on the edge.

The total spending on removing edges from T over both phases must be at most B. The objective is to specify a set of edges to buy in the first stage, and then a set of edges to buy in the second stage (depending on the realized source node from Π), such that the expected value not reachable from the realized source node is maximized. We aim to maximize the expected value *protected* from the source. We can contract all edges with costs strictly greater than B since they will not be in any optimal solution.

Special Case 0. Consider the limiting case when all second-stage actions are prohibitively expensive and also Π has support of size 1: this case is the Minimum-Size Bounded-Capacity Cut problem of Hayrapetyan, et al. [4]. They give a PTAS in graphs of bounded tree width and show that this deterministic problem with a single ignition node is NP-hard in trees.

Suppose in this deterministic single-stage case we replace the single ignition point s with a ignition set S. Now the objective is to maximize the expected value protected from every node in S by removing a budget-balanced set of edges.

Theorem 1. *There exists a PTAS in graphs of bounded tree width for the single-stage deterministic Graph Protection Problem (GPP) with a general ignition set.*

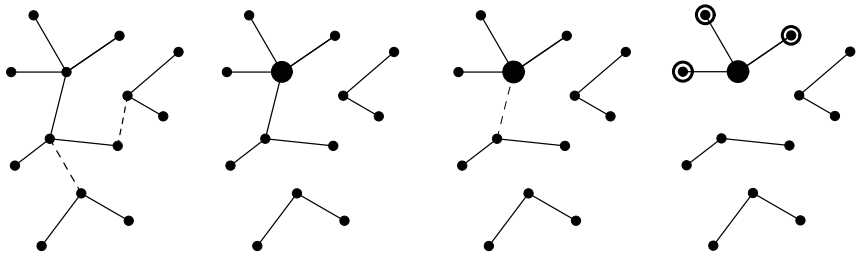


Fig. 1. Leftmost graph: The two dashed edges are removed in stage 1. Second graph: The (bolded) ignition node is realized. Third Graph: After ignition, additional edges can be removed in stage 2. Fourth Graph: Fire spreads through the connected component containing the ignition node: Non-ignition nodes lost to fire are shown circled.

A modified graph with a single source also has bounded tree width, so the existing PTAS can be applied. The PTAS asserted in Theorem 1 for trees can be produced directly by extending the classic dynamic programming framework for Knapsack. (see full paper [6]).

Applying [4] with an enumeration scheme over a polynomial number of divisions of the tree into source-containing components which can each be modified to act as a single-source deterministic problem, we get (details in [6]):

Theorem 2. *There exists a PTAS for the stochastic single-stage GPP in trees provided that the size of the support of the distribution Π and the size of each ignition set given positive weight by Π are bounded by a constant.*

Theorems 1 and 2 can be extended to bicriteria approximations for general graphs as in Engelberg, et. al [2]: the guarantees on value protected (expected value protected) are identical (though δ may be as much as n), and the budget is violated by a $O(\log n)$-factor (applying Räcke's result [5] on cut-capacity approximation). We mention this method briefly at the end of the paper.

Theorem 3. *There exists a bicriteria $(1-(1-\frac{1}{2\delta})^{2\delta}, 2)$-approximation algorithm for the 2-stage stochastic Graph Protection Problem in trees provided that each scenario has a single ignition node (δ denotes the tree diameter).*

In general graphs, for the case of a constant number of scenarios, Theorem 3 can be extended to a $(1-(1-\frac{1}{2n})^{2n}, O(\log n))$-bicriteria approximation (the multi-stage case requires an application of the Markov inequality to ensure $O(\log n)$-capacity distortion for each scenario under the cut-capacity approximation, details are at the end of the paper).

The following proof of Theorem 3 does not require that the node values are uniform across scenarios, but for notational convenience we will ignore this. This flexibility (and creative use of scenario-dependent edge costs) allows the input form to describe spatial properties of certain types of diffusive processes so that fragmenting the graph has more subtle process-specific implications for value protection than is immediately obvious when considering connectivity (details in [6]).

Roughly, the key ideas of the proof follow: the optimal fractional solution to a natural LP for 2-stage GPP acts as a starting point for a rounding algorithm. The rounding algorithm (carefully) chooses two fractional variables and rounds the LP solution along a vector that maintains their weighted sum (in order to retain feasibility of the budget constraints) while increasing a proxy function that matches the LP objective on integer points and remains boundedly close to the LP objective on fractional points. This is repeated until at most a few fractional variables remain. The effect of some final required roundings can be bounded against the value of an initial partial-enumeration phase. Since the final solution is obtained by a series of increasing steps for the proxy function, it will have high value compared to the original LP solution (for the correct partially-enumerated set). A technical point for the analysis is that a series of such integer solutions must be produced so that the effect of the final required roundings are small. Some simple alterations of this analysis will also yield results for single and k-stage versions as well as for a version in which the first and second stage budgets are specified in the input.

Proof. We formulate the following natural LP: $\max \sum_{(i,v)} (p_i v_v) x_{iv}$ such that $\sum_{e \in P(i,v)} y_e + \sum_{e \in P(i,v)} z_e^i \geq x_{iv}$ for all (i,v) pairs, $\sum_e y_e c_e + \sum_e z_e^i (M^{ie} c_e) \leq B$ for all i, and $x_{iv} \leq 1$ for all (i,v) pairs.

Here, p_i denotes the probability that node i is the ignition point under Π (the scenario where i is the ignition point is *scenario* i). In the associated IP, x_{iv} is 1 if node v is protected in scenario i, and 0 otherwise. Also, y_e is 1 if edge e is bought in the first stage, and 0 otherwise, and z_e^i is 1 if edge e is bought in the second stage for scenario i, and 0 otherwise. Constraints of the first form capture that if node v is protected in scenario i then it must be that some edge on the path from i to v is purchased either in the first stage or in the second stage for scenario i. Constraints of the second form capture that at most B can be spent buying edges in scenario i over the first and second stages combined. Preprocess by setting y_e to 0 if $c_e > B$, and z_e^i to 0 if $M^{ie} c_e > B$: the optimal solution can not use these options. Let δ denote the diameter of the tree.

Notice that in this LP, given a set of y_e and z_e^i, we can automatically determine the best x_{iv}. Following [1] we rewrite the problem as the following nonlinear optimization problem:

$$\max L(x) = \sum_{(i,v)} (p_i v_v) \min\{1, \sum_{e \in P(i,v)} y_e + \sum_{e \in P(i,v)} z_e^i\}$$

$$\text{s.t. } \sum_e y_e c_e + \sum_e z_e^i (M^{ie} c_e) \leq B \quad \forall i, \text{ and } x_{iv} \leq 1 \quad \forall (i,v).$$

Consider the function: $F(x) = \sum_{(i,v)} (p_i v_v) \Big[1 - \Big(\prod_{e \in P(i,v)} (1 - y_e) \Big) \Big(\prod_{e \in P(i,v)} (1 - z_e^i) \Big) \Big]$.

Lemma 1. *$F(x)$ has the following key properties:*

1. *$F(x)$ coincides with $L(x)$ when all the y_e and z_e^i are integral.*
2. *On non-integral (y_e, z_e^i) vectors, $F(x)$ is at least $(1-(1-\frac{1}{2\delta})^{2\delta})L(x)$.*
3. *$F(x)$ is concave in the direction of a vector that changes at most 2 y_e values at a time and changes no $z_{e'}^i$ values. $F(x)$ is concave in the direction of a vector that changes at most 2 z_e^i values for a common i at a time and changes no y_e values, and changes no $z_e^{i'}$ values for $i' \neq i$. Based on the budget constraint coefficients of the changing variables, vectors of this type can be found through appropriate scaling that maintain all budget constraints.*
4. *Let Y, Z denote sets corresponding to the y_e, z_e^i decision variables being set to 1. $F(X)$ defined on subsets of $Y \cup Z$ is a submodular set function.*

Properties 1, 2 and 4 hold just as in [1] since the function $F(x)$ has the same form (though now there is a formal distinction between first and second stage variables). For property 3: the number of terms in F's product which change for any particular (i, v) is at most 2: concavity results as in [1], but unlike in [1], not any set of two fractional decision variables will maintain budget feasibility).

Denote by LP$[I_0, I_1]$ the original LP (post preprocessing) subject to the additional constraints that decision variables in I_1 are set to 1 and decision variables in I_0 are set to 0. We use an auxiliary algorithm $\mathcal{A}$ identical to [1] except for a key additional point. First, $\mathcal{A}$ computes the optimal solution x^{LP} to LP$[I_0, I_1]$ by some known polynomial-time algorithm, then $\mathcal{A}$ transforms this solution into $x^{\mathcal{A}}$ by a series of pipage steps. Each pipage step is as follows. If there exists only a single fractional variable among the y_e, and for every i there is at most a single fractional variable among the z_e^i, stop. Otherwise, select either two fractional y_e or two fractional z_e^i for a common i and consider the vector that maintains all budget constraints as one is increased while the other is decreased: this vector intersects the boundary of the feasibility polytope at two points. At one point the first decision variable has become 0 and the second has become 1, at the other point the second decision variable has become 0 and the first has become 1. Both points are feasible since all budget constraints are maintained, and one has $F(X)$ at least as great as the previous solution due to the concavity of F along the vector. We replace the current solution with this higher-$F(X)$ solution that has a greater number of integral variables.

Each pipage step of $\mathcal{A}$ reduces the number of fractional components of the current vector. Finally $\mathcal{A}$ outputs an *almost-integral* feasible vector $x^{\mathcal{A}}$ which has at most one fractional first-stage variable, and at most one fractional second-stage variable for each scenario i.

As in [1], this rounding procedure gives $F(\mathcal{A}) \geq F(x^{LP})$. Defining $J_1 = \{(i, v) : i \text{ is separated from } v \text{ by } I_1\}$, and from property 2 of the lemma:

$$F(x^{LP}) \geq \sum_{(i,v)\in J_1} p_i v_v + \left(1-(1-\frac{1}{2\delta})\right)^{2\delta} \sum_{(i,v)\in J\setminus J_1} (p_i v_v) \min\{1, \sum_{e\in P(i,v)} (y_e)^{LP} + \sum_{e\in P(i,v)} (z_e^i)^{LP}\}$$

Main Algorithm. For each set of at most three y_e, set them to 1, then find the PTAS 2nd stage decision that can be made in each scenario (no additional first

stage edges purchased), and evaluate the objective of each such solution. Take the best such solution and call it q^*.
1. For each $I_1 \subseteq Y$ such that $|I_1| = 4$ and $\sum_{i \in I_1} c_i \leq B$:

- Set $I_0 = \emptyset$.
- Set $t = 0$.
- While $t = 0$: apply $\mathcal{A}$ to LP$[I_0, I_1]$.
  1. If all the $x_i^{\mathcal{A}}$ (decision variables in either stage) are integral, then set t to 1 and set $\hat{x}$ to $x_i^{\mathcal{A}}$.
  2. Else, if $x_i^{\mathcal{A}}$ has no fractional y_e, then round up any fractional z_e^i, set t to 1 and set $\hat{x}$ to $x_i^{\mathcal{A}}$ with the rounded up second stage variables.
  3. Else, if neither of these conditions holds, round down the single fractional y_e and round up all fractional z_e^i, set $\hat{x}$ to $x_i^{\mathcal{A}}$ with the rounded variables. Also, add the index of the y_e that was rounded down to I_0.
  4. If $F(\hat{x}) > F(\bar{x})$, then set $\bar{x}$ to $\hat{x}$. *(Since $\hat{x}$ and $\bar{x}$ are integral, this chooses the highest L-value among all the $\hat{x}$ considered by the algorithm).*

Now we prove that this algorithm meets claim of Theorem 3. First observe that the algorithm spends at most $2B$ for scenario i: pipage rounding maintains budget feasibility for every scenario and the final roundings used to achieve integrality round up at most a single fractional decision variable per scenario. Our preprocessing guarantees that this single round up costs at most B in addition to the cost of the fractional solution returned by $\mathcal{A}$.

Let X^* be the optimal set of decision variables, let Y^* denote the first stage variables in X^*. If $|Y^*| \leq 3$, then step 0. finds a $(1+\epsilon)$ approximation to OPT. So, we address the case when $|Y^*| \geq 4$. W.l.o.g. we can assume that the set of decision variables is ordered such that $Y^* = \{1, ..., |Y^*|\}$ and for each $i \in Y^*$, among the elements $\{i, ..., |Y^*|\}$ the element i protects the maximum total weight of (i, v) pairs which are not already protected by the set $\{1, ..., i-1\}$.

For the iteration in which $I_1 = \{1, 2, 3, 4\}$, let q denote the number of runs of the while loop. Since each run of the while loop either terminates the iteration or sets a first stage variable to 0, q is at most $n - 4$. During the iteration the algorithm finds a series of q feasible solutions to the LP. Let I_0^j denote I_0 in the jth run of the while loop. The jth feasible solution $\hat{X}_j$ has $\hat{X}_j \cap I_0^j = \emptyset$ (from the form of the algorithm). Index the elements of I_0^q in the order that the algorithm adds them to I_0, that is, $I_0^j = \{i_1, ..., i_j\}$ where i_l is the index of the lth first stage variable added to I_0 for this iteration.

Assume first that $I_0^q \cap Y^* = \emptyset$. That is, when the iteration terminates, no first stage variables used by OPT have been forced to 0: OPT is a feasible solution for LP$[I_1, I_0^q]$. Since this is the last run of the while loop, it must have ended in an *if* statement of one of the first 2 types. In the first case: all the $x_i^{\mathcal{A}}$ (decision variables in either stage) are integral and $x_i^{\mathcal{A}}$ is the outcome of pipage rounding of the fractional optimal of LP$[I_1, I_0^q]$. In particular: since $\hat{x}$ is integral, $L(\hat{x}) = F(\hat{x}) = F(x_i^{\mathcal{A}}) \geq F(x^{LP})$. For the second case, rounding up the second-stage variables only increases the value of F, and after the rounding we have an integral solution, so $L(\hat{x}) = F(\hat{x}) \geq F(x_i^{\mathcal{A}}) \geq F(x^{LP})$. Either way, the

following inequality derived from property 2 and the fact that OPT is feasible for $\mathrm{LP}[I_1, I_0^q]$ now gives that $\hat{x}$ is a budget-balanced $(1-(1-\frac{1}{2\delta})^{2\delta})$-approximation:

$$F(x^{LP}) \geq \sum_{(i,v)\in J_1} p_i v_v + \left(1-(1-\frac{1}{2\delta})\right)^{2\delta} \sum_{(i,v)\in J\setminus J_1} (p_i v_v) \min\{1, \sum_{e\in P(i,v)} (y_e)^{LP} + \sum_{e\in P(i,v)} (z_e^i)^{LP}\}$$
$$\geq \left(1-(1-\frac{1}{2\delta})\right)^{2\delta} OPT.$$

Now, assume that $I_0^q \cap Y^* \neq \emptyset$. Let I_0^{s+1} be the first I_0 in the series $I_0^1, ..., I_0^q$ that has nonempty intersection with Y^*: the sth run of the while loop is the first run of the while loop for this iteration in which the algorithm adds a first stage variable from Y^* to I_0 (call that variable i_s). The algorithm adds i_s to I_0 after considering a solution $\hat{x}$ in which i_s was the single fractional first stage variable was rounded down (this is the third type of *if* statement in the while loop). We claim that the $\hat{x}$ that resulted when i_s was rounded down (and fractional second stage variables were rounded up) was a $(1-(1-1/2\delta)^{2\delta})$-approximation. Proving this claim will be establish Theorem 3.

As in [1], $F(X)$ defined on subsets of $Y \cup Z$ is a submodular set function. Thus, we have the *diminishing-returns* property: for any subsets R and G of $Y \cup Z$ and any element $i \in Y \cup Z$, we get $F(R \cup i) - F(R) \geq F(R \cup G \cup i) - F(R \cup G)$. Now, letting h denote a member of Y^* which is not in $\{1,2,3,4\}$, and letting H denote any superset of $\{1,2,3,4\}$:

$$1/4F(I_1) = 1/4F(1,2,3,4)$$
$$= 1/4[F(\{1,2,3,4\}) - F(\{1,2,3\}) + F(\{1,2,3\}) - F(\{1,2\}) + F(\{1,2\}) - F(\{1\}) + F(\{1\}) - F(\emptyset)]$$
$$\geq 1/4[F(\{1,2,3,h\}) - F(\{1,2,3\}) + F(\{1,2,h\}) - F(\{1,2\}) + F(\{1,h\}) - F(\{1\}) + F(\{h\}) - F(\emptyset)]$$
$$\geq F(H \cup \{h\}) - F(H).$$

The first equality results from a collapsing sum where we remove the final $+F(\emptyset)$ since it is 0 (since the tree is connected and every scenario has a source). By the labeling of the decision variables in Y^*: since h is not in $\{1,2,3,4\}$, the additional marginal value h protects beyond what is protected by any prefix of $\{1,2,3,4\}$ is at most the additional value that the index which does follow the prefix protects. Finally, we apply the diminishing-returns property 4 times to get the final inequality.

Also, as in [1], rounding up a fractional solution produced by $\mathcal{A}$ only increases the value of F. Let $x^{\mathcal{A}}$ denote the unrounded solution returned by $\mathcal{A}$. Let $I(x^{\mathcal{A}})$ be the integral positive elements of $x^{\mathcal{A}}$, let $\{j_1, ..., j_i\}$ denote the set of fractional second stage variables in $x^{\mathcal{A}}$, and i_s denote the fractional first stage variable in $x^{\mathcal{A}}$ from Y^*. Then $\hat{x}$ is $I(x^{\mathcal{A}}) \cup \{j_1, ..., j_i\}$, so we can use the integrality of $\hat{x}$ to bound its LP value as follows:

$$L(\hat{x}) = L(I(x^{\mathcal{A}}) \cup \{j_1, ..., j_i\}) = F(I(x^{\mathcal{A}}) \cup \{j_1, ..., j_i\})$$

Adding and subtracting a common quantity:

$$= F(I(x^{\mathcal{A}}) \cup \{j_1, ..., j_i\} \cup \{i_s\}) - \Big(\underbrace{F(I(x^{\mathcal{A}}) \cup \{j_1, ..., j_i\} \cup \{i_s\}) - F(I(x^{\mathcal{A}}) \cup \{j_1, ..., j_i\})}\Big)$$

Applying our bound to bracketed quantity since $I(x^{\mathcal{A}})$ contains $\{1,2,3,4\}$ and $i_s \in Y^*$:

$$\geq F(I(x^{\mathcal{A}}) \cup \{j_1, ..., j_i\} \cup \{i_s\}) - 1/4F(I_1) \geq F(x^{\mathcal{A}}) - 1/4F(I_1)$$

The second inequality holds because F increases when its argument is rounded up, and $I(x^{\mathcal{A}}) \cup \{j_1, ..., j_i\} \cup \{i_s\}$ is just $x^{\mathcal{A}}$ rounded up. Now write out $F(x^{\mathcal{A}})$:

$$= \sum_{(i,v)\in J_1} p_i v_v + \sum_{(i,v)\in J\setminus J_1} (p_i v_v)\Big[1 - \Big(\prod_{e\in P(i,v)} (1-(y_e)^{\mathcal{A}})\Big)\Big(\prod_{e\in P(i,v)} (1-(z_e^i)^{\mathcal{A}})\Big)\Big] - 1/4F(I_1)$$

$$= 3/4 \sum_{(i,v)\in J_1} p_i v_v + \sum_{(i,v)\in J\setminus J_1} (p_i v_v)\Big[1 - \Big(\prod_{e\in P(i,v)} (1-(y_e)^{\mathcal{A}})\Big)\Big(\prod_{e\in P(i,v)} (1-(z_e^i)^{\mathcal{A}})\Big)\Big]$$

Pipage rounding produces $x^{\mathcal{A}}$ from x^{LP} while increasing F:

$$\geq 3/4 \sum_{(i,v)\in J_1} p_i v_v + \sum_{(i,v)\in J\setminus J_1} (p_i v_v)\Big[1 - \Big(\prod_{e\in P(i,v)} (1-(y_e)^{LP})\Big)\Big(\prod_{e\in P(i,v)} (1-(z_e^i)^{LP})\Big)\Big]$$

Apply the well-known inequality which holds for all fractional solutions:

$$\geq 3/4 \sum_{(i,v)\in J_1} p_i v_v + (1-(1-1/2\delta)^{2\delta}) \sum_{(i,v)\in J\setminus J_1} (p_i v_v) \min\{1, \sum_{e\in P(i,v)} (y_e)^{LP} + \sum_{e\in P(i,v)} (z_e^i)^{LP}\}$$

Notice that $3/4 \geq (1-(1-1/2\delta)^{2\delta})$. Also, x^{LP} is the optimal solution for LP$[I_1, I_0^s]$ and X^* is feasible for LP$[I_1, I_0^s]$. Thus, the last quantity is bounded below by $(1-(1-1/2\delta)^{2\delta})L(X^*) = (1-(1-1/2\delta)^{2\delta})OPT$.

Suppose that the division of the budget between first and second stages is specified in the input as (B_1, B_2). Adding the additional constraints $\sum_e y_e c_e \leq B_1$ and $\sum_e z_e^i (M^{ie} c_e) \leq B_2$ for all i to the LP alters our analysis only slightly: preprocess to eliminate decision variables that are too expensive to fully buy in their corresponding stages, the algorithm now enumerates over four-member sets of first-stage decision variables, at the conclusion of the pipage phase the remaining fractional first-stage variable is rounded down (so B_1 is respected) and at most one second-stage variable per scenario is rounded up (B_2 is overspent by at most a factor of 2), first stage variables which are rounded down are excluded one by one in the iterations of the while loop. Thus, we get:

Theorem 4. *Given a specific first-stage budget B_1 and second-stage budget B_2, there exists a $(1-(1-\frac{1}{2\delta})^{2\delta})$-approximation algorithm for the 2-stage stochastic GPP in trees that respects B_1 and violates B_2 by a factor of at most 2 (each ignition set has size 1, δ denotes the diameter of the tree).*

Stochastic Single-Stage and k-Stage Results. In the limiting single-stage stochastic case (where second-stage action is prohibitively expensive) there is only a single budget constraint: the proof of Theorem 3 can be simplified so that it directly follows [1] to get:

Theorem 5. *There exists a $(1-(1-\frac{1}{\delta})^{\delta})$-approximation algorithm for the single-stage stochastic GPP in trees provided that each ignition set has size 1 (δ denotes the diameter of the tree).*

For the 2-stage stochastic GPP in trees with single ignition node, consider the algorithm that chooses the better performance between spending all of B in stage 1 vs. spending all of B in stage 2: apply Theorem 5 for stage 1 and the PTAS for deterministic single-source GPP for stage 2 assuming that the optimal solution earns $\alpha(\mathrm{OPT})$ in the first stage, and minimize over $\alpha \in (0,1)$ to get a worst case guarantee of $(0.387, 1)$. For a constant number of scenarios, use Theorem 2 in the place of Theorem 5 to get a $(.5(1-\epsilon), 1)$- approximation.

The k-stage stochastic graph protection problem in trees (for constant k) has k stages in which information is revealed and decisions about edge removal are made (rather than one or two stages). This information can be considered as updates that arrive at k specific times which condition the distribution on where the ignition will occur (by specifying that the ignition will occur among some particular subset of the nodes). For each stage the input includes a partition of the node set, and the partition for stage i refines the partition for stage $i-1$. In each stage the planner has the option to remove additional edges from the graph at some (stage, partition piece)-specific cost. A solution specifies which edges will be removed for each partition piece realization at each stage. The total cost incurred for each realized sequence of k partition pieces should be B.

Theorem 6. *For a restricted class of information revelation hierarchies, there exists a bicriteria* $(1-(1-\frac{1}{k\delta})^{k\delta}), 2+\epsilon)$*-approximation algorithm for the k-stage stochastic GPP in trees provided that each ignition set has size 1 (k is a constant,* δ *denotes the diameter of the tree).*

Theorem 6 requires that the number of partition pieces added over all stages excluding the last stage (in which any of n points may be realized) is bounded by a constant: guessing the optimal division of the budget to ϵ/k-precision for each possible information realization takes polynomial time. As in the (B_1, B_2) case: impose additional constraints based on the guess of optimal budget division, reject too-expensive decision variables, pipage round (now roundings take place between pairs of fractional variables that correspond to a common partition piece within a stage). Last, round up all fractional variables (see [6] for details).

If there is a specified budget for each of the k stages, then the guessing (enumeration) may be dropped: with no requirements on the information revelation hierarchy the same analysis gives a $(1-(1-\frac{1}{k\delta})^{k\delta})$ value-protection guarantee which violates each stage's budget by a factor of at most 2.

Reductions, Results for Stochastic Multistage MCKP. A looser $(1-\frac{1}{e})$ guarantee which matches Theorem 5 asymptotically may be obtained by reducing single-stage stochastic GPP in trees to the weighted Budgeted Separating Minimum Cut Problem in trees for which the analysis of Engelberg, et. al [2] applies: submodularity of the objective allows application of the result of Sviridenko [7]). The tighter result in Theorem 5 can alternately be proved by a more subtle reduction to MCKP addressed in [1] (reducing wBSMC in trees to MCKP gives the tighter result for wBSMC as well). Full Reductions in [6].

Maximum Coverage with a Knapsack Constraint (MCKP): Given a family $F = \{S_j : j \in J\}$ of subsets of a set $I = \{1, 2, ..., n\}$ with associated nonnegative weights w_j and costs c_j of the elements, and positive integer B, find a subset $X \subseteq I$ with $\sum_{j \in X} c_j \leq B$ so as to maximize the total weight of the sets in F having nonnegative intersections with X.

- *Stochastic MCKP:* There is also a distribution Π: each scenario specifies how much value will be received for covering the subset S_j for each j. The objective is to maximize the expected weight of subsets covered.
- *Multistage MCKP:* Elements may be purchased in different stages at a cost that is stage-, scenario-, and element-dependent (costs are specified in the input). Stochastic multistage versions of wBSMC in trees reduce to these MCKP problems.

The features of the LP we analyzed (objective function and budget constraints) also hold for the natural LPs for these problems: the analysis proving theorems 3, 4, 5, and 6 can be extended with identical guarantees to the corresponding multistage stochastic MCKP generalizations.

3 1-Stage Extension to Probabilistic Edge Transmission

In ecological fact, fuel-treated areas are not 100% burn resistant (e.g. they may burn if extreme weather arises). Also, different types of treatments (with different costs) may reduce the probability of fire passing between adjacent parcels by different amounts. These considerations motivate a version of GPP in which the input specifies a more complicated relationship between spending on each edge and the resulting transmission probability across that edge. Previously we had two options: pay 100% of the edge cost to get probability of transmission 0, or pay 0% of the edge cost to get probability of transmission 1.

To single-stage stochastic GPP where each ignition set has size 1, we add the feature that each edge has (as part of the input) a specified monotonically-decreasing step function that gives the probability of transmission across that edge as a function of the spending level (the spending level may range from 0% to 100% of the edge cost, the events of transmissions across edges are assumed to be independent). We give an approximation result assuming that the running time of the algorithm is allowed to depend polynomially on the number of steps in each step function. The objective remains to maximize the expected value protected from the ignition point, only now this expectation is over realization of both the scenario and the individual edge-transmission events that arise.

The analogous notion for MCKP is of *probabilistic element failure*: for each element there is a step function that represents the probability that the element will *fail to cover the subsets which contain it* (generalizing that an element e fails to cover subsets which contain it with probability 1 if we do nothing, and with probability 0 if we pay c_e). The objective is to maximize the expected weight of subsets covered, where this expectation is over both element and scenario realization. The generalization of wBSMC in trees to a case with probabilistic edge occurrence reduces to MCKP with *probabilistic element failure*.

Theorem 7. *There exists a* $(1-\frac{1}{e})$*-approximation algorithm for the single-stage stochastic GPP in trees with probabilistic edge transmission (provided that each ignition set has size 1). For MCKP with probabilistic element failure: there exists a* $(1-\frac{1}{e})$*-approximation algorithm.*

Proof (GPP). Each (spending level, edge) pair is an element the solution can buy with cost corresponding to the spending level times the edge cost (we only have elements corresponding to critical spending levels at which the transmission probability instantaneously drops). Let X denote the set of such elements. The expected value protected is a set function over these elements. Denote this function by E. We wish to maximize this set function by buying elements subject to a knapsack constraint: if we show that this set function is submodular, [7] will immediately yield a $(1-\frac{1}{e})$-approximation that is budget-balanced (provided that we can compute in polynomial time the element which gives largest improvement). To prove submodularity we will establish the law of diminishing returns: for an arbitrary (spending level, edge) pair denoted by a, if $A \subseteq B \subseteq X$, then $E(A \cup a) - E(A) \geq E(B \cup a) - E(B)$.

Let the edge of the (spending level, edge) pair a be denoted by e. According to the step function for e, buying a results in some probability of transmission α_i. Before a is added, A contains some set of elements which affect the transmission probability on e, and B contains a superset of these elements. Thus the probability of transmission on e is (weakly) larger for the set A than for the set B. In both cases, when a is added to a set, the new probability of transmission on e is the minimum of α_i and the current probability of transmission on e. The gap is larger for A than for B. Let $\wp_e(\cdot)$ denote the probability of transmission on e as a function of the set of elements: $\wp_e(A) \geq \wp_e(B) \Rightarrow \wp_e(A) - \wp_e(A \cup a) \geq \wp_e(B) - \wp_e(B \cup a)$.

Next, focus on a particular (ignition point, node) pair (i, v). If the path from i to v does not contain e, then adding e does not change the (i, v)th term in the expression for expected value protected. If the path from i to v does contain e, for each non-e edge on this i to v path, the probability of transmission under A is at least the probability of transmission under B. Let $P(Q)$ denote the probability that every edge on the i to v path (excluding e) transmits under Q:

$$
\begin{aligned}
&P(A) \geq P(B) \Rightarrow P(A)(\wp_e(A) - \wp_e(A \cup a)) \geq P(B)(\wp_e(B) - \wp_e(B \cup a)) \Rightarrow \\
&\quad \Rightarrow P(A)\wp_e(A) - P(A)\wp_e(A \cup a) \geq P(B)\wp_e(B) - P(B)\wp_e(B \cup a). \\
&\quad \Rightarrow (1 - P(A)\wp_e(A \cup a)) - (1 - P(A)\wp_e(A)) \geq (1 - P(B)\wp_e(B \cup a)) - (1 - P(B)\wp_e(B)). \\
&\quad \Rightarrow E(A \cup a) - E(A) \geq E(B \cup a) - E(B).
\end{aligned}
$$

The third line compares the changes in probability that v is protected from i which result when a is added to A and when a is added to B. The final inequality follows from summing change in expected valued protected over (ignition point, node) pairs (including pairs for which the addition of e caused no change). This establishes submodularity. Computing the change in E resulting from the addition of a single element simply requires computing the product along the (ignition

point, node) path twice for each (i, v) pair. This takes polynomial time for each of polynomially-many elements.□.(Similar MCKP analysis in [6]).

Extensions to General Graphs

Single-Stage. Theorems 1, 2, 5 can be extended to bicriteria approximations for general graphs: the guarantees on value protected (expected value protected) are identical, and the budget is violated by a $O(\log n)$-factor. As in Engelberg, et. al [2] we apply the result of Räcke [5] on cut-capacity approximation: approximate the costs graph by a distribution over tree graphs (whose maximum diameter is n), solve a suitably modified instance in each tree, translate solutions back to the original graph, select the best solution.

Multi-stage. If the number of scenarios is bounded by a constant, then Theorems 3, 4, and 6 can be extended to general graphs: the guarantees on expected value protected are identical, but the budget(s) is violated by a $O(\log n)$-factor. To apply the result of Räcke [5] to the multistage case we need that some tree produced by the cut-capacity approximation has $O(\log n)$-distortion *for the optimal solution in every scenario* (not just for a single set of edges purchased in the first stage). Details in [6].

References

1. Ageev, A.A., Sviridenko, M.: Pipage rounding: A new method of constructing algorithms with proven performance guarantee. J. Comb. Optim. 8(3), 307–328 (2004)
2. Engelberg, R., Könemann, J., Leonardi, S., Naor, J(S.): Cut Problems in Graphs with a Budget Constraint. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 435–446. Springer, Heidelberg (2006)
3. Finney, M.A.: A computational method for optimising fuel treatment locations. International Journal of Wildland Fire 16, 702–711 (2007)
4. Hayrapetyan, A., Kempe, D., Pál, M., Svitkina, Z.: Unbalanced Graph Cuts. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 191–202. Springer, Heidelberg (2005)
5. Räcke, H.: Optimal hierarchical decompositions for congestion minimization in networks. In: 40th STOC, pp. 255–264 (2008)
6. Shmoys, D., Spencer, G.: Full paper: Approximation algorithms for fragmenting a graph against a stochastically-located threat, http://people.orie.cornell.edu/gms39/images/documents/gsfragment.pdf
7. Sviridenko, M.: A note on maximizing a submodular set function subject to a knapsack constraint. Operations Research Letters 32, 41–43 (2004)
8. Wei, Y., Rideout, D., Kirsch, A.: An opt. model for locating fuel treatments across a landscape to reduce expected fire losses. Can. J. of Forest Res. 38, 868–877 (2008)

Non-clairvoyant Weighted Flow Time Scheduling on Different Multi-processor Models

Jianqiao Zhu, Ho-Leung Chan*, and Tak-Wah Lam**

University of Hong Kong, Hong Kong
{jqzhu,hlchan,twlam}@cs.hku.hk

Abstract. We study non-clairvoyant scheduling to minimize weighted flow time on two different multi-processor models. In the first model, processors are all identical and jobs can possibly be speeded up by running on several processors in parallel. Under the non-clairvoyant model, the online scheduler has no information about the actual job size and degree of speed-up due to parallelism during the execution of a job, yet it has to determine dynamically when and how many processors to run the jobs. The literature contains several $O(1)$-competitive algorithms for this problem under the unit-weight multi-processor setting [9,10] as well as the weighted single-processor setting [2]. This paper shows the first $O(1)$-competitive algorithm for weighted flow time in the multi-processor setting.

In the second model, we consider processors with different functionalities and only processors of the same functionality can work on the same job in parallel to achieve some degree of speed up. Here a job is modeled as a sequence of non-clairvoyant demands of different functionalities. This model is derived naturally from the classical job shop scheduling; but as far as we know, there is no previous work on scheduling to minimize flow time under this multi-processor model. In this paper we take a first step to study non-clairvoyant scheduling on this multi-processor model. Motivated by the literature on 2-machine job shop scheduling, we focus on the special case when processors are divided into two types of functionalities, and we show a non-clairvoyant algorithm that is $O(1)$-competitive for weighted flow time.

1 Introduction

We study online scheduling of jobs with varying importance (weight). Jobs arrive online and have arbitrary arrival times, weights and sizes. We consider the non-clairvoyant model in which a scheduler only knows the existence of a job and its weight when it arrives, the size information is known only at the time when the job is completed. We assume preemptive scheduling. This model is natural from the viewpoint of operating systems. The flow time of a job is the length

* Ho-Leung Chan is supported in part by GRF Grant HKU 710210E.
** Tak-Wah Lam is supported in part by HKU Grant 201007176149.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 137–149, 2012.

of the duration from its arrival until its completion. We study the objective of minimizing the total weighted flow time of all jobs. When there is only one processor, the problem is well-understood. It is known that no algorithm can be $O(1)$-competitive even when all weights are 1 (i.e., the unweighted case) [12]. Kalyanasundaram and Pruhs [11] proposed analyzing the online algorithms when they are given a slightly faster processor. For the unweighted case, they showed that SETF is $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive [11]. Later, Bansal and Dhamdhere [2] extended the result to the weighted case and showed that an algorithm WSETF is $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive for total weighted flow time.

In this paper, we study non-clairvoyant scheduling to minimize weighted flow time on two different multi-processor models.

- **Model 1:** *Homogeneous processors and jobs with varying parallelizability.* In this model, there are $m \geq 1$ identical processors and jobs can be processed by any one processor. Following [9,10], we further consider jobs that can be speeded up by running on more than one processors in parallel, and in general, each job may have an arbitrary degree of speed up due to parallelism at different times. That is, each job consists of a number of phases each with an arbitrary size and an arbitrary degree of speed-up when running on different numbers of processors. Non-clairvoyant scheduling under such multi-processor model has been well studied when the jobs are unweighted; in particular, Edmonds showed that a simple non-clairvoyant algorithm Equi is $(2+\epsilon)$-speed $O(1/\epsilon)$-competitive [9] for minimizing total flow time, and later Edmonds and Pruhs gave an improved algorithm LAPS that is $(1+\beta+\epsilon)$-speed $O(1/\beta\epsilon)$-competitive [10]. The results of Equi and LAPS are non-trivial even though they are restricted to unweighted jobs; the difficulty arises from the fact that they cannot assume any information about a phase during the execution of a job. An open problem is whether these results can be generalized to jobs with arbitrary weights.

- **Model 2:** *Heterogeneous processors and jobs demanding different functionalities.* In this model, we consider processors divided into different functionalities and only processors of the same functionality can run the same job in parallel to achieve some degree of speed up. Here a job is modeled as a sequence of non-clairvoyant demands of different functionalities, targeting different groups of processors at different times. This model is derived naturally from the classical job shop scheduling (see [4] for a survey); but as far as we know, there is no previous work on scheduling to minimize total flow time under this multi-processor model. In this paper we take a first step to study non-clairvoyant scheduling on this multi-processor model. Motivated by the literature on 2-machine job shop scheduling (e.g., [7,1,3] for minimizing makespan, [8] for total completion time), we focus on the special case when processors are divided into two types of functionalities. We hope to

devise a non-clairvoyant algorithm that is $O(1)$-speed $O(1)$-competitive for weighted flow time.

Our Results. In this paper, we have derived new results on non-clairvoyant scheduling in the above multi-processor models. For Model 1 (homogeneous processors), we analyze the algorithm WLAPS (which is a weighted version of LAPS) and show that it is $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive for total weighted flow time. It shows that an algorithm can be competitive with any tiny amount of extra speed. Technically speaking, WLAPS was first proposed by [6] in the context of single processor scheduling to minimize weighted flow time plus energy usage under the dynamic speed scaling model. WLAPS divides the processing power to the latest arrival jobs according to their weights, and it was shown that WLAPS is $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive in that context. To adapt WLAPS to the multi-processor setting of Model 1, the main difficulty is that when jobs have unknown parallelizability, the actual processing rate of the jobs can be arbitrary and cannot be controled by the online algorithm. When we try to adapt the potential function analysis of WLAPS [6], it becomes difficult to bound the processing progress or the change of potential. To this end, we adopt the technique from LAPS [10] by classifying the work done on a job according to whether it fully utilizes the processors assigned. We mark a portion of work as *unsaturated* if it fully utilizes the processors and mark the rest *saturated*. We can analyze the unsaturated work by the potential function of WLAPS, while we can charge the weighted flow time of the saturated work directly to the optimal offline schedule.

For Model 2 (heterogeneous processors), we consider a natural extension of WLAPS for the case with two different functionality types, which we call 2WLAPS. We show that 2WLAPS is $O(1)$-competitive given constantly faster processors. Specifically, we show 2WLAPS to be s-speed $\frac{4s}{(\sqrt{s}-2)^2-2}$-competitive for any $s > (2+\sqrt{2})^2$. To ease the discussion, we call the functionalities type-1 and type-2 functionalities. We also call a processor a type-1 or type-2 processor depending on the functionality it can provide. Roughly speaking, 2WLAPS again focuses on the latest arrived jobs. Among these latest arrived jobs, some are requiring type-1 functionalities. 2WLAPS divides the processing power of all type-1 processors to these jobs proportional to their weight. 2WLAPS schedules the jobs requiring type-2 functionalities similarly. The analysis of 2WLAPS is interesting. We bound the total weighted flow time of 2WLAPS by the total weighted flow time of the optimal offline schedule plus the total weighted flow time incurred when a job is requiring type-1 functionality. Symmetrically, we bound the total weighted flow time of 2WLAPS by the total weighted flow time of the optimal offline schedule plus the total weighted flow time incurred when a job is requiring type-2 functionality. Summing the two bounds and rearranging the terms gives the required result.

We also show a better algorithm for the special case when there is only one processor for each of the two functionality types. The algorithm is $2(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive for weighted flow time.

2 Formal Problem Definitions

This section gives the formal definitions of the problems. We always consider online jobs, where each job j has an arbitrary release time $r(j)$ and a weight $w(j)$ known at arrival time. The jobs are preemptive and migratory.

In Model 1, we are given a set of m identical processors. Each job j consists of a sequence of phases, where the k-th phase of j, denoted j^k, has a size $p(j^k)$ and a speed-up function Γ_{j^k}, which specifies the amount of speed-up when running the job with multiple processors. When j^k is processed by y processors each of speed s, the processing rate of the k-th phase is $\Gamma_{j^k}(y) \cdot s$. The size of j, denoted $p(j)$, is the sum of $p(j^k)$ over all phases of j. The jobs are non-clairvoyant, i.e., the number of phases and the size and speed-up function of each phase are unknown to the algorithm during the execution of the job. We need an online algoithm that, at any time, determines which subset of jobs to be processed and the number of processors assigned to each job. The objective is to minimize the total weighted flow time of all jobs.

We give more details on the speed-up functions. Similar to [9,10,5], we assume that each speed-up function Γ is non-negative, monotonically increasing and sublinear, i.e., $\frac{\Gamma(y)}{y} \geq \frac{\Gamma(y')}{y'}$ for any $y \leq y'$. A phase is *fully-parallelizable* if its speed-up function satisfies that $\Gamma(y) = y$ for all y. A phase is *parallel up to n* processors if $\Gamma(y) = y$ for all $y \leq n$ and $\Gamma(y) = n$ for $y > n$. We assume that for any phase, its speed-up function Γ satisfies that $\Gamma(y) = y$ for $y \in [0, 1]$. This assumption corresponds to the fact that when a phase is processed by time-sharing a $y \leq 1$ fraction of a processor, its processing rate should be y times the speed of the processor.

In Model 2, we assume two types of functionalities, which we call type-1 and type-2 functionalities. We call a processor type-1 or type-2 depending on the functionality it provides. We are given m_1 type-1 processors and m_2 type-2 processors. Let $m = m_1 + m_2$. Each job j consists of a sequence of phases, where the k-th phase of j, denoted j^k, has a known functionality requirement, a size $p(j^k)$ and a speed-up function Γ_{j^k}. The jobs are non-clairvoyant, i.e., the number of segments and the size and speed-up function of each phase are unknown to the algorithm until the job is completed. A phase requiring type-1 functionality (resp., type-2 functionality) can only be processed by type-1 processors (resp., type-2 processors). At any time, an algorithm decides the set of jobs to be processed and the number of processors assigned to each job. The objective is again to minimize the total weighted flow time of all jobs.

We denote Opt as the optimal offline algorithm which always minimizes the objective for any input and any model. For any algorithm A, denote $W_A(I)$ as the total weighted flow time when I is scheduled by A.

3 Analyzing WLAPS for Homogeneous Processors

This section analyzes the algorithm WLAPS [6] when it is used to schedule jobs with varying parallelizability on homogeneous processors. It also serves as a warm

up and the proof structure will be extended to the heterogeneous processors setting in the next section. We first state the definition of WLAPS.

> **Algorithm WLAPS(β).** WLAPS is parameterized by a constant $\beta \in (0, 1]$. At any time t, let $n_a(t)$ be the number of unfinished jobs remaining in WLAPS and $w_a(t)$ be their total weight. Let $\{j_1, j_2, \ldots, j_{n_a(t)}\}$ be the unfinished jobs ordered in increasing order of arrival times. Let r be the largest integer such that the latest arrived jobs $j_r, j_{r+1}, \ldots, j_{n_a(t)}$ have a total weight at least $\beta w_a(t)$. Define the *adjusted weight* of j_i at time t, denoted $w'(j_i, t)$, as follows. For $i < r$, $w'(j_i, t) = 0$; for $i > r$, $w'(j_i, t) = w(j_i)$; and $w'(j_r, t) = \beta w_a(t) - \sum_{i=r+1}^{n_a(t)} w(j_i)$. WLAPS shares the processing power among all unfinished jobs proportional to its adjusted weight.

Denote the set $\{j_r, j_{r+1}, \ldots, j_{n_a(t)}\}$ as $R(t)$. Note that only jobs in $R(t)$ have non-zero adjusted weight and WLAPS only processes jobs in $R(t)$. When there are m homogeneous processors, WLAPS assigns a share of $\frac{w'(j_i,t)}{\beta w_a(t)} m$ processors to each job $j_i \in R(t)$. The main result of this section is that by putting $\beta = \frac{\epsilon}{2(1+\epsilon)}$, WLAPS($\beta$) is $(1+\epsilon)$-speed $8(1+\frac{1}{\epsilon})^2$-competitive. The analysis is as follows.

3.1 Restricting the Input Instance

We first show that for any non-clairvoyant algorithm, the worst-case competitive ratio occurs when each phase of a job is either fully parallelizable or parallel up to one processor. The analysis is similar to that in [9,10], but they are concerned with flow time instead of weighted flow time, and they reduce each phase to fully-parallelizable or "special" in the sense that the processing rate is 1 regardless of the number of processors assigned to that phase.

Lemma 1. *Let Alg be any non-clairvoyant algorithm and I be any job instance. We can construct another job instance I' where each phase is either fully-parallelizable or parallel up to one processor such that $W_{Alg}(I') = W_{Alg}(I)$ and $W_{Opt}(I') \leq W_{Opt}(I)$.*

We omit the proof of Lemma 1 here. Instead, in the next section, we prove a more general lemma for the setting of heterogeneous processors, which will imply this lemma. To analyze WLAPS, for the rest of this section, we focus on an instance I where each phase is fully-parallelizable or parallel up to one processor. We will omit the parameter I from the notations when it is clear. For example, we write W_{Opt} to mean $W_{Opt}(I)$.

3.2 A Lower Bound on W_{Opt}

We can show that for any instance I, we can find a certain set of jobs so that the weighted flow time of Opt is at least the weighted flow time of WLAPS incurred on this set of jobs. Details are as follows. Consider the schedule of I as defined

by WLAPS. For any job j, we divide its work into chunks, each classified as saturated or or unsaturated as follows. We mark a chunk of work as *unsaturated* if it is in a fully-parallelizable phase of j, or if it is in a phase that is parallel up to one processor and WLAPS processes the work with at most one processor. We mark other work as *saturated*. Note that WLAPS processes the unsaturated work at a rate of $p(1+\epsilon)$, where p is the number of processors assigned to the work; for saturated work, the processing rate by WLAPS is always $(1+\epsilon)$.

Consider any time t. Let $j_i \in R(t)$ be one of the jobs currently processed by WLAPS. For the sake of analysis, we assume we know the entire schedule of I as defined by WLAPS, and we can figure out how much of the unfinished work of j_i at time t will be classified as unsaturated later. Thus, we can define $q_a(j_i, t)$ to be the amount of unfinished unsaturated work in j_i in WLAPS. Let $q_o(j_i, t)$ be that in Opt. We say that j_i is lagging at time t if $q_a(j_i, t) > q_o(j_i, t)$.

Lemma 2. *Let $L(t) \subseteq R(t)$ be the set of jobs j such that WLAPS is processing the unsaturated work of j and j is lagging at time t.*

$$\int_0^\infty \sum_{j \in R(t)/L(t)} w'(j,t)\, dt \leq 2\ W_{Opt}$$

Proof. For any time t, we can partition $R(t)$ into three disjoint sets: Let $S(t) \subseteq R(t)$ be the set of jobs such that WLAPS is processing its saturated work. $L(t) \subseteq R(t)$ is the set of jobs such that WLAPS is processing its unsaturated work and are lagging. Let $N(t)$ be the set of jobs such that WLAPS is processing its unsaturated work and are non-lagging. Note that $R(t) = S(t) \cup L(t) \cup N(t)$. Then

$$\sum_{j \in R(t)/L(t)} w'(j,t) \leq \sum_{j \in R(t)/L(t)} w(j) = \sum_{j \in S(t)} w(j) + \sum_{j \in N(t)} w(j)$$

Consider any job j in $S(t)$. Note that WLAPS is processing j at a rate of $(1+\epsilon)$. Let $T_a(j)$ denote the union of time intervals during which WLAPS is processing the saturated work of j. Let $T_o(j)$ denote the union of time intervals during which Opt is processing the saturated work of j. Note that Opt can process the saturated work with rate at most one. Hence,

$$||T_a(j)|| \leq \text{total amount of saturated work in } j \leq ||T_o(j)||$$

where $||T_a(j)||$ and $||T_o(j)||$ denote the total length of time intervals in the corresponding set. Therefore, $\int_0^\infty \sum_{j \in S(t)} w(j) dt = \sum_{j \in I} ||T_a(j)|| \cdot w(j) \leq \sum_{j \in I} ||T_o(j)|| \cdot w(j) \leq W_{Opt}$ The last inequality comes from the fact that the weighted flow time of Opt on j is at least $||T_o(j)|| \cdot w(j)$.

For $N(t)$, note that any job j in $N(t)$ is non-lagging at time t. Since j is unfinished in WLAPS at time t, j is also unfinished in Opt. Let $w_o(t)$ be the total weight of unfinished jobs in Opt at time t. Then, $\sum_{j \in N(t)} w(j) \leq w_o(t)$ and $\int_0^\infty \sum_{j \in N(t)} w(j) dt \leq \int_0^\infty w_o(t) dt = W_{Opt}$.

Combining the analysis for $S(t)$ and $N(t)$, the lemma follows. □

3.3 Potential Function Analysis

We are now ready to bound the total weighted flow time of WLAPS using a potential function analysis. Our target is to define a potential function Φ satisfying the following three conditions.

- *Boundary condition.* $\Phi(t) = 0$ before any job is released and after all jobs are completed.
- *Discrete-event condition.* $\Phi(t)$ does not increase when a job arrives or when a job is completed by WLAPS or Opt.
- *Running condition.* At any time t where there is no job arrival or completion,

$$w_a(t) + \frac{d\Phi(t)}{dt} \leq 4(1+\frac{1}{\epsilon})^2 \sum_{j \in R(t)/L(t)} w'(j,t) \ . \tag{1}$$

Then, by integrating (1), we have $W_{WLAPS} \leq 4(1+\frac{1}{\epsilon})^2 \int_0^\infty \sum_{j \in R(t)/L(t)} w'(j,t)dt$.

We define $\Phi(t)$ as follows. At any time t, recall that $\{j_1, j_2, \ldots, j_{n_a(t)}\}$ are the unfinished jobs in WLAPS in increasing order of arrival times. Let $c_i(t) = \sum_{k=1}^{i} w(j_k)$. Let $x_i(t) = \max\{0, q_a(j_i,t) - q_o(j_i,t)\}$, i.e., $x_i(t)$ is the amount of work done in unsaturated work that WLAPS is lagging behind Opt. Then,

$$\Phi(t) = \gamma \sum_{i=1}^{n_a(t)} c_i(t) \cdot x_i(t)$$

where γ is a constant which will be set to $\frac{2}{m\epsilon}$ later. We can check that the boundary and discrete-event conditions are satified. For the running condition, let $\frac{d\Phi_a(t)}{dt}$ and $\frac{d\Phi_o(t)}{dt}$ be the rate of change of $\Phi(t)$ due to the processing of WLAPS and Opt, respectively. Then, $\frac{d\Phi(t)}{dt} = \frac{d\Phi_a(t)}{dt} + \frac{d\Phi_o(t)}{dt}$.

Lemma 3. *At any time t,* **(i)** $\frac{d\Phi_a(t)}{dt} \leq -\frac{\gamma m}{\beta}(1-\beta)(1+\epsilon)\sum_{j \in L(t)} w'(j,t)$, *and* **(ii)** $\frac{d\Phi_o(t)}{dt} \leq \gamma m w_a(t)$.

Proof. For (i), note that for each job $j_i \in L(t) \subseteq R(t)$, $c_i(t) \geq (1-\beta)w_a(t)$ and $x_i(t) > 0$. Consider $\Phi(t)$. For each term $\gamma c_i(t)x_i(t)$, if $j_i \in L(t)$, the term is decreasing at a rate of $\gamma c_i(t)\frac{w'(j,t)}{\beta w_a(t)}m(1+\epsilon)$ due to the processing of WLAPS; else if $j_i \notin L$, the term is non-increasing. Note that $\gamma c_i(t)\frac{w'(j,t)}{\beta w_a(t)}m(1+\epsilon) \geq \gamma(1-\beta)w_a(t)\frac{w'(j,t)}{\beta w_a(t)}m(1+\epsilon)$. Hence, $\frac{d\Phi_a(t)}{dt} \leq -\frac{\gamma m}{\beta}(1-\beta)(1+\epsilon)\sum_{j \in L(t)} w'(j,t)$.

For (ii), the worst case occurs when Opt is processing the job the largest $c_i(t)$ at the maximum speed. Since $c_i(t) \leq w_a(t)$ and Opt has only m 1-speed processors, $\frac{d\Phi_o(t)}{dt} \leq \gamma m w_a(t)$. □

Lemma 4. *Let $\beta = \frac{\epsilon}{2(1+\epsilon)}$ and $\gamma = \frac{2}{m\epsilon}$. Then,*

$$w_a(t) + \frac{d\Phi(t)}{dt} \leq 4(1+\frac{1}{\epsilon})^2 \sum_{j \in R(t)/L(t)} w'(j,t)$$

Proof. Note that $w_a(t) + \frac{d\Phi_o(t)}{dt} \leq (1+\gamma m)w_a(t) = \frac{2+\epsilon}{\epsilon} w_a(t)$. Since $\beta w_a(t) = \sum_{j \in R(t)} w'(j,t)$, we have $w_a(t) + \frac{d\Phi_o(t)}{dt} \leq \frac{2+\epsilon}{\beta\epsilon} \sum_{j \in R(t)} w'(j,t)$. Next note that $-\frac{\gamma m}{\beta}(1-\beta)(1+\epsilon) = -\frac{2}{\beta\epsilon}(1 - \frac{\epsilon}{2(1+\epsilon)})(1+\epsilon) = -\frac{2+\epsilon}{\beta\epsilon}$. We have $\frac{d\Phi_a(t)}{dt} \leq -\frac{\gamma m}{\beta}(1-\beta)(1+\epsilon) \sum_{j \in L(t)} w'(j,t) = -\frac{2+\epsilon}{\beta\epsilon} \sum_{j \in L(t)} w'(j,t)$.

Summing the two inequalities, we have $w_a(t) + \frac{d\Phi_o(t)}{dt} + \frac{d\Phi_a(t)}{dt}$ is at most

$$\frac{2+\epsilon}{\beta\epsilon} \sum_{j \in R(t)/L(t)} w'(j,t) \;\leq\; 4(1+\frac{1}{\epsilon})^2 \sum_{j \in R(t)/L(t)} w'(j,t)$$

□

We conclude with the main result of this section.

Theorem 1. *Let $\beta = \frac{\epsilon}{2(1+\epsilon)}$. WLAPS($\beta$) is $(1+\epsilon)$-speed $8(1+\frac{1}{\epsilon})^2$-competitive.*

Proof. By integrating the both sides of Lemma 4 and using the result of Lemma 2, we have

$$W_{WLAPS} \;\leq\; 4(1+\frac{1}{\epsilon})^2 \sum_{j \in R(t)/L(t)} w'(j,t)\,dt \;\leq\; 8(1+\frac{1}{\epsilon})^2 \cdot W_{Opt}$$

□

4 An $O(1)$-Competitive Algorithm for Two Functionality Types

This section considers non-clairvoyant scheduling with two different types of functionalities. Recall that we have m_1 type-1 processors and m_2 type-2 processors. We call a job a *type-1* job (resp., a *type-2* job) at time t if its functionality requirement at time t is of type 1 (resp., of type 2). We consider a natural extension of WLAPS which we call 2WLAPS.

> **2WLAPS(β).** Let $\beta \in (0,1]$ be any constant. We first disregard the functionality requirements and calculate the adjusted weight of each job identically as WLAPS. Specifically, let $n_a(t)$ be the total number of unfinished jobs in 2WLAPS at time t at let $w_a(t)$ be their total weight. Let $\{j_1, j_2, \ldots, j_{n_a(t)}\}$ be the unfinished jobs ordered in increasing order of arrival times and $R(t) = \{j_r, j_{r+1}, \ldots, j_{n_a(t)}\}$ be the smallest set of latest arrival jobs with total weight at least $\beta w_a(t)$. Define the adjusted weight $w'(j_i,t)$ of j_i at time t identically as WLAPS.
>
> We only make use of the functionality information when we share the processing power. Specifically, let $R_1(t) \subseteq R(t)$ be the set of type-1 jobs in $R(t)$. We share the processing power of the m_1 type-1 processors among the jobs in $R_1(t)$ proportional to their adjusted weights. Define $R_2(t)$ and share the processing power of the type-2 processors similarly.

Note that we may idle all processors of a certain type if $R(t)$ contains no job of that type. The main result of this section is that 2WLAPS(β), when $\beta = 1 - \frac{1}{\sqrt{s}}$, is s-speed $\frac{4s}{(\sqrt{s}-2)^2-2}$-competitive, for any $s > (2+\sqrt{2})^2$.

4.1 Restricting the Input Instance

Similar as the previous section, we first show that for any non-clairvoyant algorithm, the worst-case competitive ratio occurs when each phase is either fully-parallelizable or parallel up to one processor.

Lemma 5. *Consider any number of functionality types. Let Alg be any non-clairvoyant algorithm and I be any job instance. We can construct another job instance I' where each phase is either fully parallelizable or parallel up to 1 processor such that $W_{Alg}(I') = W_{Alg}(I)$ and $W_{Opt}(I') \leq W_{Opt}(I)$.*

Proof. We convert each job j in I into another job j' in I'. By changing both the size and speed-up function of j at the same time, any non-clairvoyant algorithm, without knowledge of both, will schedule the resulting job j' identically as j. We will guarantee that Opt can schedule j' no worse than j, so the lemma follows. Note that the functionality of the job is unchanged. Details are as follows.

Consider any job j. We show how to construct j' based on j. Consider a infinitesimal chunk of work in j with size Δ and speed-up function Γ. Let p_a and p_o be the number of processors assigned by Alg and Opt, respectively, to process Δ. We define a chunk of work in j' with size Δ' and speed-up function Γ' based on p_a and p_o. The functionality requirement is unchanged.

If $p_a < p_o$, we set $\Delta' = \frac{p_a}{\Gamma(p_a)}\Delta$ and Γ' to be fully-parallelizable. Note that for Alg, both the amount of work and the processing rate are increased by a same factor $\frac{p_a}{\Gamma(p_a)}$, so the schedule in Alg is unaffected. For Opt, it will finish the work with $\Delta'/p_o = \frac{p_a}{\Gamma(p_a)\cdot p_o}\Delta$ unit of time. Since Γ is sublinear, we have $\frac{p_a}{\Gamma(p_a)} \leq \frac{p_o}{\Gamma(p_o)}$ for $p_a < p_o$. So $\Delta'/p_o = \frac{p_a}{\Gamma(p_a)\cdot p_o}w \leq \frac{p_o}{\Gamma(p_o)\cdot p_o}\Delta = \Delta/\Gamma(p_o)$. So the processing time in Opt will not increase.

If $p_a > p_o$, we set $\Delta' = \min\{\frac{1}{\Gamma(p_a)}, 1\}\Delta$ and Γ' to be parallel up to one processor. For Alg, still both the amount of work and the processing rate are decreased by a same factor, so the schedule in Alg is unaffected. For Opt, it will have processing rate $\min\{p_o, 1\}$ on Δ' and will finish the work with $\Delta'/\min\{p_o, 1\} = \frac{\min\{\frac{1}{\Gamma(p_a)},1\}}{\min\{p_o,1\}}\Delta$ time. Note that since $p_a > p_o$ we have $\min\{\frac{1}{\Gamma(p_a)}, 1\} \leq \min\{\frac{1}{\Gamma(p_o)}, 1\}$, and since Γ is sublinear we have $\min\{p_o, 1\} \geq \min\{\Gamma(p_o), 1\}$. Thus $\Delta'/\min\{p_o, 1\} \leq \frac{\min\{\frac{1}{\Gamma(p_o)},1\}}{\min\{\Gamma(p_o),1\}}\Delta = \Delta/\Gamma(p_o)$. The processing time in Opt will not increase. □

4.2 A Lower Bound of Opt

We derive a lower bound on the total weighted flow time of Opt. Consider any job j. Note that it is still well-defined to partition the work of j as unsaturated or saturated as before. I.e., we mark those work as *unsaturated* if it is in a fully-parallelizable phase of j, or if it is in a phase that is parallel up to one processor but 2WLAPS processes the work with at most 1 processor. We mark the other work as *saturated*. Hence, we can partition the work of j into four different parts depending on whether it is type-1 or type-2 and whether it is saturated or unsaturated.

Defining the notion of lagging jobs is slightly more complicated and needs some new notations. Consider any time t. Denote $q_{1,a}(j_i,t)$ the amount of unfinished type-1 unsaturated work remaining in j_i in WLAPS at time t. Define $q_{1,o}(j_i,t)$ similarly for Opt. We say that j_i is *lagging in type-1 work* at time t if $q_{1,a}(j_i,t) > q_{1,o}(j_i,t)$. Conversely, j_i is non-lagging in type-1 work at time t $q_{1,a}(j_i,t) \leq q_{1,o}(j_i,t)$. We can define $q_{2,a}(j_i,t)$, $q_{2,o}(j_i,t)$ and lagging for type-2 work similarly. Note that job can be lagging in type-1 work and non-lagging in type-2 at the same time.

Lemma 6. *Recall that $R_1(t)$ and $R_2(t)$ are the sets of type-1 and type-2 jobs, respectively, in $R(t)$. Let $L_1(t) \subseteq R_1(t)$ be the set of jobs j such that 2WLAPS is processing the type-1 unsaturated work of j and j is lagging in type-1 work.*

$$\int_0^\infty \sum_{j \in R(t)/L_1(t)} w'(j,t)dt \leq 2\ W_{Opt} + \int_0^\infty \sum_{j \in R_2(t)} w(j)dt$$

Proof. At any time t, we can partition $R_1(t)$ into three disjoint sets: Let $S_1(t) \subseteq R_1(t)$ be the set of jobs such that 2WLAPS is processing its saturated work. $L_1(t) \subseteq R_1(t)$ is the set of jobs such that 2WLAPS is processing its unsaturated work and are lagging in type-1 work. Let $N_1(t) \subseteq R_1(t)$ be the set of jobs such that 2WLAPS is processing its unsaturated work and are non-lagging in type-1 work. Note that $R_1(t) = S_1(t) \cup L_1(t) \cup N_1(t)$. Then,

$$\sum_{j \in R(t)/L_1(t)} w'(j,t) \leq \sum_{j \in R(t)/L_1(t)} w(j) = \sum_{j \in S_1(t)} w(j) + \sum_{j \in N_1(t)} w(j) + \sum_{j \in R_2(t)} w(j)$$

Similar as the proof of Lemma 2, we observe that for each job j, the amount of time that 2WLAPS spent on processing the saturated type-1 work of j is at most that of Opt. Hence, $\int_0^\infty \sum_{j \in S_1(t)} w(j)dt \leq W_{Opt}$. Similarly, at any time t, each job j in $N_1(t)$ is unfinished in Opt, so $\int_0^\infty \sum_{j \in N_1(t)} w(j)dt \leq W_{Opt}$. Hence, by integrating the above inequality from time 0 to time infinity, the lemma follows. □

4.3 Potential Function Analysis

Finally, we bound the total weighted flow time of 2WLAPS by a potential function analysis. Our target is to prove the following lemma.

Lemma 7. *Let $\beta = 1 - \frac{1}{\sqrt{s}}$ be the parameter given to 2WLAPS.*

$$W_{2WLAPS} \leq \frac{s}{(\sqrt{s}-1)^2} \int_0^\infty \sum_{j \in R(t)/L_1(t)} w'(j,t)dt$$

where $s > (2+\sqrt{2})^2$ is the speed of the processors given to 2WLAPS.

We first show that this would imply our main result on 2WLAPS.

Theorem 2. *Let $\beta = 1 - \frac{1}{\sqrt{s}}$. 2WLAP($\beta$) is s-speed $\frac{4s}{(\sqrt{s}-2)^2-2}$-competitive for scheduling with two functionality types.*

Proof. Using Lemma 6 and 7, we have that

$$W_{2WLAPS} \leq \frac{s}{(\sqrt{s}-1)^2}\left(2W_{Opt} + \int_0^\infty \sum_{j \in R_2(t)} w(j)dt\right)$$

Note that the previous analysis is symmetric we can swap the role of type-1 and type-2 work to prove that $W_{2WLAPS} \leq \frac{s}{(\sqrt{s}-1)^2}\left(2W_{Opt} + \int_0^\infty \sum_{j \in R_1(t)} w(j)dt\right)$. By summing up the two inequalities and noticing that $\int_0^\infty \sum_{j \in R_1(t)} w(j)dt + \int_0^\infty \sum_{j \in R_2(t)} w(j)dt \leq W_{2WLAPS}$, we have that

$$2W_{2WLAPS} \leq \frac{s}{(\sqrt{s}-1)^2}\left(4W_{Opt} + W_{2WLAPS}\right)$$

The theorem follows by rearranging the terms. □

It remains to prove Lemma 7 using a potential function analysis. We define a potential function $\Phi_1(t)$ as follows. At any time t, recall that $\{j_1, j_2, \ldots, j_{n_a(t)}\}$ are the unfinished jobs in 2WLAPS and $c_i(t) = \sum_{k=1}^{i} w(j_k)$. We define $x_{1,i}(t) = \max\{0, q_{1,a}(j_i, t) - q_{1,o}(j_i, t)\}$ be the amount of type-1 unsaturated work that 2WLAPS is lagging behind Opt. Then,

$$\Phi_1(t) = \gamma_1 \sum_{i=1}^{n_a(t)} c_i(t) \cdot x_{1,i}(t)$$

where γ_1 is a constant which will be set to $\frac{1+\sqrt{s}}{(s-1)m_1}$. Note that $\Phi_1(t)$ satisfies the boundary condition and the discrete-event condition. It remains to prove the running condition that at any time t,

$$w_a(t) + \frac{d\Phi_1(t)}{dt} \leq \frac{s}{(\sqrt{s}-1)^2} \sum_{j \in R(t)/L_1(t)} w'(j,t) \tag{2}$$

We prove the running condition in two lemmas. Let $\frac{d\Phi_{1,a}(t)}{dt}$ and $\frac{d\Phi_{1,o}(t)}{dt}$ be the rates of change of $\Phi_1(t)$ due to the action of 2WLAPS and Opt, respectively.

Lemma 8. *At any time t, (i) $\frac{d\Phi_{1,a}(t)}{dt} \leq -\frac{\gamma_1 m_1}{\beta}(1-\beta)s \sum_{j \in L_1(t)} w'(j,t)$, and (ii) $\frac{d\Phi_{1,o}(t)}{dt} \leq \gamma_1 m_1 w_a(t)$.*

Proof. For (i), note that for each job $j_i \in L_1(t) \subseteq R(t)$, $c_i(t) \geq (1-\beta)w_a(t)$ and $x_{1,i}(t) > 0$. Furthermore, 2WLAPS assigns $\frac{w'(j_i,t)}{\sum_{j \in R_1(t)} w'(j,t)} m_1 \geq \frac{w'(j_i,t)}{\beta w_a(t)} m_1$ processors to j_i. Consider $\Phi_1(t)$. For each term $\gamma_1 c_i(t) x_{1,i}(t)$, if $j_i \in L_1(t)$, the term is decreasing at a rate of at least $\gamma_1 c_i(t) \frac{w'(j,t)}{\beta w_a(t)} m_1 s$ due to the processing of

2WLAPS; else if $j_i \notin L$, the term is non-increasing. Note that $\gamma_1 c_i(t) \frac{w'(j,t)}{\beta w_a(t)} m_1 s \geq \gamma_1(1-\beta) w_a(t) \frac{w'(j,t)}{\beta w_a(t)} m_1 s$. Hence, $\frac{d\Phi_{1,a}(t)}{dt} \leq -\frac{\gamma_1 m_1}{\beta}(1-\beta)s \sum_{j \in L_1(t)} w'(j,t)$.

For (ii), the worst case occurs when Opt is processing the job the largest $c_i(t)$ at the maximum speed. Since $c_i(t) \leq w_a(t)$ and Opt has only m_1 type-1 processors of speed 1, $\frac{d\Phi_o(t)}{dt} \leq \gamma_1 m_1 w_a(t)$. □

Putting $\beta = 1 - \frac{1}{\sqrt{s}}$ and $\gamma_1 = \frac{1+\sqrt{s}}{(s-1)m_1}$ into Lemma 8 and summing the terms $w_a(t)$, $\frac{d\Phi_{1,a}(t)}{dt}$ and $\frac{d\Phi_{1,o}(t)}{dt}$, the running condition (2) follows. Lemma 7 follows by integrating (2) over all period of times.

4.4 A Better Competitive Algorithm with $m_1 = m_2 = 1$

When there is only one processor for each functionality, we can have a simple and better competitive algorithm as follows. Assume the online algorithm is given one $2(1+\epsilon)$-speed processor for each functionality type. The online algorithm can consider them as *one* processor of speed $2(1+\epsilon)$ that can process work of both types. That is, at any time t, if we want to process a set $I(t)$ of jobs using this $2(1+\epsilon)$-speed processor, we can process each job in $I(t)$ using a processor of the corresponding functionality.

Hence, an online algorithm can simply run WLAPS on this $2(1+\epsilon)$-speed processor. We can also assume that the optimal offline schedule is given one 2-speed processor that can process work of both types, instead of two 1-speed heterogeneous processors, and it may only decrease the total weighted flow time of the optimal offline schedule. Then, this becomes a single processor scheduling problem. By Theorem 1, WLAPS with a $2(1+\epsilon)$-speed processor is $8(1+\frac{1}{\epsilon})^2$-competitive against the optimal algorithm with a 2-speed processor. Hence, the overall algorithm is $2(1+\epsilon)$-speed $8(1+\frac{1}{\epsilon})^2$-competitive.

References

1. Anderson, E.J., Jayram, T.S., Kimbrel, T.: Tighter bounds on preemptive job shop scheduling with two machines. Computing 67(1), 83–90 (2001)
2. Bansal, N., Dhamdhere, K.: Minimizing weighted flow time. ACM Transactions on Algorithms 3(4) (2007)
3. Bansal, N., Kimbrel, T., Sviridenko, M.: Job shop scheduling with unit processing times. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 207–214 (2005)
4. Brucker, P.: Job-shop scheduling problem. In: Floudas, C.A., Pardalos, P.M. (eds.) Encyclopedia of Optimization, pp. 1782–1788. Springer, Heidelberg (2009)
5. Chan, H.-L., Edmonds, J., Pruhs, K.: Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In: SPAA, pp. 1–10 (2009)
6. Chan, S.-H., Lam, T.-W., Lee, L.-K.: Non-Clairvoyant Speed Scaling for Weighted Flow Time. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 23–35. Springer, Heidelberg (2010)
7. Chen, B., Vestjens, A.P.A., Woeginger, G.J.: On-line scheduling of two-machine open shops where jobs arrive over time. J. Comb. Optim. 1(4), 355–365 (1998)

8. Della Croce, F., Narayan, V., Tadei, R.: The two-machine total completion time flow shop problem. European Journal of Operational Research 90(2), 227–237 (1996)
9. Edmonds, J.: Scheduling in the dark. Theor. Comput. Sci. 235(1), 109–141 (2000)
10. Edmonds, J., Pruhs, K.: Scalably scheduling processes with arbitrary speedup curves. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 685–692 (2009)
11. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM 47(4), 617–643 (2000)
12. Motwani, R., Phillips, S., Torng, E.: Non-clairvoyant scheduling. Theor. Comput. Sci. 130(1), 17–47 (1994)

A New Perspective on List Update: Probabilistic Locality and Working Set

Reza Dorrigiv and Alejandro López-Ortiz

Cheriton School of Computer Science, University of Waterloo,
Waterloo, Ont., N2L 3G1, Canada
{rdorrigiv,alopez-o}@uwaterloo.ca

Abstract. In this paper we study the performance of list update algorithms under arbitrary distributions that exhibit strict locality of reference and prove that Move-to-Front (MTF) is the best list update algorithm under any such distribution. Furthermore, we study the working set property of online list update algorithms. The working set property indicates the good performance of an online algorithm on sequences with locality of reference. We show that no list update algorithm has the working set property. Nevertheless, we can distinguish among list update algorithms by comparing their performance in terms of the working set bound. We prove bounds for several well known list update algorithms and conclude that MTF attains the best performance in this context as well.

1 Introduction

The list update problem is one of the most studied online problems. It was first studied by McCabe [25] more than 45 years ago in the context of maintaining a sequential file. Since then, various list update algorithms have been proposed (e.g., [14,29,19,9,31,22,33,28,5,2]) and different aspects of the problem have been studied (e.g., [20,27,24,6,3,18]). Despite this, there still are various interesting aspects of the problem not yet explored. In this paper we aim to provide new insights for the list update problem by studying the performance of list update algorithms under probabilistic and deterministic inputs with locality of reference.

Consider an unsorted list L of ℓ items. An online list update algorithm $\mathcal{A}$ is a strategy for reordering the elements of L after each access. The input to the algorithm is an *access sequence* $X = \langle x_1, x_2, \ldots, x_m \rangle$ that must be served in an online manner. To serve a request to an item x_j, $\mathcal{A}$ linearly searches the list until it finds x_j. If x_j is the i-th item in the list, $\mathcal{A}$ incurs a cost i to access x_j. Immediately after this access, $\mathcal{A}$ can move x_j to any position closer to the front of the list at no extra cost. This is called a *free exchange*. Also $\mathcal{A}$ can exchange any two consecutive items at a cost of 1. These are called *paid exchanges*. An efficient algorithm can thus use free and paid exchanges to minimize the overall cost of serving a sequence.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 150–163, 2012.

Three well known deterministic online algorithms for list update are *Move-To-Front* (MTF), *Transpose*, and *Frequency-Count* (FC). MTF moves the requested item to the front of the list whereas Transpose exchanges the requested item with the item that immediately precedes it. FC maintains an access count for each item ensuring that the list always contains items in non-increasing order of frequency of access. *Timestamp* (TS) is an efficient list update algorithm introduced by Albers [2]. After accessing an item a, TS inserts a in front of the first item b that is before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorganize the list.

In the early stages, list update algorithms were analyzed using the distributional or average-case model (e.g. [25,14,29,9,19]). In this model, the request sequences are generated according to a probability distribution and the efficiency of an algorithm is related to the expected cost it incurs. According to this model, FC is the best online list update algorithm, followed by Transpose and TS, and finally MTF. In contrast, in some real-life applications of list update, e.g., data compression [10,13,17], MTF has the best performance among these algorithms, and Transpose and FC have much worse performance than MTF and TS. This inconsistency can be explained by the fact that sequences for list update usually exhibit *locality of reference* [20,11] and online list update algorithms try to take advantage of this property [20,28]. A sequence has high locality of reference if a recently accessed item is more likely to be accessed in the near future. Summarizing experimental results on list update, Albers and Lauer [3] conclude that the performance and ranking of list update algorithms depend on the amount of locality in the input. In addition, it has been commonly assumed, based on intuition and experimental evidence, that MTF is the best algorithm on sequences with high locality of reference, e.g., Hester and Hirschberg [20] claim: "move-to-front performs best when the list has a high degree of locality" (see also [4], page 327). Although this was observed more than twenty years ago [20], only recently some theoretical models for list update with locality of reference have been proposed [6,3,16]. These models show the superiority of MTF to other online list update algorithms on sequences with high locality of reference.

However, to the best of our knowledge, no probabilistic model for list update with locality of reference has been proposed so far. We introduce such a model by refining the distributional analysis using the diffuse adversary model of Koutsoupias and Papadimitriou [23]. More specifically, we restrict the "acceptable" probability distributions to those with high locality of reference. So far, the diffuse adversary model has only been applied to paging algorithms [23,8]. Under this model we prove the superiority of MTF and the non-optimality of static list update algorithms. Furthermore we show that the performance of MTF improves as the amount of locality increases.

We also study the *working set* property [32] of list update algorithms. The working set property is based on the idea that an operation on a recently accessed item should take less time. The working set property of most other self-organizing data structures has been studied before [32,12,21]. We show that although no

list update algorithm has the working set property, their performance can be expressed in terms of the working set bound. Our analysis shows that MTF is the best list update algorithm in this setting. Considering the connection between the working set property and locality, this result confirms (yet again) that MTF is the best online list update algorithm on sequences with high locality of reference.

2 List Update with Locality of Reference

In this section we refine the distributional model for analysis of list update algorithms by incorporating locality. First we provide more details about the distributional model and review the known results in this model. Let $L = (a_1, a_2, \ldots, a_\ell)$ be the list of items and $p = (p_1, p_2, \ldots, p_\ell)$ be a vector of positive probabilities with $\sum_{i=1}^{n} p_i = 1$. At each step, item a_i is requested with probability p_i. For a list update algorithm $\mathcal{A}$, let $E_{\mathcal{A}}(p)$ be the asymptotic expected cost of $\mathcal{A}$ in serving a single request in a request sequence generated by p. Traditionally, the performance of online list update algorithms was compared to that of the optimal static ordering, SOPT. SOPT knows the probability distribution and initially rearranges the items in non-increasing order of their probabilities and does not change their order afterwards. By the strong law of large numbers we have $E_{\mathrm{FC}}(p) = E_{SOPT}(p)$ for any p [29]. For MTF, Chung et al. [15] showed that for any probability distribution p, $E_{\mathrm{MTF}}(p) \leq (\pi/2)E_{SOPT}(p)$ and Gonnet et al. [19] showed that this bound is tight. Transpose outperforms MTF in this model: Rivest [29] proved that for any distribution p, we have $E_{Transpose}(p) \leq E_{\mathrm{MTF}}(p)$. For TS, we have $E_{\mathrm{TS}}(p) \leq (1.34)E_{SOPT}(p)$ for any probability distribution p [4].

Therefore, FC is the best online list update algorithm in this model, followed by Transpose and TS, and finally MTF. As stated before, this is not consistent with experimental results and one apparent reason for this is the fact that the model does not incorporate locality of reference assumptions. In this section we analyze list update algorithms under probability distributions with locality of reference and show that MTF outperforms other algorithms under this model. Our model is based on the *diffuse adversary* model, in which we restrict the set of "acceptable" probability distributions.

Definition 1. *[23] Let $\mathcal{A}$ be an online algorithm for a minimization problem and let Δ be a class of distributions over the input sequences. Then $\mathcal{A}$ is c-competitive against Δ, if there exists a constant b, such that*

$$E_{\sigma \in D}[\mathcal{A}(\sigma)] \leq c \cdot E_{\sigma \in D}[OPT(\sigma)] + b,$$

for every distribution $D \in \Delta$, where $\mathcal{A}(\sigma)$ denotes the cost of $\mathcal{A}$ on the input sequence σ and $E[\,]$ denotes the expectation under D.

We model locality of reference by considering a class Δ of sequences that exhibit locality of reference. Let $L = (a_1, a_2, \ldots, a_\ell)$ be the list of items and σ be a

sequence of requests to the items. We define $p(a_i, \sigma)$, the probability of accessing item a_i after the sequence σ, in a way that reflects locality of reference. The idea is to favour recently accessed items. Let the *age* of an item a_i in a sequence σ, denoted by $age(a_i, \sigma)$, be j if a_i is the j-th most recently accessed item in σ. To handle the case that an item is not requested in σ, we assume that all sequences are prepended by the sequence $a_\ell, a_{\ell-1}, \ldots, a_1$. For example, for the empty sequence ε we have $age(a_i, \varepsilon) = i$. Observe that the items have unique ages between 1 and ℓ, i.e., the set of ages at each time is exactly $\{1, 2, \ldots, \ell\}$. Now we define probability of accessing a_i after σ in terms of the age of a_i in σ: $p(a_i, \sigma) = f(age(a_i, \sigma))$, where the non-increasing function f is a probability distribution on $\{1, 2, \ldots, \ell\}$. Observe that in contrast to the traditional probabilistic models for list update, we consider a dynamic probability distribution on items. By requiring f to be non-increasing we ensure that more recently accessed items are more probable, thus reflecting the locality of reference assumption. Furthermore, we can measure the amount of locality of such probability distributions. Define a random variable X_f such that $X_f = x$ with probability $f(x)$. The expected value of X_f, $E[X_f] = \sum_{i=1}^{\ell} i \cdot f(i)$ can be considered as a measure for the amount of non-locality of reference: if $E[X_f]$ is small then we know that the probability of requesting most recently accessed items is much higher than accessing the rest of items. We also require $f(i) > 0$ for $1 \leq i \leq \ell$ to ensure that all items can be accessed. The following examples show different possible amounts of locality.

1. Consider the probability distribution $f_1(i) = 1/\ell$ for $1 \leq i \leq \ell$. Intuitively, f_1 does not have much locality and we have $E[X_{f_1}] = \sum_{1 \leq i \leq \ell} (i/\ell) = (\ell+1)/2$, which is a relatively large number. Actually, this is the largest $E[X_f]$ for non-increasing probability distributions on $\{1, 2, \ldots, \ell\}$.
2. Consider the probability distribution f_2 for which $f_2(i+1) = f_2(i)/2$, i.e., the probability of accessing an item is halved as its age is increased by one unit. It can be proved that $E[X_{f_2}] < 2$, so f_2 has constant expected value and high amount of locality.
3. Let f_3 be the Zipfian distribution $f_3(i) = \alpha/i$, i.e., probability of accessing an item is inversely proportional to its age. We have

$$\sum_{i=1}^{\ell} f_3(i) = 1 \Rightarrow \sum_{i=1}^{\ell} \frac{\alpha}{i} = 1 \Rightarrow \alpha H_\ell = 1 \Rightarrow \alpha = 1/H_\ell,$$

where H_ℓ is the ℓ-th Harmonic number. We have $f_3(i) = \frac{1}{iH_\ell}$ and

$$E[X_{f_3}] = \sum_{i=1}^{\ell} i \cdot f_3(i) = \sum_{i=1}^{\ell} i \cdot \frac{1}{iH_\ell} = \frac{\ell}{H_\ell}.$$

Thus the expected value of f_3 is between the expected values of f_1 and f_2.

We computed the empirical probability of accessing items in terms of their ages in the files of Calgary Corpus [34] and Canterbury Corpus [1], which are the

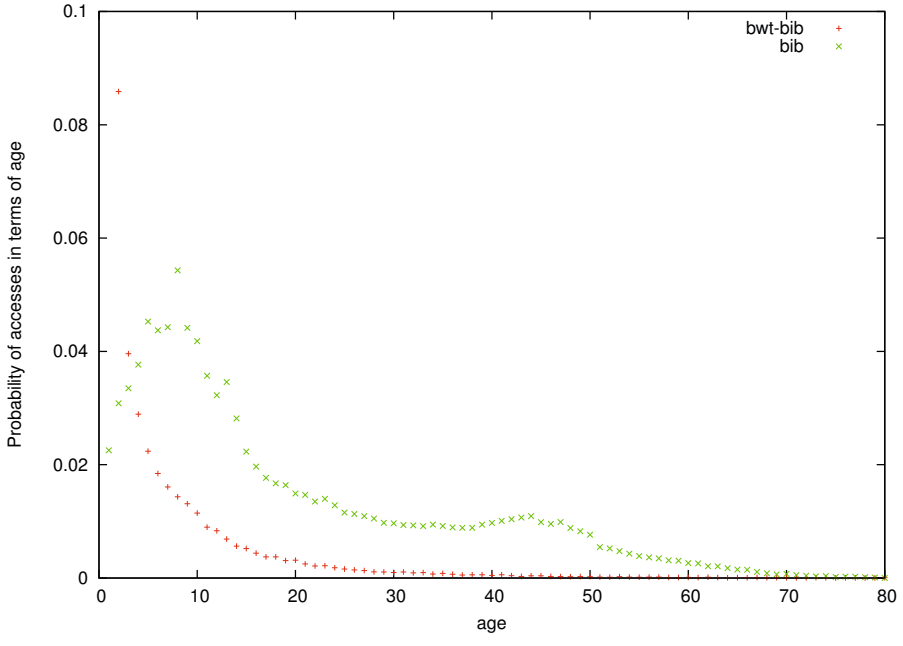


Fig. 1. Probability of accessing items in terms of their age in file **bib** before and after BWT. The probability of accessing the youngest age after BWT (0.67) is off-scale and thus not shown.

standard benchmarks for data compression. As stated before list update algorithms are widely used in data compression. These two corpora include files of different types such as English text (technical writing, poetry, fiction and non-fiction books), source code in various programming languages, picture files, object code, and spreadsheets. We computed the empirical probabilities for files before and after *Burrows-Wheeler Transform* (BWT). The Burrows-Wheeler transform (BWT) rearranges a string of symbols to one of its permutations that is believed to have more locality of reference. Then list update algorithms are used to encode this transform. The well known compression program bzip2 [30] is based on the BWT. The results for files *bib*, *progp*, and *trans* in Calgary Corpus are shown in Figures 1-3. The results for other files are very similar and can be found in the journal version of the paper. Observe from these results that after BWT the probability distribution f is non-increasing and has a very low $E[X_f]$, i.e., high locality, while before BWT the function is increasing at some intervals and has a much higher expected value. This confirms our intuition that BWT increases the amount of locality.

Observe that the probability of accessing a particular item changes over time in our model. Thus $E_{\mathcal{A}}(p)$ could be different at different times and we need to incorporate time in the definition. We define $E^t_{\mathcal{A}}(f)$ to be the expected cost of $\mathcal{A}$ in serving the t-th request in a sequence generated under f. It is not obvious whether $E^t_{\mathcal{A}}(f)$ converges as $t \to \infty$. We define $E_{\mathcal{A}}(f)$ to be $\lim_{t\to\infty} E^t_{\mathcal{A}}(f)$ if the

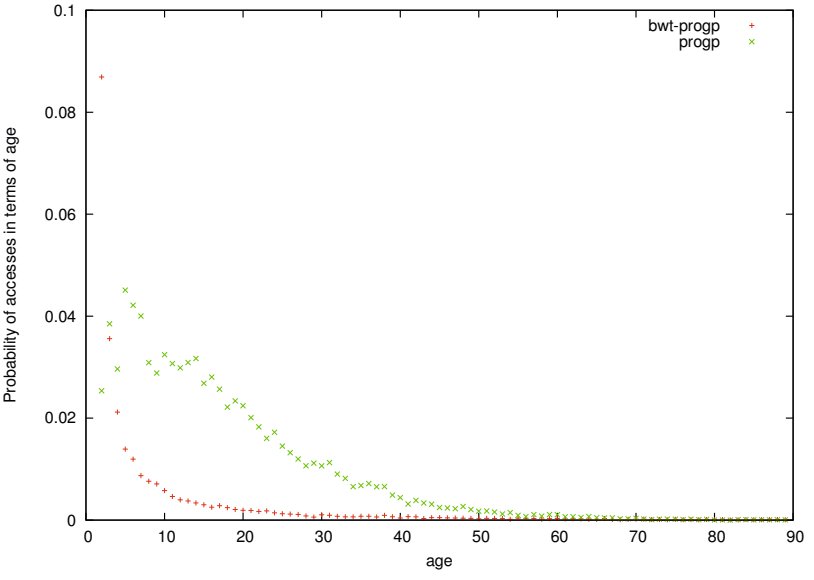


Fig. 2. Probability of accessing items in terms of their age in file **progp** before and after BWT. The probability of accessing the youngest age after BWT (0.74) is off-scale and thus not shown.

corresponding limit exists. Observe that MTF maintains the items in decreasing order of their ages. Thus the cost of MTF on an item is exactly the age of that item and we have $E^t_{\mathrm{MTF}}(f) = E[X_f]$ for every t. Therefore $E_{\mathrm{MTF}}(f) = E[X_f]$. If we have high locality of reference, then $E[X_f]$ is small and the expected cost of MTF will be low. Hence, MTF has good performance on sequences with locality of reference as expected. Good performance of MTF in this model is due to the fact that MTF tries to take advantage of locality by moving younger items to the front of its list. On the other hand, static strategies (algorithms that do not change the positions of items over the time) do not adapt to locality and so we expect them not to be optimal in the new model even if they know the distribution. This intuition is formalized in the following Lemma.

Lemma 1. *Let $\mathcal{A}$ be a static list update algorithm. Then there exists a non-increasing function f such that $E_{\mathcal{A}}(f) > E_{\mathrm{MTF}}(f)$, even if $\mathcal{A}$ knows f.*

Proof. Define the function f as follows: $f(1) = 0.9$ and $f(i) = 0.1/(\ell - 1)$ for $2 \leq i \leq \ell$. We have

$$E_{\mathrm{MTF}}(f) = 0.9 \times 1 + \frac{0.1}{\ell - 1}(2 + 3 + \cdots + \ell) = \ell/20 + 1.$$

$\mathcal{A}$ can rearrange the list $(a_1, a_2, \ldots, a_\ell)$ at the beginning and then cannot change the order of the items. Since a_1 is the youngest item at the beginning it seems reasonable for $\mathcal{A}$ to leave a_1 at the front of the list. So assume that this is the case. We have $E^1_{\mathcal{A}}(f) = \ell/20 + 1$ and so $\mathcal{A}$ has the same expected cost as MTF

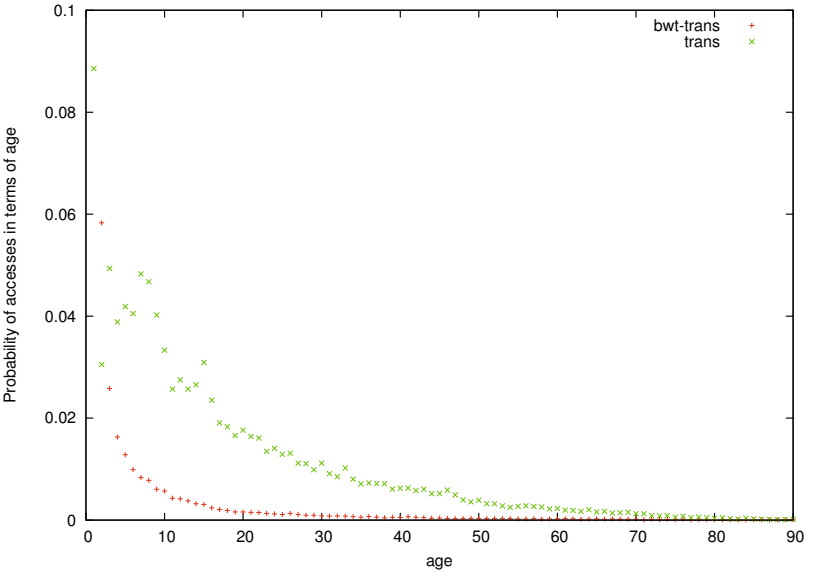


Fig. 3. Probability of accessing items in terms of their age in file **trans** before and after BWT. The probability of accessing the youngest age after BWT (0.79) is off-scale and thus not shown.

on the first request. In order to compute the asymptotic performance of $\mathcal{A}$, we define a two-state Markov chain as follows. We are in state A if we have a request to a_1 and we are in state B otherwise. If we are at state A, the probability of staying at A is 0.9 and the probability of going to B is 0.1. If we are at B, the probability of going to A is $0.1/(\ell - 1)$ and the probability of staying at B is $1 - 0.1/(\ell - 1)$. Let $[q_1 \quad q_2]$ be the stationary distribution of the Markov chain. We have

$$q_1 \times (-0.1) + q_2 \times \frac{0.1}{\ell - 1} = 0 \Rightarrow q_1 = \frac{q_2}{\ell - 1}.$$

Furthermore, $q_1 + q_2 = 1$ and so $q_1 = 1/\ell$ and $q_2 = 1 - 1/\ell$. Therefore asymptotically we have

$$E_{\mathcal{A}}(f) = (1/\ell) \times 1 + (1 - 1/\ell) \times \frac{2 + 3 + \cdots + \ell}{\ell - 1} = \frac{\ell + 1}{2}.$$

Thus the asymptotic expected cost of $\mathcal{A}$ is 10 times more than that of MTF.

Thus MTF outperforms any static strategy. Actually, we can show that the performance of a static list update algorithm is the same for any function f.

Theorem 1. *Let $\mathcal{A}$ be a static list update algorithm and f be an arbitrary probability distribution on $\{1, 2, \ldots, \ell\}$. We have $E_{\mathcal{A}}(f) = \frac{\ell+1}{2}$.*

Proof. We define a Markov chain based on f. We sort items by their age and consider a single state for any permutation of $(a_1, a_2, \ldots, a_\ell)$. So originally we are at

state $(a_1, a_2, \ldots, a_\ell)$. From this state we move to state $(a_2, a_1, a_3, \ldots, a_\ell)$ with probability $f(2)$, to state $(a_3, a_1, a_2, a_4, \ldots, a_\ell)$ with probability $f(3)$,..., and to state $(a_\ell, a_1, \ldots, a_{\ell-1})$ with probability $f(\ell)$. The Markov chain has $l!$ states and we remain in the same state with probability $f(1)$. Let M be the transition matrix of the Markov chain and $[q_1 \ q_2 \ \ldots \ q_{\ell!}]$ be its stationary distribution. Since we have $f(i) > 0$ for $1 \leq i \leq \ell$, this Markov chain is irreducible and aperiodic. Consider an arbitrary state $(a_{i_1}, a_{i_2}, \ldots, a_{i_\ell})$. We move to this state from $(a_{i_2}, a_{i_1}, \ldots, a_{i_\ell})$ with probability $f(2)$, from $(a_{i_2}, a_{i_3}, a_{i_1}, \ldots, a_{i_\ell})$ with probability $f(3)$,..., and from $(a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}, a_{i_1})$ with probability $f(l)$. Thus the column corresponding to this state in M sums to $f(1) + f(2) + \cdots + f(\ell) = 1$. Therefore M is doubly stochastic. It is known (e.g., [26, page 157]) that the stationary distribution of doubly stochastic matrices is the uniform distribution. Therefore we have $q_j = \frac{1}{\ell!}$ for $1 \leq j \leq \ell!$. Let $(a_{j_1}, a_{j_2}, \ldots, a_{j_\ell})$ be the static list maintained by $\mathcal{A}$. a_{j_k} appears as the i-th item in exactly $(\ell - 1)!$ states of the Markov chain, for $1 \leq i \leq \ell$. If we are in such a state, the probability of accessing a_{j_k} is $f(i)$. Thus the asymptotic probability of accessing a_{j_k} is $\sum_{i=1}^{\ell} (\ell - 1)! \cdot \frac{1}{\ell!} \cdot f(i) = \frac{1}{\ell}(f(1) + f(2) + \cdots + f(\ell)) = 1/\ell$. The cost of $\mathcal{A}$ on a_{j_k} is k and the asymptotic expected cost of $\mathcal{A}$ is $\sum_{k=1}^{\ell} k/\ell = (\ell + 1)/2$.

This theorem shows that the performance of static strategies does not improve by increasing the amount of locality. For instance, for the probability distribution f_2 defined above we have $E_{\mathcal{A}}(f_2) = (\ell + 1)/2$ while $E_{\mathrm{MTF}}(f_2)$ is a constant smaller than 2. Next we prove that MTF has the best possible performance and cannot be beaten by any other strategy.

Lemma 2. *Let $\mathcal{A}$ be a list update algorithm, $t > 0$, and f be a non-increasing probability distribution. We have $E^t_{\mathrm{MTF}}(f) \leq E^t_{\mathcal{A}}(f)$.*

Proof. Let σ be an arbitrary sequence of length $t - 1$ and $(a_1, a_2, \ldots, a_\ell)$ be the list maintained by MTF after serving σ. We know that $Pr[a_i|\sigma] \geq Pr[a_{i+1}|\sigma]$, i.e., after requesting σ the probability of requesting a_i is at least the probability of requesting a_{i+1}, for $1 \leq i \leq \ell - 1$. Therefore we have

$$\sum_{i=1}^{\ell} (Pr[a_i|\sigma] \times \mathrm{MTF}^t(\sigma \odot a_i)) \leq \sum_{i=1}^{\ell} (Pr[a_i|\sigma] \times \mathcal{A}^t(\sigma \odot a_i)),$$

where $\mathcal{A}^t(\sigma \odot a_i)$ denotes the cost incurred by $\mathcal{A}$ in serving the t-th request of $\sigma \odot a_i$, i.e., the sequence obtained by appending a_i to σ. Observe that we have $\mathrm{MTF}^t(\sigma \odot a_i) = i$. Since this holds for any sequence σ of length $t - 1$, we conclude that $E^t_{\mathrm{MTF}}(f) \leq E^t_{\mathcal{A}}(f)$.

3 Working Set Property for List Update

In this section we study the performance of list update algorithms in terms of the *working set bound*. Consider the access sequence $X = \langle x_1, x_2, \cdots, x_m \rangle$. The working set number of an item z at time i, $t_i(z)$, is the number of distinct

items that are requested since the last request to z (including z) or the number of distinct items that are requested so far if this is the first access to z. The working set bound of X is defined as $WS(X) = \sum_{i=1}^{m} \log(t_i(x_i) + 1)$.[1] If the total cost of X in a data structure is $O(WS(X))$ (or equivalently, the amortized cost of x_i is $O(\log(t_i(x_i) + 1))$ we say that data structure has the working set property. Observe that we have $t_i(x_i) = age(x_1 x_2 \dots x_{i-1}, x_i)$ and so there is a close relationship between the working set bound and the probabilistic model for locality in the previous section.

As stated before, list update algorithms are used in data compression. As noted in [17], the cost model is different in this case: the cost of encoding an item in position i is $\Theta(\log i)$. It is not hard to see that under this logarithmic cost model MTF has the working set property as $t_i(x_i)$ is the position of x_i in the list maintained by MTF at time i. In contrast, the following lemma shows that no online list update algorithm in the standard cost model has the working set property.

Lemma 3. *Let $\mathcal{A}$ be an online list update algorithm. There is an access sequence X such that $\mathcal{A}(X) \geq \frac{\ell}{\log(\ell+1)} \cdot WS(X)$.*

Proof. Consider an access sequence X of length m obtained by requesting the item that is in the last position of list maintained by $\mathcal{A}$ at each time. We have $\mathcal{A}(X) = m \cdot \ell$. Also we have $t_i(x_i) \leq \ell$ for $1 \leq i \leq m$, because we do not have more than ℓ distinct items. Therefore $WS(X) \leq m \cdot \log(\ell + 1)$, and $\mathcal{A}(X) \geq \frac{WS(X)}{\log(\ell+1)} \cdot \ell$.

However we can still rank list update algorithms by comparing how far their performance is from the working set bound. First we prove a general upper bound.

Lemma 4. *Let $\mathcal{A}$ be an online list update algorithm. For any access sequence X we have $\mathcal{A}(X) \leq \ell \cdot WS(X)$.*

Proof. Consider an arbitrary sequence X of length m. Since the maximum cost that $\mathcal{A}$ incurs on a request is ℓ, we have $\mathcal{A}(X) \leq m \cdot \ell$. At the same time clearly $t_i(x_i) \geq 1$ for any i. Thus we have $WS(X) \geq m \cdot \log 2 = m$ which implies $\mathcal{A}(X) \leq \ell \cdot WS(X)$.

The following lemma shows that MTF achieves the best possible performance in terms of the working set bound.

Lemma 5. *For any access sequence X we have* $\mathrm{MTF}(X) \leq \frac{\ell}{\log(\ell+1)} \cdot WS(X)$.

Proof. Consider an arbitrary access sequence X of length m. We have

$$\frac{\mathrm{MTF}(X)}{WS(X)} = \frac{\sum_{i=1}^{m} t_i(x_i)}{\sum_{i=1}^{m} \log(t_i(x_i) + 1)},$$

[1] In this paper all logarithms are base 2.

where $1 \leq t_i(x_i) \leq \ell$. Since the terms in the numerator grow exponentially compared to terms in the denominator, this expression takes its maximum when we have $t_i(x_i) = \ell$ for $1 \leq i \leq m$. Therefore

$$\frac{\mathrm{MTF}(X)}{WS(X)} \leq \frac{\sum_{i=1}^{m} \ell}{\sum_{i=1}^{m} \log(\ell+1)} = \frac{m \cdot \ell}{m \cdot \log(\ell+1)} = \frac{\ell}{\log(\ell+1)},$$

which implies $\mathrm{MTF}(X) \leq \frac{\ell}{\log(\ell+1)} \cdot WS(X)$.

Other list update algorithms do not behave optimally in terms of the working set bound.

Theorem 2. *In the worst case we have*
a) $Transpose(X) \geq \frac{\ell}{\log 3} \cdot WS(X)$.
b) $\mathrm{FC}(X) \geq \frac{\ell+1}{2} \cdot WS(X)$.
c) $\mathrm{TS}(X) \geq \frac{2\ell}{\log(\ell+1)+1} \cdot WS(X)$

Proof. a) Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list. Consider a sequence X of length m obtained by several repetitions of the pattern $a_\ell a_{\ell-1}$. Then $Transpose(X) = m \cdot \ell$. Observe that $t_1(x_1) = 1$ and $t_i(x_i) = 2$ for $2 \leq i \leq m$. Therefore $WS(X) = 1 + \sum_{i=2}^{m} \log(2+1) = 1 + (m-1) \cdot \log 3$, and

$$\frac{Transpose(X)}{WS(X)} = \frac{m \cdot \ell}{1 + (m-1) \cdot \log 3} \geq \frac{m \cdot \ell}{m \cdot \log 3} = \frac{\ell}{\log 3}.$$

b) Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list and k be an arbitrary integer. Consider the following access sequence: $X = a_1^k a_2^{k-1} a_3^{k-2} \ldots a_\ell^{k-\ell+1}$. On serving X, FC does not change the order of items in its list and incurs cost

$$\sum_{i=1}^{\ell} (k-i+1) \times i = \frac{k \cdot \ell(\ell+1)}{2} + \frac{\ell(1-\ell^2)}{3}.$$

We have $WS(X) = (k-1) \cdot \log 2 + (k-2) \cdot \log 2 + \cdots + (k-\ell) \cdot \log 2 + \sum_{i=2}^{\ell+1} \log i = k\ell - \ell(\ell+1)/2 + \sum_{i=2}^{\ell+1} \log i$. Therefore

$$\frac{\mathrm{FC}(X)}{WS(X)} = \frac{k\ell(\ell+1)/2 + l(1-\ell^2)/3}{k\ell - \ell(\ell+1)/2 + \sum_{i=2}^{\ell+1} \log i}.$$

Since k can be selected to be arbitrary larger than ℓ, we get

$$\frac{\mathrm{FC}(X)}{WS(X)} \geq \frac{k\ell(\ell+1)/2}{k\ell} = \frac{\ell+1}{2}.$$

c) Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list and k be an arbitrary integer. Consider the access sequence X obtained by repeating k times the block $a_\ell^2 a_{\ell-1}^2 \ldots a_1^2$.

Let B be such a block in X. Each item a_i is accessed twice in B. TS does not move a_i after its first access in B, because all other items have been accessed twice since the last access to a_i. After the second access, TS moves the item to the front of the list. Therefore each access is to the last item of the list and TS incurs a cost of ℓ on each access. We have $\mathrm{TS}(X) = 2k \cdot \ell^2$. Next we compute $WS(X)$. The first and second access to a_i in block B contributes $\log(\ell+1)$ and $\log 2$ to $WS(X)$, respectively. Considering the special case of the first block, we have $WS(X) = \ell + \sum_{i=2}^{\ell+1} \log i + (k-1) \cdot \ell(\log(\ell+1) + \log 2)$. Therefore

$$\frac{\mathrm{TS}(X)}{WS(X)} = \frac{2k \cdot \ell^2}{\ell + \sum_{i=2}^{\ell+1} \log i + (k-1) \cdot \ell(\log(\ell+1) + 1)},$$

which becomes arbitrarily close to $\frac{2\ell}{\log(\ell+1)+1}$ as k grows.

We can also analyze the performance of randomized list update algorithms in terms of the working set bound by considering their expected cost. Algorithm *Bit*, introduced by Reingold et al. [28], is a simple randomized algorithm that achieves a competitive ratio 1.75, thus beating any deterministic algorithm [11]. Bit allocates a bit $b(a_i)$ for each item a_i and initializes these bits uniformly and independently at random. Upon an access to a_i, it first complements $b(a_i)$, then if $b(a_i) = 0$ it moves a_i to the front, otherwise it does nothing. The following lemma shows that although randomization (for Bit) can improve the competitive ratio it cannot lead to the working set property. In fact the performance of Bit in terms of the working set bound is worse than MTF. This is inconsistent with competitive analysis but consistent with experimental results [7].

Lemma 6. *In the worst case* $E(Bit(X)) \geq \frac{3\ell+1}{2(\log(\ell+1)+1)} \cdot WS(X)$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list and k be an arbitrary integer. Consider the access sequence $X = \{a_\ell^2 a_{\ell-1}^2 \ldots a_1^2\}^k$. Let x_i and x_{i+1} be two consecutive accesses to a_j. After two consecutive accesses to each item, a_j will have been moved to the front of the list with probability 1. Therefore a_j is in the last position of the list maintained by Bit at the time of request x_i and Bit incurs cost ℓ on this request. After this request, Bit moves a_j to the front of the list if and only if $b(a_j)$ is initialized to 1. Since $b(a_j)$ is initialized uniformly and independently at random, this will happen with probability 1/2. Therefore the expected cost of Bit on x_{i+1} is $\frac{1}{2}(\ell+1)$ and the expected cost of Bit on X is $k \cdot \ell(\ell + \frac{\ell+1}{2})$. We have $WS(X) = \ell + \sum_{i=2}^{\ell+1} \log i + (k-1) \cdot \ell(\log(\ell+1) + 1)$. Therefore

$$\frac{E(Bit(X))}{WS(X)} = \frac{k \cdot \ell(\ell + \frac{\ell+1}{2})}{\ell + \sum_{i=2}^{\ell+1} \log i + (k-1) \cdot \ell(\log(\ell+1) + 1)},$$

which becomes arbitrary close to $\frac{3\ell+1}{2(\log(\ell+1)+1)}$ as k grows.

Table 1. Working set bounds of files in Calgary Corpus (normalized by their sizes) before and after Burrows-Wheeler Transform

File	Category	Size (bytes)	l	WS/n	WS/n (BWT)
bib	Bibliography	111261	81	3.9	1.6
book1	Fiction book	768771	82	3.4	1.7
book2	Non-fiction book	610856	96	3.5	1.6
geo	Geophysical data	102400	256	4.2	2.3
news	USENET batch file	377109	98	3.6	1.8
obj1	Object code for VAX	21504	256	3.8	2.1
obj2	Object code for Mac	246814	256	4.1	1.5
paper1	Technical paper	53161	95	3.56	1.73
paper2	Technical paper	82199	91	3.47	1.72
pic	fax picture	513216	159	1.37	1.25
progc	Source code in "C"	39611	92	3.65	1.74
progl	Source code in LISP	71646	87	3.22	1.45
progp	Source code in PASCAL	49379	89	3.38	1.44
trans	Transcript of terminal session	93695	99	3.61	1.38

Table 2. Working set bounds of files in Canterbury Corpus (normalized by their sizes) before and after Burrows-Wheeler Transform

File	Category	Size(bytes)	l	WS/n	WS/n (BWT)
alice29.txt	English text	152089	74	3.9	2.0
asyoulik.txt	Shakespeare play	125179	68	3.6	1.8
cp.html	HTML source	24603	86	3.8	1.8
fields.c	C source	11150	90	3.5	1.6
grammar.lsp	LISP source	3721	76	3.3	1.7
kennedy.xls	Excel Spreadsheet	1029744	256	2.6	1.5
lcet10.txt	Technical writing	426754	84	3.4	1.6
plrabn12.txt	Poetry	481861	81	3.5	1.8
ptt5	CCITT test set	513216	159	1.4	1.1
sum	SPARC Executable	38240	255	3.1	1.7
xargs.1	GNU manual page	4227	74	3.6	1.9

4 Experimental Results

In this section we compute the working set bound for some real life inputs for list update and study the performance of well known list update algorithms in terms of the working set bound. We computed the working set of files of Calgary and Canterbury corpora before and after BWT. Tables 1 and 2 show the results

Table 3. Performance of list update algorithm (normalized by the working set bound) on files of Canterbury and Calgary Corpora after Burrows-Wheeler Transform

File	MTF	TS	FC	TR	$\frac{l}{\log l+1}$
alice29.txt	1.98	2.04	5.80	2.67	11.88
asyoulik.txt	2.10	2.14	6.24	2.79	11.13
cp.html	2.96	3.31	8.20	6.09	13.23
fields.c	2.66	3.43	8.96	9.23	13.83
grammar.lsp	3.04	3.91	6.42	10.57	12.13
kennedy.xls	3.90	3.79	5.55	5.15	22.15
lcet10.txt	1.93	1.97	6.41	2.48	13.10
plrabn12.txt	1.99	1.96	5.26	2.21	12.74
ptt5	1.23	1.20	1.31	1.22	12.25
sum	3.51	4.02	10.00	8.37	18.26
xargs.1	3.04	3.64	6.38	9.01	11.88

File	MTF	TS	FC	TR	$\frac{l}{\log l+1}$
bib	2.18	2.38	9.22	3.78	12.74
book1	1.98	1.92	5.28	2.16	12.86
book2	2.03	2.12	6.90	2.83	14.55
geo	5.66	5.35	6.07	5.74	18.26
news	2.68	2.95	8.34	4.02	14.78
obj1	4.86	5.04	7.45	8.74	18.26
obj2	3.07	3.40	9.60	5.88	18.26
paper1	2.44	2.82	7.32	5.14	14.43
paper2	2.19	2.34	5.64	3.58	13.95
pic	1.79	1.71	2.18	2.04	21.38
progc	2.77	3.22	8.90	6.35	14.07
progl	2.03	2.38	7.66	4.04	13.47
progp	2.16	2.72	9.03	5.38	13.71
trans	2.13	2.75	13.12	5.76	14.90

for the Calgary and Canterbury corpora, respectively. From these results we conclude that the working set bound for BWT of each file is much less than the working set bound of the original file. This reflects the intuition that BWT increases the amount of locality of reference in a sequence.

We also computed the performance of list update algorithms on the BWT of these files. Table 3 shows the corresponding costs normalized by the working set bound of each file. Comparing the experimental results with the theoretical bounds we proved in Section 3 shows that the actual performance of the algorithms is much better than the theoretical worst case bounds. In particular, the worst case lower bound of $\frac{l}{\log l+1}$ seems pessimistic. Furthermore, MTF and TS have close performance and outperform FC and TR. This is consistent with our theoretical results.

5 Conclusions

We introduced a probabilistic model for list update with locality of reference. This model is based on the diffuse adversary model and considers a dynamic probability distribution for accessing the items. We proved that MTF outperforms other algorithms in this model and its performance improves as the locality increases. Analyzing other list update algorithms under this model remains open. Furthermore, we analyzed several online list update algorithms in terms of the working set bound. We proved that MTF achieves the optimal performance in terms of the working set bound, while several other algorithms do not. Thus, both these models confirms the well known belief that MTF is the best list update algorithm on sequences with high locality of reference. Our experiments showed that the working set bound of files decreases after applying BWT. This is consistent with our intuition that BWT increases locality of reference.

References

1. The canterbury corpus, http://corpus.canterbury.ac.nz/index.html
2. Albers, S.: Improved randomized on-line algorithms for the list update problem. SICOMP 27(3), 682–693 (1998)
3. Albers, S., Lauer, S.: On List Update with Locality of Reference. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 96–107. Springer, Heidelberg (2008)
4. Albers, S., Mitzenmacher, M.: Average case analyses of list update algorithms, with applications to data compression. Algorithmica 21(3), 312–329 (1998)
5. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. IPL 56, 135–139 (1995)
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: SODA, pp. 53–62 (1997)
8. Becchetti, L.: Modeling Locality: A Probabilistic Analysis of LRU and FWF. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 98–109. Springer, Heidelberg (2004)
9. Bentley, J., McGeoch, C.: Amortized analyses of self-organizing sequential search heuristics. CACM 28, 404–411 (1985)

10. Bentley, J.L., Sleator, D.D., Tarjan, R.E., Wei, V.K.: A locally adaptive data compression scheme. CACM 29, 320–330 (1986)
11. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
12. Bose, P., Douïeb, K., Langerman, S.: Dynamic optimality for skip lists and B-trees. In: SODA, pp. 1106–1114 (2008)
13. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. Technical Report 124, DEC SRC (1994)
14. Burville, P., Kingman, J.: On a model for storage and search. Journal of Applied Probability 10, 697–701 (1973)
15. Chung, F.R., Hajela, D.J., Seymour, P.D.: Self-organizing sequential search and hilbert's inequalities. In: STOC, pp. 217–223 (1985)
16. 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)
17. Dorrigiv, R., López-Ortiz, A., Munro, J.I.: An Application of Self-Organizing Data Structures to Compression. In: Vahrenhold, J. (ed.) SEA 2009. LNCS, vol. 5526, pp. 137–148. Springer, Heidelberg (2009)
18. Ehmsen, M.R., Kohrt, J.S., Larsen, K.S.: List Factoring and Relative Worst Order Analysis. In: Jansen, K., Solis-Oba, R. (eds.) WAOA 2010. LNCS, vol. 6534, pp. 118–129. Springer, Heidelberg (2011)
19. Gonnet, G.H., Munro, J.I., Suwanda, H.: Toward self-organizing linear search. In: FOCS, pp. 169–174 (1979)
20. Hester, J.H., Hirschberg, D.S.: Self-organizing linear search. ACM Computing Surveys 17(3), 295 (1985)
21. Iacono, J.: Improved Upper Bounds for Pairing Heaps. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 32–45. Springer, Heidelberg (2000)
22. Irani, S.: Two results on the list update problem. IPL 38, 301–306 (1991)
23. Koutsoupias, E., Papadimitriou, C.: Beyond competitive analysis. SICOMP 30(1), 300–317 (2000)
24. Martínez, C., Roura, S.: On the competitiveness of the move-to-front rule. Theoretical Computer Science 242(1–2), 313–325 (2000)
25. McCabe, J.: On serial files with relocatable records. Op. Res. 12, 609–618 (1965)
26. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
27. Munro, J.I.J.: On the Competitiveness of Linear Search. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 338–345. Springer, Heidelberg (2000)
28. Reingold, N., Westbrook, J., Sleator, D.: Randomized competitive algorithms for the list update problem. Algorithmica 11, 15–32 (1994)
29. Rivest, R.: On self-organizing sequential search heuristics. CACM 19, 63–67 (1976)
30. Seward, J.: bzip2, a program and library for data compression, `http://www.bzip.org/`
31. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. CACM 28, 202–208 (1985)
32. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. Journal of the ACM 32(3), 652–686 (1985)
33. Teia, B.: A lower bound for randomized list update algorithms. IPL 47, 5–9 (1993)
34. Witten, I.H., Bell, T.: The Calgary/Canterbury text compression corpus. Anonymous ftp from, `ftp.cpsc.ucalgary.ca/pub/text.compression/corpus/text.compression.corpus.tar.Z`

OnlineMin: A Fast Strongly Competitive Randomized Paging Algorithm

Gerth Stølting Brodal[1,⋆], Gabriel Moruz[2,⋆⋆], and Andrei Negoescu[2]

[1] MADALGO, Department of Computer Science, Aarhus University, Åbogade 34, 8200 Aarhus N, Denmark
gerth@cs.au.dk

[2] Goethe University Frankfurt am Main. Robert-Mayer-Str. 11-15, 60325 Frankfurt am Main, Germany
{gabi,negoescu}@cs.uni-frankfurt.de

Abstract. In the field of online algorithms paging is one of the most studied problems. For randomized paging algorithms a tight bound of H_k on the competitive ratio has been known for decades, yet existing algorithms matching this bound have high running times. We present the first randomized paging approach that both has optimal competitiveness and selects victim pages in subquadratic time. In fact, if k pages fit in internal memory the best previous solution required $O(k^2)$ time per request and $O(k)$ space, whereas our approach takes also $O(k)$ space, but only $O(\log k)$ time in the worst case per page request.

1 Introduction

Online algorithms are algorithms for which the input is not provided beforehand, but is instead revealed item by item. The input is to be processed sequentially, without assuming any knowledge of future requests. The performance of an online algorithm is usually measured by comparing its cost against the cost of an optimal offline algorithm, i.e. an algorithm that is provided all the input beforehand and processes it optimally. This measure, denoted *competitive ratio* [9,12], states that an online algorithm A has competitive ratio c if its cost satisfies $cost(A) \leq c \cdot cost(OPT) + b$, where $cost(OPT)$ is the cost of an optimal offline algorithm and b is a constant. If A is a randomized algorithm, $cost(A)$ denotes the expected cost. In particular, an online algorithm is denoted *strongly competitive* if its competitive ratio is optimal. While the competitive ratio is a quality guarantee for the cost of the solution computed by an online algorithm, factors such as space complexity, running time, or simplicity are also important.

In this paper we study *paging algorithms*, a prominent and well studied example of online algorithms. We are provided with a two-level memory hierarchy, consisting of a cache and a disk, where the cache can hold up to k pages and

⋆ Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

⋆⋆ Partially supported by the DFG grant ME 3250/1-2, and by MADALGO.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 164–175, 2012.

the disk size is infinite. When a page is requested, if it is in the cache a *cache hit* occurs and the algorithm proceeds to the next page. Otherwise, a *cache miss* occurs and the algorithm has to load the page from the disk; if the cache was full, a page must be evicted to accommodate the new one. The cost is given by the number of cache misses performed.

Related work. Paging has been extensively studied over the last decades. In [4] an optimal offline algorithm, denoted MIN, was given. In [12] a lower bound of k on the competitive ratio for deterministic paging algorithms was shown. Several algorithms, such as LRU and FIFO, meet this bound and are thus strongly competitive. For randomized algorithms, Fiat et al. [7] proved a lower bound of H_k on the competitive ratio, where $H_k = \sum_{i=1}^{k} 1/i$ is the k-th harmonic number. They also gave an algorithm, named Mark, which is $(2H_k - 1)$-competitive. The first strongly competitive randomized algorithm being H_k-competitive was Partition [11]. For Partition, the memory requirement and runtime per request can reach $\Theta(n)$, where n is the number of page requests, and n can be far greater than k. Partition was characterized in [1] as counter-intuitive and difficult to understand. The natural question arises if there exist simpler and more efficient strongly competitive randomized algorithms. The Mark algorithm can be easily implemented using $O(k)$ memory and very fast running time ($O(1)$ dictionary operations) per request, but it is not strongly competitive. Furthermore, in [6] it was shown that no Mark-like algorithm can be better than $(2H_k - 1)$-competitive. The strongly competitive randomized algorithm Equitable [1] was a first breakthrough towards efficiency, improving the memory complexity to $O(k^2 \log k)$ and the running time to $O(k^2)$ per page request. In [3] a modification of Equitable, denoted Equitable2, improved the space complexity to $O(k)$. Both Equitable algorithms are based on a characterization in [10] in the context of work functions. The main idea is to define a probability distribution on the set of all possible configurations of the cache and ensure that the cache configuration obeys this distribution. For each request, it requires k probability computations, each taking $O(k)$ time. For a detailed view on paging algorithms, we refer the interested reader to the comprehensive surveys [2,5,8].

Our contributions. In this paper we propose a strongly competitive randomized paging algorithm, denoted OnlineMin, that handles each page request in $O(\log k)$ time in the worst case. This is a significant improvement over the fastest known algorithm, Equitable[1], which needs $O(k^2)$ time per request. The space requirements of our algorithm are $O(k)$, like Equitable2.

The main building block of our algorithm is a priority based incremental selection process starting from the same characterization of an optimal solution in [10] as the Equitable algorithms. The analysis of this process yields a simple cache update rule which is different from the one in [1,3], but leads to the same probability distribution of the cache content. A straightforward implementation

[1] Since no explicit implementation of Equitable2 is provided, due to their similarity we assume it to be the same as for Equitable.

of our update rule requires $O(k)$ time per request. Additionally we design appropriate data structures that result in an implementation which processes a page request in $O(\log k)$ time in the worst case.

2 Randomized Selection Process

In this section we first give some preliminary notions about *offset functions* for paging algorithms introduced in [10]. We then describe in Section 2.2 a new priority based selection process which is the basis of our algorithm OnlineMin. We analyze the selection process in order to obtain a simple page replacement rule which remains at all times consistent with the outcome of the selection process. Finally, in Section 2.3 we prove equivalences between the cache distribution of our selection process and the Equitable algorithms [1,3], which implies that OnlineMin is H_k-competitive.

2.1 Preliminaries

Let σ be the request sequence so far. For the construction of a competitive paging algorithm it is of interest to know the possible cache configurations if σ has been processed with minimal cost. We call these configurations *valid*.

For fixed σ and an arbitrary cache configuration C (a set of k pages), the *offset function* ω assigns C the difference between the minimal cost of processing σ ending in configuration C and the minimal cost of processing σ. Thus C is a valid configuration iff $\omega(C) = 0$. Koutsoupias and Papadimitriou [10] showed that ω can be represented by a sequence of $k+1$ disjoint page sets, denoted layers, and proved the following[2].

Lemma 1. *If $(L_0, \ldots, L_k)$ is a layer representation of ω, then a set C of k pages is a valid configuration, i.e. $\omega(C) = 0$, iff $|C \cap (\cup_{i \le j} L_i)| \le j$ for all $0 \le j \le k$.*

The layer representation is defined as follows. Initially each layer L_i, where $i > 0$, consists of one of the first requested k pairwise distinct pages. The layer L_0 contains all pages not in $L_1, \ldots, L_k$. We denote by ω^p the offset function which results from ω by requesting p. We have the following update rule.

$$\omega^p = \begin{cases} (L_0 \setminus \{p\}, L_1, \ldots, L_{k-2}, L_{k-1} \cup L_k, \{p\}), & \text{if } p \in L_0 \\ (L_0, \ldots, L_{i-2}, L_{i-1} \cup L_i \setminus \{p\}, L_{i+1}, \ldots, L_k, \{p\}), & \text{if } p \in L_i, i > 0 \end{cases}$$

We give an example of an offset function for $k = 3$ in Figure 1. The support of ω is defined as $S(\omega) = L_1 \cup \cdots \cup L_k$. In the remainder of the paper, we call a set with a single element *singleton*. Also, let i be the smallest index such that $L_i, \ldots, L_k$ are singletons. We distinguish the set of *revealed* pages $R(\omega) = L_i \cup \cdots \cup L_k$, and the set of *non-revealed* pages $N(\omega) = L_1 \cup \cdots \cup L_{i-1}$. A valid configuration contains all revealed pages and no page from L_0. Note that when requesting

[2] We use a slightly modified, yet equivalent, version of the layer representation in [10].

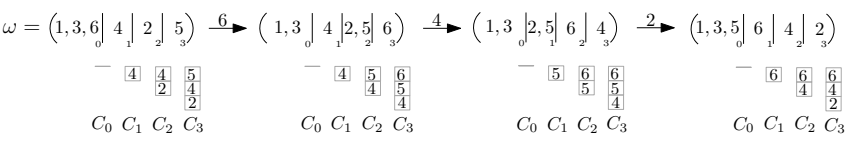


Fig. 1. The update of ω and the selection sets. The initial cache configuration is $\{4, 2, 5\}$ for $k = 3$ and request the pages $6, 4, 2$. The priority of a page is its number.

some non-revealed page p in the support, we have $R(\omega^p) = R(\omega) \cup \{p\}$ and the number of layers containing non-revealed items decreases by one. Moreover, if $p \notin L_1$ then $N(\omega^p) = N(\omega) \setminus \{p\}$ and otherwise $N(\omega^p) = N(\omega) \setminus L_1$. Also, the layer representation is not unique and especially each permutation of the layers containing revealed items describe the same offset function.

Equitable *and* Equitable2. Based on the layer partition above both Equitable algorithms are described using a probability distribution over all configurations where the probability that C is the cache content is defined as the probability of being obtained at the end of the following random process. Starting with $C = R(\omega)$ a page p is selected uniformly at random from $N(\omega)$, p is added to C, and ω is set to ω^p. This process is iterated until C has k pages. The probability for each configuration reachable by one page replacement is computed from its actual configuration such that the distribution remains consistent with the random process. The request is handled according to the computed probabilities.

2.2 Selection Process for OnlineMin

If ω is the offset function for the input requested so far an online algorithm should have a configuration similar to the cache C_{OPT} of an optimal strategy. We know that C_{OPT} contains all revealed items and no item from L_0. Which non-revealed items are in the cache depends on future requests. To guess the order of future requests of non-revealed items OnlineMin assigns priorities to pages when they are requested. It maintains the cache content of an optimal offline algorithm under the assumption that the priorities reflect the order of future requests. We introduce a priority based selection process for the layer representation of ω. Assuming that each order of priorities has equal probability, we prove that the outcome of the selection process has the same probability distribution as the Equitable algorithms. Our approach allows an efficient and easy-to-implement update method for the cache of OnlineMin, which is consistent with our selection process.

In the following we assume that pages from $L_1, \ldots, L_k$ have pairwise distinct priorities. For some set S we denote by $\min_j(S)$ and $\max_j(S)$ the subset of S of size j having the smallest and largest priorities respectively. Furthermore, $\min(S) = \min_1(S)$ and $\max(S) = \max_1(S)$.

Definition 1. *We construct iteratively* $k + 1$ selection sets $C_0(\omega), \ldots, C_k(\omega)$ *from the layer partition* $\omega = (L_0, \ldots, L_k)$ *as follows. We first set* $C_0(\omega) = \emptyset$ *and then for* $j = 1, \ldots, k$ *we set* $C_j(\omega) = \max_j(C_{j-1}(\omega) \cup L_j)$.

When ω is clear from the context, we let $C_i = C_i(\omega)$. For a page request p and offset function $\omega = (L_0, \ldots, L_k)$, denote $\omega^p = (L'_0, \ldots, L'_k)$ and let C'_k be the result of the selection process on ω^p. By the layer update rule each layer contains at least one element and the following result follows immediately.

Fact 1. *$|C_j| = j$ for all $j \in \{0, \ldots, k\}$. If $|L_j|$ is singleton then $C_j = C_{j-1} \cup L_j$. Moreover, all revealed pages are in C_k.*

Updating C_k. We analyze how C_k changes upon a request. First we give an auxiliary result in Lemma 2 and then show in Theorem 1 that C'_k can be obtained from C_k by at most one page replacement. We get how C'_k can be directly constructed from C_k and the layers, without executing the whole selection process.

Lemma 2. *Let p be the requested page from layer L_i, where $0 < i < k$. If for some j, with $i \leq j < k$ we have $q \in C_j$ and $C'_{j-1} = C_j \setminus \{q\}$, then we get:*

$$C'_j = \begin{cases} C_{j+1} \setminus \{q\}, & \text{if } q \in C_{j+1} \\ C_{j+1} \setminus \min\{C_{j+1}\}, & \text{otherwise} \end{cases}$$

Proof. We have:

$$C'_j = \max_j (L'_j \cup C'_{j-1}) = \max_j (L_{j+1} \cup C_j \setminus \{q\}) = C_{j+1} \setminus \{q\} \text{ (case: } q \in C_{j+1})$$

$$C'_j = \max_j (L'_j \cup C'_{j-1}) = \max_j (L_{j+1} \cup C_j \setminus \{q\}) = \max_j (C_{j+1}) \text{ (case: } q \notin C_{j+1})$$

In both cases, we first use the assumption $C'_{j-1} = C_j \setminus \{q\}$ and the partition update rule, $L'_j = L_{j+1}$. In the case $q \in C_{j+1}$ we use $C_{j+1} = \max_{j+1}(L_{j+1} \cup C_j) = \max_j (L_{j+1} \cup C_j \setminus \{q\}) \cup \{q\}$, which holds as $q \in C_j$ implies $q \notin L_{j+1}$. If $q \notin C_{j+1}$, we use $C_{j+1} = \max_{j+1}(L_{j+1} \cup C_j) = \max_{j+1}(L_{j+1} \cup C_j \setminus \{q\})$. We have $q \in C_j$, $q \notin C_{j+1}$ and $|C_{j+1}| = j + 1$, which leads to $C'_j = \max_j (C_{j+1}) = C_{j+1} \setminus \min\{C_{j+1}\}$. □

Theorem 1. *Let p be the requested page. Given C_k, we obtain C'_k as follows:*

1. *$p \in C_k$: $C'_k = C_k$*
2. *$p \notin C_k$ and $p \in L_0$: $C'_k = C_k \setminus \min(C_k) \cup \{p\}$*
3. *$p \notin C_k$ and $p \in L_i$, $i > 0$: $C'_k = C_k \setminus \min(C_j) \cup \{p\}$, and $j \geq i$ is the smallest index with $|C_j \cap C_k| = j$.*

Before the proof, note that for the third case $|C_j \cap C_k| = j$ is equivalent to $|(L_1 \cup \cdots \cup L_j) \cap C_k| = j$ since C_j has elements only in $L_1 \cup \cdots \cup L_j$ and $C_j \subseteq C_k$.

Proof. First assume that $p \in L_0$. In this case, by construction p is not in C_k. The only layers that change are L_{k-1} and L_k: $L'_{k-1} = L_{k-1} \cup L_k$ and $L'_k = \{p\}$. Applying the definition of C'_k and the fact that $C_k = \max_{k-1}(C_{k-2} \cup L_{k-1}) \cup L_k$, since L_k is singleton, we get:

$$C'_k = C'_{k-1} \cup \{p\} = \max_{k-1}(C_{k-2} \cup L_{k-1} \cup L_k) \cup \{p\} = C_k \setminus \min(C_k) \cup \{p\};$$

Now we consider the case when $p \in L_i$. We distinguish two cases: $p \in C_k$ and $p \notin C_k$. If $p \in C_k$, we have by construction that p is in all sets $C_i, \ldots, C_k$ and we get $C_i = \max_i (C_{i-1} \cup L_i) = \max_{i-1} (C_{i-1} \cup L_i \setminus \{p\}) \cup \{p\}$. Based on this observation we show that $C'_{i-1} = C_i \setminus \{p\}$. It obviously holds for $i = 1$ since C'_0 is empty. For $i > 1$ we get:

$$C'_{i-1} = \max_{i-1} (C_{i-2} \cup L_{i-1} \cup L_i \setminus \{p\}) = \max_{i-1} (C_{i-1} \cup L_i \setminus \{p\}) = C_i \setminus \{p\} .$$

Using $C'_{i-1} = C_i \setminus \{p\}$ and $p \in C_i$, applying Lemma 2 we get $C'_i = C_{i+1} \setminus \{p\}$. Furthermore, using that p is in all sets $C_{i+1}, \ldots, C_k$, we apply Lemma 2 for all these sets which leads to $C'_{k-1} = C_k \setminus \{p\}$ and we obtain $C'_k = C'_{k-1} \cup \{p\} = C_k$.

Now we assume that $p \notin C_k$. This implies that p is a non-revealed page. First we analyze the structure of C'_{i-1} which will serve as starting point for applying Lemma 2. If $p \in C_i$ we argued before that $C'_{i-1} = C_i \setminus \{p\}$. Otherwise, we show that $C'_{i-1} = C_i \setminus \min(C_i)$. It holds for $i = 1$ since C_0 is always empty and by Fact 1 we have $|C_1| = 1$. For $i > 1$ we get:

$$C'_{i-1} = \max_{i-1} (C_{i-2} \cup L_{i-1} \cup L_i \setminus \{p\}) = \max_{i-1} (C_{i-1} \cup L_i \setminus \{p\}) = C_i \setminus \min(C_i) .$$

Let $j \geq i$ be the smallest index such that $|C_j \cap C_k| = j$. By construction, we have $C_j \subseteq C_k$. Applying Lemma 2 for sets $C'_{i-1}, \ldots, C'_{j-1}$ we get $C'_{j-1} = C_j \setminus \{s\}$, where $s \in C_j$ and either $s = p$, $s = \min C_j$, or s is a page with minimal priority from a set C_l, with $i \leq l < j$. Note that page s is also in C_k by the definition of C_j and thus $s = p$ can be excluded since p is not in C_k. If s is a page with minimal priority from some set C_l then all the other pages in C_l are also in C_j and thus in C_k because all of them have higher priorities than s. This leads to $C_l \subset C_k$ which contradicts the minimality of j. Thus we have $s = \min C_j$. Since the page $s = \min(C_j)$ is in all sets $C_j, \ldots, C_k$ by Lemma 2 we get $C'_{k-1} = C_k \setminus \min(C_j)$ and it follows $C'_k = C_k \setminus \min(C_j) \cup \{p\}$. □

2.3 Probability Distribution of C_k

Theorem 2. *Assume that non-revealed pages are assigned priorities such that the order of the priorities is distributed uniformly at random. For any offset function ω, the distribution of C_k over all possible cache configurations is the same as the distribution of the cache configurations for the* Equitable *algorithms.*

Proof. Let u be the index of the last non-revealed layer, more precisely $|L_u| > 1$ and $|L_i| = 1$ for all $i > u$. The set of non-revealed items is $N(\omega) = L_1 \cup \cdots \cup L_u$ and the singletons $L_{u+1}, \ldots, L_k$ contain the revealed items $R(\omega)$.

The following selection process is used by both Equitable and Equitable2 to obtain the probability distribution of the cache M. Initially M contains all $k-u$ revealed items $R(\omega)$. Then u elements $x_1, \ldots, x_u$ are added to M, where x_i is chosen uniformly at random from the set of non-revealed items of $\omega^{x_1,\ldots,x_{i-1}}$, the offset function obtained from ω after requesting the sequence $x_1, \ldots, x_{i-1}$.

We define an auxiliary selection $C_k^*(\omega)$ which is a priority based version of EQUITABLE's random process and then prove for every fixed priority assignment that $C_k(\omega) = C_k^*(\omega)$ holds.

Assume that pages in $N(\omega)$ have pairwise distinct priorities, with a uniformly distributed priority order. Initialize $C_k^*(\omega)$ to $R(\omega)$ and add elements $x_1^*, \ldots, x_u^*$ to $C_k^*(\omega)$, where x_i^* is the page with maximal priority from the non-revealed items of $\omega^{x_1^*,\ldots,x_{i-1}^*}$. Obviously all pages from $N(\omega)$ have the same probability to posses the maximal priority and thus x_1^* and x_1 have the same distribution. Since x_1^* is a revealed item in $\omega^{x_1^*}$, the priority order of pages in $N(\omega^{x_1^*})$ remains uniformly distributed. This implies inductively that $C_k^*(\omega)$ has the same distribution as EQUITABLE. Note that by the definition of C_k^* we have $C_k^*(\omega) = C_k^*(\omega^{x_1^*})$ because x_1^* becomes a revealed item in $\omega^{x_1^*}$.

Now we prove for each fixed priority assignment that $C_k(\omega) = C_k^*(\omega)$ by induction on u. For $u = 0$ both C_k^* and C_k contain all k revealed items. For $u \geq 1$, let x_1^* be the non-revealed page with the largest priority in ω. For the auxiliary process, we have already shown that $C_k^*(\omega) = C_k^*(\omega^{x_1^*})$. Also, the index u for $\omega^{x_1^*}$ is smaller by one than for ω, which by inductive hypothesis leads to $C_k^*(\omega) = C_k^*(\omega^{x_1^*}) = C_k(\omega^{x_1^*})$. It remains to prove that $C_k(\omega^{x_1^*}) = C_k(\omega)$. By the definition of the selection process for $C_1, \ldots, C_k$ we have $C_k(\omega) = C_u(\omega) \cup R(\omega)$. Page x_1^* has the highest priority from $N(\omega) = L_1 \cup \cdots \cup L_u$ and thus it is a member of $C_u(\omega)$ and hence also in $C_k(\omega)$. Applying the update rule from Theorem 1 we get $C_k(\omega) = C_k(\omega^{x_1^*})$, and this concludes the proof. □

3 Algorithm ONLINEMIN

3.1 Algorithm

ONLINEMIN initially holds in its cache M the first k pairwise distinct pages. Note that the last requests for all pages in L_i are smaller than the last requests for all pages in L_{i+1}.

Page replacement. The algorithm maintains as invariant that $M = C_k$ after each request. To do so, it keeps track of the layer partition $\omega = (L_0, \ldots, L_k)$, where it suffices to store only the support layers $(L_1, \ldots, L_k)$. The cache update is performed according to Theorem 1. More precisely, if the requested page p is in the cache, M remains unchanged. If a cache miss occurs and p is from L_0 the page with minimal priority from M is replaced by p. If p is from L_i with $i > 0$, and $p \notin M$ we first identify the set C_j in Theorem 1 satisfying $|C_j \cap M| = j$. This can be done as follows. Let $p_1, \ldots, p_k$ be the pages in M sorted in increasing order by their layer index. We search the minimal index $j \geq i$, such that the layer index of p_j is j, i.e. $p_j \in L_j$. We evict the page with minimal priority from $p_1, \ldots, p_j$. The layers are updated after the cache update is done.

Forgiveness. If the amount of pages in $(L_1, \ldots, L_k)$ is $3k$ and a page in L_0 is requested we apply the *forgiveness mechanism* in [3]. More precisely, we perform the partition and cache update as if the requested page was from L_1. Doing this

all pages in L_1 are moved to L_0, i.e. they are removed from the support, and the support size never exceeds $3k$.

Priorities. If page p is requested from L_0, we select for p a rank within the support chosen uniformly at random, i.e. a number in $\{0, \ldots, |S_w|\}$, and we assign it a priority such that it reflects its rank.

Time and space complexity. Storing the layer partition together with the page priorities needs $O(k)$ space by applying the forgiveness mechanism. A naive implementation storing the layers in an array processes a page request in $O(k)$ time. In the remainder of the paper we show how to improve this complexity to $O(\log k)$ time per request in the worst case.

Competitive ratio. We showed in Theorem 2 that the probability distribution over the cache configurations for OnlineMin and Equitable2 are the same. This holds also when using the forgiveness step, and thus the two algorithms have the same expected cost. This leads to the result in Lemma 3.

Lemma 3. OnlineMin *is* H_k*-competitive.*

3.2 Algorithm Implementation

We show how to implement OnlineMin efficiently, such that a page request is processed in $O(\log k)$ worst case time while using $O(k)$ space. In the following we represent each page in the support by the timestamp of its last request.

Basic structure. Consider a list $L = (l_1, \ldots, l_t)$, with $t \leq 4k$, where L has two types of elements: k *layer delimiters* and at most $3k$ *page elements*. Furthermore, we distinguish two types of page elements: *cache elements* which are the pages in the cache and *support elements* which are pages in the support but not in the cache. We store in L the layers $L_1, \ldots, L_k$ from left to right, separated by k layer delimiters. For each layer L_i we store its layer delimiter, followed by the pages in L_i. For each list element l_i, be it page element or layer delimiter, we store a timestamp t_i and a *v-value* v_i with $v_i \in \{-1, 0, 1\}$; for page elements we also store the priority. For some element l_i, if it is a layer delimiter for some layer L_j, we set $v_i = 1$ and t_i to the minimum of all page timestamps in L_j. If l_i is a page element, then t_i is set to the timestamp corresponding to the last request of the page; we set $v_i = -1$ for cache elements and $v_i = 0$ for support elements. Note that the layer delimiters always have t_i values matching the first page in their layer. As described before, layer delimiters always precede page elements. An example is given in Figure 2.

Note that the v-values have the property that $|C_k \cap (L_1 \cup \cdots \cup L_i)| = i$ iff the prefix sum of the v-values for the last element in L_i is zero. Furthermore, since $|C_k \cap (L_1 \cup \cdots \cup L_i)| \leq i$ the prefix sum cannot be negative. This property will be used when dealing with a cache miss caused by a page from L_i, with $i > 0$.

v	1	0	−1	1	0	1	0	−1	−1	1	0	−1	1	−1	1	−1
t	2	2	4	5	5	8	8	10	11	13	13	15	18	18	21	21

Fig. 2. Example for list L: representing pages by timestamps of last requests, we have $L_1 = \{2, 4\}$, $L_2 = \{5\}$, $L_3 = \{8, 10, 11\}$, $L_4 = \{13, 15\}$, $L_5 = \{18\}$, and $L_6 = \{21\}$. Layer delimiters are emphasized and the memory is $M = \{4, 10, 11, 15, 18, 21\}$.

We show how to implement OnlineMin using the following operations on L:

- *find-layer(l_p).* For some page l_p, find its layer delimiter.
- *search-page(l_p).* Check whether l_p is a page in L.
- *insert(l_p), delete(l_p).* The item l_p is inserted (or deleted) in L.
- *find-min(l_p).* Find the cache element $l_q \in (l_1, \ldots, l_p)$ with minimum priority.
- *find-zero(l_p).* Find the smallest j, with $p \leq j$ such that $\sum_{l=1}^{j} v_l = 0$, and return l_j.

We describe how to update the list L upon a request for some page p. OnlineMin keeps in memory at all times the elements in L having the v-value equal to -1.

If $p \notin M$, we must identify a page to be evicted from M. To evict a page we set its v-value to zero and to load a page we set its v-value to -1. We first find the layer delimiter for p. We can have $p \in L_i$ with $0 < i \leq k$ or $p \in L_0$. If $p \in L_i$, the page to be evicted is the cache element in $L_1 \cup \cdots \cup L_j$ having the minimum priority, where $j \geq i$ is the minimal index satisfying $|M \cap (L_1 \cup \cdots \cup L_j)| = j$. This is done using `find-zero`(l_{L_i}), where l_{L_i} is the layer delimiter of L_i, and the page to be evicted is identified using `find-min` applied to the value returned by `find-zero`. If $p \in L_0$, if the forgiveness need not be applied, the page having the smallest priority in M is to be evicted. We identify this page in L using `find-min` on the last element in L. If we must apply forgiveness, we treat p as being a support page in L_1.

After updating the cache, we perform in L the layer updates as follows. If $p \in L_i$ with $i > 0$, the layers are updated as follows: $L_{i-1} = L_{i-1} \cup L_i \setminus \{p\}$, $L_j = L_{j+1}$ for all $j \in \{i, \ldots, k-1\}$, and $L_k = \{p\}$. We first delete the layer delimiter for L_i and the page element for p, which triggers not only the merge of L_{i-1} and $L_i \setminus \{p\}$, but also shifts all the remaining layers, i.e. $L_j = L_{j+1}$ for all $j \geq i$. If we deleted the layer delimiter for L_1, we also delete all pages in L_1 because in this case L_1 is merged with L_0. To create $L_k = \{p\}$, we simply insert at the end a new layer delimiter followed by p, both items having as timestamp the current timestamp.

If $p \in L_0$, we first check whether we must apply the forgiveness step, and if so we apply it by simulating the insertion of p in L_1 and then requesting it, as described above. If forgiveness need not be applied, we update the layers $L_{k-1} = L_{k-1} \cup L_k$ and $L_k = \{p\}$ as follows. We first delete the layer delimiter of L_k, which translates into merging L_{k-1} and L_k. Then, we insert a new layer delimiter having the timestamp of the current request, i.e. create L_k, and insert p with the same timestamp.

3.3 Data Structures

We implement all the operations previously introduced using two data structures: a *set structure* and a *page-set* structure. The set structure focuses only on the `find-layer` operation, and the page-set data structure deals with the remaining operations. While most operations can be implemented using standard data structures, i.e. balanced binary search trees, the key operation for the page-set structure is `find-zero`. That is because we need to find in sublinear time the first item to the right of an arbitrary given element having the prefix sum zero in the presence of updates, and the item that is to be returned can be as far as $\Theta(k)$ positions in L.

Set structure. The set structure is in charge only for the `find-layer` operation. To do so, it must also support updating the layers. It is a classical balanced binary search tree, e.g. an AVL tree, built on top of the layer delimiters in L having as keys the timestamps of the delimiters. Whenever a layer delimiter is inserted or deleted from L, the set structure is updated accordingly. Each operation takes $O(\log k)$ time in the worst case.

Page-set structure. The page-set structure contains all elements of L and supports all the remaining operations required on L. We store the elements of L, i.e. both page elements and layer delimiters, in the leaves of a regular leaf oriented balanced binary search tree indexed by the timestamps. For some node u, denote by $\mathcal{T}(u)$ the subtree rooted at u and by $\mathcal{L}(u)$ the leaves of $\mathcal{T}(u)$. For each node u we store the sum s_u of the v-values in $\mathcal{L}(u)$. We also store the minimum prefix sum value m_u among all the prefix sums restricted on the elements within $\mathcal{L}(u)$. More precisely, if $\mathcal{L}(u) = (p_1, \ldots, p_m)$, we have $m_u = \min_{l=1}^{m}(\sum_{j=1}^{l} p_j)$. Finally, in each node u we also store the minimum priority of a cache page in the subtree rooted at u. Note that if the subtree rooted at u has no cache elements the priority field is set to infinity.

Fact 2. *For each internal node u we have that $m_u = \min(m_{u_l}, s_{u_l} + m_{u_r})$, where u_l and u_r denote the left and right child of u respectively.*

Updates. We discuss how to perform insertions and deletions in the page-set structure. To insert an element, we first identify its location and then insert it. It remains to update the information at the internal nodes, i.e. the sum of the v-values, the minimum prefix-sum values and the minimum priorities. The sums of the elements of the subtrees are easily updated in a bottom up traversal, together with the minimum priorities, even if rotations need to be done. The minimum prefix sum values can also be updated in a bottom up traversal using the observation stated in Fact 2. Deleting an element in the page-set structure is done analogously to insertion. We note however that when requesting a page in L_1 we must delete both the layer delimiter and all page elements in L_1 from the data structure which leads to $O(\log k)$ amortized time. We will show later how to improve this bound to $O(\log k)$ worst case time for deletions as well.

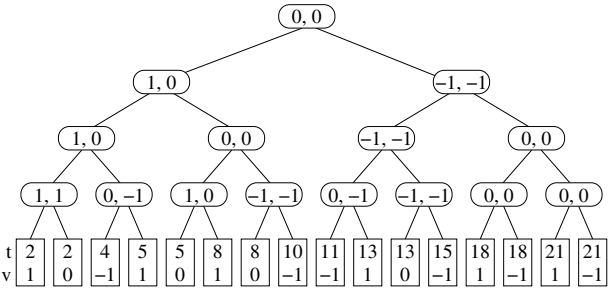


Fig. 3. The page-set data structure for $L_1 = \{2, 4\}$, $L_2 = \{5\}$, $L_3 = \{8, 10, 11\}$, $L_4 = \{13, 15\}$, $L_5 = \{18\}$, and $L_6 = \{21\}$, and the memory $M = \{4, 10, 11, 15, 18, 21\}$. For each internal node u we show the (s_u, m_u) values.

Queries. We turn to queries supported by the page-set structure, which are the queries required on L. The search-page operation is implemented using a standard search in a leaf-oriented binary search tree.

To find the page element having the minimum priority in $l_1, \ldots, l_p$, we first find the value of the priority as follows. On the path from l_p to the root, for each node u we consider the minimum priority value stored at its left child if the left child is not on the path. The priority to be returned is the smallest among these minimums. To find the page, we traverse the tree top-down and at each node we branch on the subtree matching the minimum priority value. Since it does a bottom-up and a top-down traversal, this operation takes $O(\log k)$ time.

It remains to deal with the `find-zero` operation, where we are given some leaf storing l_p and must return the first leaf to the right which has the prefix sum of the v-values zero. We note that the prefix sum cannot be negative, and thus it suffices to find the first leaf to the right having the minimum prefix sum. We do so in two steps: we first identify a subtree containing the leaf having the minimum prefix sum in bottom-up traversal and then we identify the leaf itself in a top-down traversal of this subtree. To identify the subtree containing the leaf to be returned, we traverse the path from the leaf storing l_p to the root while maintaining a sum s of the v-values of the right children not on this path, and at each node u we compute a score as follows. If the right child u_r of u is not on the path, the score of u is given by $s + m_{u_r}$ and afterwards we set $s = s + s_{u_r}$. The subtree we are looking for is the one having the minimum score; in case of several subtrees having an identical score, the leftmost one, i.e. the first one encountered on the path from the leaf to the root, is considered. To identify the leaf having the minimum prefix sum, we do a top-down traversal of the subtree previously computed and we use the observation stated in Fact 2 to decide which way to branch, i.e. we branch left if $m_{u_l} \leq s_{u_l} + m_{u_r}$ and we branch right otherwise. This operation requires a bottom-up and a top-down traversal of the tree and thus takes $O(\log k)$ time in the worst case.

Worst-case bounds. The only operation taking $\omega(\log k)$ time is page deletion, more precisely when a page in L_1 is requested all pages in L_1 are moved to L_0 and thus should be removed from the support. Instead of deleting the set delimiter and all the pages corresponding to L_1, we delete only the set delimiter. With the leading set delimiter removed, the list L no longer starts with a set delimiter, but with at most $O(k)$ elements having the v-value set to 0, since all of these pages belong to L_0 and thus cannot be cache elements. Also, these pages do not influence the prefix sums for the v-values. When we process a page, we simply start by checking if the leftmost element in the tree has a v-value of 0, and if so we delete it. Since each page requests adds at most one new element to the support, the space complexity is still $O(k)$. This way deletions can be done in $O(\log k)$ time in the worst case.

Each page request uses $O(1)$ operations in both data structures. In Theorem 3 we give the time and space complexities for OnlineMin.

Theorem 3. OnlineMin *uses* $O(k)$ *space and processes a request in* $O(\log k)$ *time in the worst case.*

Acknowledgements. We would like to thank previous anonymous reviewers for very insightful comments and suggestions. Also, we would like to thank Annamária Kovács for useful advice on improving the presentation of the paper.

References

1. Achlioptas, D., Chrobak, M., Noga, J.: Competitive analysis of randomized paging algorithms. Theoretical Computer Science 234(1-2), 203–218 (2000)
2. Albers, S.: Online algorithms: a survey. Mathematical Programming 97(1–2), 3–26 (2003)
3. Bein, W.W., Larmore, L.L., Noga, J., Reischuk, R.: Knowledge state algorithms. Algorithmica 60(3), 653–678 (2011)
4. Belady, L.A.: A study of replacement algorithms for virtual-storage computer. IBM Systems Journal 5(2), 78–101 (1966)
5. Borodin, A., El-Yaniv, R.: Online computation and competitive anlysis. Cambridge University Press (1998)
6. Chrobak, M., Koutsoupias, E., Noga, J.: More on randomized on-line algorithms for caching. Theoretical Computer Science 290(3), 1997–2008 (2003)
7. Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive paging algorithms. Journal of Algorithms 12(4), 685–699 (1991)
8. Fiat, A., Woeginger, G.J. (eds.): Online Algorithms, The State of the Art (the book grow out of a Dagstuhl Seminar (June 1996, 1998)
9. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica 3, 77–119 (1988)
10. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. In: Proc. 35th Symposium on Foundations of Computer Science, pp. 394–400 (1994)
11. McGeoch, L.A., Sleator, D.D.: A strongly competitive randomized paging algorithm. Algorithmica 6(6), 816–825 (1991)
12. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)

Faster and Simpler Approximation of Stable Matchings

Katarzyna Paluch*

Institute of Computer Science, Wrocław University

Abstract. We give a $\frac{3}{2}$-approximation algorithm for stable matchings that runs in $O(m)$ time. The previously best known algorithm by McDermid has the same approximation ratio but runs in $O(n^{3/2}m)$ time, where n denotes the number of people and m is the total length of the preference lists in a given instance. Also the algorithm and the analysis are much simpler. We also give the extension of the algorithm for computing stable many-to-many matchings.

1 Introduction

In the paper we consider a variant of the problem called **Stable Matchings**, known also in the literature as the **Stable Marriage** problem. The problem is defined as follows. We are given two sets W and U of women and men. Each woman w of W has a preference list L_w of a subset of men and similarly each man m of U has a preference list L_m of a subset of women. The preference lists are linearly ordered lists of **ties**, which are subsets of men (or resp. women), who are equally good for a given woman (resp. man). Ties are disjoint and can contain also one person, appropriately a man or a woman. Thus if m and m' are on list L_w of woman w, then either (1) w prefers m to m' or in other words m is better for w than m' or (2) m and m' are in a tie on L_w and then we say that w is indifferent between m and m' or that m and m' are equally good for her or (3) w prefers m' to m. Man m and woman w are said to be **mutually acceptable** to each other if they belong to each other's preference lists. The most preferred person(s) is(are) at the top the preference lists. A **matching** is a set of pairs (m, w) such that $m \in U, w \in W$ and m and w are mutually acceptable and each man/woman belongs to at most one pair. If (m, w) belongs to a certain matching M, then we write $M(m) = w$, which means that in M woman w is a partner of m and analogously that $M(w) = m$. If man m (or woman w) is not contained in any pair of a matching M, then we say that m (w) is **unmatched** or **free** in M. A matching M is called **stable** if it does not admit a **blocking pair**. A pair (m, w) is blocking for M if (0) m and w are mutually acceptable and (1) m is unmatched or prefers w to $M(m)$ and (2) w is unmatched or prefers m to $M(w)$. Each instance of the problem can be represented by a bipartite graph $G = (U \cup W, E)$ with vertices U representing men, vertices W representing women and edges E connecting all mutually acceptable pairs of men and women. The

* Supported by MNiSW grant number N N206 1723 33, 2007-2010.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 176–187, 2012.

problem we are interested in is that of finding a stable matching that has the largest cardinality. The version in which there are no ties in the preference lists of men and women has been long known and an algorithm by Gale and Shapley [4] solves it exactly in $O(m)$ time, where m denotes the number of edges in the underlying graph. In the version without ties a stable matching always exists and every stable matching has the same cardinality. If we allow ties, as in the problem we consider in this paper, then a stable matching also always exists and can be found via the Gale/Shapley algorithm by breaking ties arbitrarily. However, the sizes of stable matchings can vary considerably and the problem of finding a stable matching of maximum cardinality is NP-hard, which was shown by Manlove et al. in [13]. Therefore it is desirable to devise an approximation algorithm for the problem.

Previous Results. Previous approximation algorithms were presented in [13], [8], [9], [10], [11]. Currently the best approximation algorithm is by McDermid [14] and achieves the approximation guarantee $\frac{3}{2}$. Its running time is $O(n^{3/2}m)$, where n denotes the number of vertices and m the number of edges. Inapproximabilty results were shown in [5], [6], [16].

Our Results. While constructing approximation algorithms the goal is not only to achieve a good approximation guarantee but also good running time (to name just two examples, see [1],[15]). We give a 3/2-approximation algorithm that runs in $O(m)$ time and additionally is significantly simpler than that of McDermid. In devising the algorithm we were led by the observation that it suffices to find a stable matching that will not create a dangerous path, which is defined later. We also give the extension of the algorithm for computing stable many-to-many matchings, which runs in $O(mlogc)$ time, where c denotes the minimum of the maximal capacities in each side of the bipartition. In particular it means we give an $O(m)$-time algorithm for the Hospitals-Residents problem, improving on an $O(d^{5/2}n^{3/2}m)$ time algorithm given by McDermid, where d denotes the maximal capacity of a hospital. McDermid's algorithm follows from the reduction of the Hospitals-Residents problem to the Stable Matchings problem by "cloning" hospitals. The approach by cloning does not work if the vertices on both sides of the bipartition are allowed to have capacities larger than 1. Since the problems have many practical applications (see [2], [3], [7] for example), we believe our algorithms will be of help.

2 Algorithm

For a given instance of the problem let M_{opt} denote an optimal (i.e. largest) stable matching and let M, M' be any two matchings. We say that e is an M-edge if $e \in M$. A path P or a cycle C is called **alternating (wrt M)** if its edges alternate between M-edges and edges of $E \setminus M$. It is well known from matching theory (see [12] for example) that $M \oplus M'$ can be partitioned into a set of maximal alternating paths and alternating cycles. (For two sets X, Y, the set $X \oplus Y$ denotes $(X \setminus Y) \cup (Y \setminus X)$.) Let S denote a set of maximal alternating

paths and cycles of $M \oplus M_{opt}$. Consider any alternating cycle c of S or any alternating path p of even length of S. Then both c and p contain the same number of M-edges and M_{opt}-edges. Consider an alternating path p of length $2k+1$ of S. Then either $\frac{|M_{opt} \cap p|}{|M \cap p|} = \frac{k+1}{k}$ or $\frac{|M \cap p|}{|M_{opt} \cap p|} = \frac{k+1}{k}$. Therefore if M is stable and S does not contain a path of length 3 with the middle edge being an M-edge, then $|M_{opt}| \leq \frac{3}{2}|M|$ and M is a $\frac{3}{2}$-approximation of M_{opt}. To achieve a $\frac{3}{2}$-approximation we will eliminate such potential paths of length 3 of $M \oplus M_{opt}$.

Accordingly we define a **dangerous path wrt to a matching** M to be an alternating path $P = (w, m_1, w_1, m)$ such that w and m are unmatched in M (which means that (m_1, w_1) is in M and $(w, m_1), (w_1, m)$ do not belong to M) and (m_1, w_1) is not a blocking pair for matching $M' = \{(w, m_1), (w_1, m)\}$. Let us notice that if P is a dangerous path, then either m_1 is indifferent between w and w_1 and then we say that P is a **masculine dangerous path** or w_1 is indifferent between m and m_1 and then we say that P is a **feminine dangerous path**. A path P can of course be both a masculine and feminine dangerous path.

We also introduce the following terminology. If man m is matched to woman w and there is at least one free woman w_1 such that w and w_1 are equally good for m, then we say that w_1 is a **satellite** of m and m is **satellitic**. If woman w is matched to a satellitic man m, then we say that w is **insecure**. If $e = (m, w)$ is such that w is free and there is at least one free woman w_1 such that w and w_1 are equally good for m, then e is called **special**. If man m has at least one free woman incident with him, then he is said to be **subsatellitic**. Woman w matched to a subsatellitic man m and not insecure is said to be **uneasy wrt to** m' if m and m' are equally good for her.

2.1 Description of Algorithm GS Modified

Algorithm GS Modified given further on is to some extent modeled on the Gale-Shapley algorithm in which men propose to women on their lists and women dispose. In the course of running the algorithm preference lists L_m will diminish and some additional lists L'_m will be built. If at some point a free man m has a nonempty list L_m, it means that he has not yet proposed to all women on his list L_m and potentially belongs to a blocking pair or a masculine dangerous path. If a free man m has a nonempty list L'_m, it means that he potentially belongs to a feminine dangerous path.

Whenever it is man m's turn to propose and $L_m \neq \emptyset$, he would like to get matched to the best possible woman on his list without creating a blocking pair (as in GS algorithm) but also ensure that he does not belong to any masculine dangerous path. To this end m proposes to the woman w to whom he has not yet proposed and who is as high on L_m as possible. If w is free or matched to someone worse for her than m, she accepts m and rejects her current partner if she had one. If w is insecure, which means that she is matched to some man m' such that there is a free woman w' who is equally good for m' as w, then it means that m currently belongs to a masculine dangerous path (m, w, m', w'). In this case w does not care whether m is better for her than m' and accepts him while rejecting m' and immediately afterwards m' proposes to w', who accepts him.

This operation can be very well viewed as though m' proposed to w' without having proposed to w first and some time later m proposed to w (here edge (m', w) was special at the moment m' proposed to w for the first time and that's why if it is m''s next turn to propose, he will propose to w again, because in this case w was not removed from $L_{m'}$.) To avoid multiple operations of this kind concerning one woman we will assume that given a tie a man proposes to unmatched women before proposing to matched ones. If a woman w, to whom m proposes is matched to man m' equally good for her as m and w is uneasy wrt to m, meaning that m' has some free women on his list, then at the current moment m belongs to a feminine dangerous path. What happens now is that w rejects m but m adds w to his list L'_m. (w does not accept m because m may be subsatellitic.) In every other case w rejects m.

If man m has proposed to all women on his list L_m and remained free but his list L'_m is nonempty, he will propose to women on L'_m starting from the top. If he proposes to w and w is matched to some man m' who is equally good for her as m and additionally m' is subsatellitic, then w accepts m and rejects m'. Otherwise w rejects him. (Notice that if m proposes to $w \in L'_m$ (this means also that $L_m = \emptyset$), then it cannot be the case that m is better for w than $M(w)$.)

Algorithm GS Modified

Each man m's preference list L_m is organized in such a way that if L_m contains a tie t, then free women in t come before matched women in t. At the beginning all women are free and ties on men's lists are broken arbitrarily and in the course of running the algorithm whenever woman w becomes matched for the first time, say to man m, we move her to the end of every tie she belongs to but the one on list L_m.

while there exists a free man m with a nonempty list L_m or a nonempty list L'_m
  if $L_m \neq \emptyset$, **then**
    $w \leftarrow$ woman at the top of m' s list L_m
    if (m, w) is not special, remove w from L_m
    if w is free, **then** $M \leftarrow M \cup (m, w)$
    else if w is insecure, **then**
      let w' be a satellite of $M(w)$
      if $(M(w), w')$ is not special, remove w' from $L_{M(w)}$
      $M \leftarrow M \cup \{(m, w), (M(w), w')\} \setminus (w, M(w))$
    else if w prefers m to $M(w)$, **then** $M \leftarrow M \cup (m, w) \setminus (w, M(w))$
    else if w is uneasy wrt to m, **then** add w to the end of list L'_m
  else
    $w \leftarrow$ woman at the top of m' s list L'_m
    remove w from L'_m
    if w is uneasy wrt to m, **then** $M \leftarrow M \cup (m, w) \setminus (w, M(w))$

First we show how Algorithm GS Modified runs on the following example. Suppose the preference lists of men m_1, m_2, m_3, m_4 and women w_1, w_2, w_3, w_4 are as follows. The brackets indicate ties.

$m_1 : (w_1, w_2)\ w_3$		$w_1 : m_1\ m_2$	m_3
$m_2 : w_1$	$w_3\ w_4$	$w_2 : m_3\ m_1$	m_2
$m_3 : w_2$	$w_1\ w_3$	$w_3 : m_1\ (m_2, m_4)$	m_3
$m_4 : w_3$		$w_4 : m_2$	

Suppose that m_1 starts. m_1 proposes to w_1 and gets accepted ((m_1, w_1) is a special edge and w_2 is a satellite of m_1). Now suppose that it is m_2's turn to propose. (It might also be m_3 or m_4.) m_2 proposes to w_1 and gets accepted because w_1 is insecure. m_1 gets matched with w_2. m_3 proposes to w_2 and gets accepted. m_1 proposes to w_1 (as (m_1, w_1) was a special edge) and gets accepted. m_2 proposes to w_3 and gets accepted. m_4 proposes to w_3 and gets rejected but w_3 is uneasy wrt to m_4 and m_4 adds w_3 to his list L'_{m_4}. Afterwards m_4 proposes to w_3 again, this time from L'_{m_4}, and gets accepted. m_2 proposes to w_4 and gets accepted.

3 Correctness of Algorithm GS Modified

In this section we prove the correctness of Algorithm GS Modified.

If $w \in L_m$ and m proposes to w, then we will sometimes say that m proposes from L_m (to w). If $L_m = \emptyset$, $w \in L'_m$ and m proposes to w, then we will sometimes say that m proposes from L'_m (to w).

Lemma 1. *1) If woman w becomes matched, she will stay matched. 2) Woman w can become insecure only the first time someone, say m, proposes to her and only if at the time of proposal edge (m, w) is special. If an insecure woman w receives a proposal, she always accepts it and is no longer insecure. 3) If woman w is matched to man m and not insecure, she can accept man m' only if m' is at least as good gor her as m. Moreover, if m' is better for her than m, she always accepts him. If m' is equally good for her as m, then she accepts him, only if she is uneasy wrt to m' and m' proposes from $L'_{m'}$. 4) If woman w matched to man m is not insecure and changes m for m', who is equally good for her as m, then m is subsatellitic and m' is not.*

Proof. Statements 1) and 3) follow directly from the description of Algorithm GS Modified. 2) If w is matched and m proposes to her, then there is no free woman w' incident with m who is equally good for m as w (because then m would propose to w' before proposing to w). As a result if w becomes matched to m she will not become insecure and she will cease to be insecure if she was before. 4) If w changes m for m' who is equally good for her as m (and w is not insecure), then by the above statement m' proposes from $L'_{m'}$ and m is subsatellitic. Man m' proposing from $L'_{m'}$ does not have any free women incident with him. □

Lemma 2. *Let M denote a matching computed by Algorithm GS Modified. Then the graph does not contain blocking pairs and dangerous paths.*

Proof. Suppose that (m, w) are a blocking pair. m is either free or $M(m)$ is worse for him than w. It means that at some point m proposed to w from L_m when edge (m, w) was not special. (Clearly at some point m proposed to w from L_m. Assume that at that point edge (m, w) was special. Then w was free and accepted m. However m got rejected later and therefore proposed to w from L_m again, when edge (m, w) was no longer special.) If w rejected him then, then by

Lemma 1 w was not insecure and matched to someone at least as good for her as m and thus would have stayed matched to someone as good for her as m. If w accepted m, then after getting matched to m she was not insecure and by Lemma 1 would have stayed matched to someone at least as good for her as m. Either way we get a contradiction.

Suppose now that the graph contains a masculine dangerous path (m', w, m, w') such that $m = M(w)$. Thus m is satellitic and w is insecure. Since she is insecure, it means that the only proposal she ever got was from m, but m' must have proposed to her too, a contradiction.

Finally suppose that the graph contains a feminine dangerous path (m', w, m, w') such that $m = M(w)$. Thus m is subsatellitic and w is uneasy wrt to m', also m' is not subsatellitic. At some point m' proposed to w from $L_{m'}$ while (m, w) was not special. If he got accepted at that moment, then later on he could not become rejected, because by Lemma 1 after accepting m' woman w was not insecure and could not accept a subsatellitic man equally good for her as her current partner. Therefore he was rejected then and w was already matched with m (by Lemma 1 3) and 4)). Hence w was uneasy wrt to m' (because m was subsatellitic) and m' added w to the end of list $L'_{m'}$. Thus later m' proposed to w from $L'_{m'}$. According to the algorithm w would have accepted him and could not later on become matched to someone equally good for her as m' and subsatellitic. Contradiction. □

Theorem 1. *Algorithm GS Modified computes a stable matching M which is a $\frac{3}{2}$-approximation of the optimal solution. Algorithm GS Modified runs in $O(m)$ time.*

Proof. By Lemma 2 matching M computed by Algorithm GS Modified is stable and does not contain dangerous paths. Therefore M is a $\frac{3}{2}$- approximation of the optimal solution.

The running time of the algorithm is proportional to the total length of lists L_m and L'_m. Each edge of L_m is scanned at most twice - twice, only if the first time it was scanned, it was special and each edge of L'_m is scanned at most once. □

Let us finally make the following remark.

If we break ties and run the classic Gale/Shapley algorithm, then the cardinality of the computed matching depends on the order in which we break ties. Algorithm GS Modified outputs a matching that would have been output by the GS algorithm if ties were broken as follows. Men's lists would be identical to those at the end of Algorithm GS Modified but for one thing: if at some point of running Algorithm GS Modified man m proposes to an insecure woman w and as a result m gets matched to w and w's partner $M(w)$ gets matched to his satellite w', then a tie on $L_{M(w)}$ would be broken in such a way that w' comes before w. Every tie t on a woman w's list would be first broken into $(m_1, m_2, \ldots, m_s)$ in such a way that m_1 denotes the first man of t to whom w got matched without becoming insecure and assuming that it happened at some step S, m_2 denotes the first man of t who proposed to w after step S, m_3 denotes the second man of t, who proposed to w after step S and so on. Next we would make the following alterations on women's lists: if at some point man $m \in F$ proposes to an uneasy

woman w matched to $M(w)$, then a tie on L_w would be broken in such a way that m comes before $M(w)$.

4 Extension to Stable b-Matchings

Suppose we have a bipartite graph $G = (V, E)$, where $V = U \cup W$ and U, W are disjoint sets, and a function $b : V \to N$. Then a subset $M \subseteq E$ is called a b-matching if for each $v \in V$ it is $deg_M(v) \leq b(v)$, where $deg_M(v)$ denotes the degree of vertex v in a graph $G_M = (U \cup W, M)$. We will call vertices of U - U-agents and vertices of W - W-agents and vertices of $U \cup W$ - agents. Each U-agent u of U has a preference list L_u of a subset of W-agents and analogously each W-agent w has a preference list L_w of a subset of U-agents. The preference lists are linearly ordered lists of ties. The majority of the terminology for stable matchings goes through for stable b-matchings. Instead of saying that some agent or vertex is free we will use the term **unsaturated**: agent v is unsaturated in a b-matching M if $deg_M(v) < b(v)$ and if $deg_M(v) = b(v)$, then we will say that v is **saturated**. For any agent v by $M(v)$ we will denote the set $\{w \in U \cup W : (v, w) \in M\}$. A pair (u, w) is **blocking** for a b-matching M if (0) u and w are mutually acceptable and (1) u is unsaturated or prefers w to one of W-agents of $M(u)$ and (2) w is unsaturated or prefers u to one of U-agents of $M(w)$. A b-matching M is said to be stable if it does not admit a blocking pair. As previously we are interested in finding a stable b-matching of largest size. Let us also note that if for each u in U we have $b(u) = 1$, then the problem is known under the name of the Hospitals-Residents problem or one-to-many stable matching problem.

Alternating paths and cycles are defined for b-matchings in an analogous way as for matchings but we do not require paths and cycles to be simple, i.e. an **alternating path** P **wrt** a b-matching M is defined as any sequence of edges $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\}$ such that the edges alternate between M-edges and edges of $E \setminus M$ and an **alternating cycle** C **wrt** M is defined as an alternating path (wrt M) that ends and begins with the same vertex, i.e. the sequence of edges has the form $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_1)\}$. As before for any two b-matchings M, M', a symmetric difference $M \oplus M'$ can be partitioned into alternating paths and cycles. A given stable b-matching M might be not a 3/2-approximation of M_{opt}, where M_{opt} denotes a stable b-matching of maximum size, if the graph contains a dangerous path defined as follows. If M is a stable b-matching, then a path $P = (w, u_1, w_1, u)$ is called **dangerous** if (u_1, w_1) is in M, $(w, u_1), (w_1, u)$ are not in M, w and u are unsaturated, u_1, w_1 are saturated and (u_1, w_1) is not a blocking pair for a b-matching $M' = (M \setminus (u_1, w_1)) \cup (w, u_1) \cup (w_1, u)$. Since (w, u_1) is not blocking for M, w is not better for u_1 than any of the W-agents he is currently matched with and analogously u is not better for w_1 than any of the U-agents he is currently matched with. Thus if P is dangerous, then either w, w_1 are equally good for u_1 and then P is called a **masculine dangerous path**, or u, u_1 are equally good for w_1 and then P is called a **feminine dangerous path**.

An approximation algorithm for stable b-matchings is constructed analogously to the algorithm for stable matchings. U-agents play the role of men and W-agents play the role of women. For convenience we shall refer to a U-agent as "he" and to a W-agent as "she". We adapt the terminology from the one-to-one setting to the current one as follows. If a U-agent u is matched with a W-agent w and there is at least one unsaturated W-agent w_1 such that w and w_1 are equally good for u, then we say that w_1 is a **satellite of** u **wrt** w and u is **satellitic wrt** w. W-agent w matched to a U-agent u satellitic wrt w is said to be **insecure**. If $e = (u, w)$ is such that w is unsaturated and there is at least one unsaturated W-agent w_1 such that w and w_1 are equally good for u, then e is called **special**. If U-agent u has at least one unsaturated W-agent incident with him, then he is called **subsatellitic**. A saturated W-agent w matched to a subsatellitic man u and not insecure is said to be **uneasy wrt** u' if u and u' are equally good for her. By the **worst U-agent matched with a W-agent** w we will man any U-agent in $u \in M(w)$ such that there is no other U-agent $u' \in M(w)$ who is worse for w than u.

Algorithm ASBM (short for *Approximate Stable b-Matching*)

Each U-agent u's preference list L_u is organized in such a way that if L_u contains a tie t, then unsaturated W-agents in t come before saturated W-agents in t. At the beginning all W-agents are unsaturated and ties on U-agents's lists are broken arbitrarily and in the course of running the algorithm whenever W-agent w becomes matched for the first time, say to U-agent u, we move her to the end of every tie she belongs to but the one on list L_u.

while there exists an unsaturated U-agent u with a nonempty list L_u or a nonempty list L'_u

if $L_u \neq \emptyset$, **then**
$w \leftarrow W$-agent at the top of u' s list L_u
if (u, w) is not special, **then** remove w from L_u
if w is unsaturated, **then** $M \leftarrow M \cup (u, w)$
else if w is insecure, **then**
let w' be a satellite of a U-agent $u' \in M(w)$ wrt to w
if (u', w') is not special, remove w' from $L_{u'}$
$M \leftarrow M \cup \{(u, w), (u', w')\} \setminus (w, u')$
else if w prefers u to the worst U-agent in $M(w)$, **then**
let u' denote the worst U-agent matched with w who is
subsatellitic, if such one exists;
otherwise let u' denote any worst U-ag. matched with w
$M \leftarrow M \cup (u, w) \setminus (w, u')$
if w is uneasy wrt to u', **then** add w to the end of list $L'_{u'}$
else if w is uneasy wrt u, **then** add w to the end of list L'_u.
else
$w \leftarrow W$-agent at the top of u' s list L'_u
remove w from L'_u
if w is uneasy wrt to u, **then**
let u' denote a subsatellitic U-agent in $M(w)$ equally good for w as u
if w is uneasy wrt to u', add w to the end of $L'_{u'}$
$M \leftarrow M \cup (u, w) \setminus (w, u')$

4.1 Data Structures and Running Time

Each agent a (either a U-agent or W-agent) has a preference list L_a, which is a list of lists i.e. we have a list for each tie. For each list we have the acces to both its first and last element.

Each agent has a pointer to their position in every tie (1-element list is here also considered a tie) they belong to. Whenever W-agent w gets saturated for the first time, w goes over her whole list L_w and moves herself to the end of every tie she belongs to but the one, as explained in the algorithm ASBM. This operation takes $O(|L_w|)$ time.

Every W-agent w stores information about U-agents currently matched with w in a priority queue. U-agents matched with w who are equally good for w are kept in one list, thus the priority queue contains lists. This way checking by w if there exists a U-agent $u' \in M(w)$ such that w prefers some given u to u' takes $O(\log b(w))$ time.

Each U-agent u has the counter of the number of unsaturated W-agents incident with him and whenever a saturated W-agent moves herself to the end of the ties, U-agents also decrease respective counters. Therefore checking if u is subsatellitic takes constant time.

Each W-agent w has a separate list S_w of satellitic U-agents wrt w matched with w. Every time w gets matched to some new U-agent u, who is satellitic wrt w, w adds u to S_w. When we want to check if w is insecure, we go over S_w and for each $u \in S_w$ check if u is still satellitic wrt to w . If u is not satellitic wrt to w, we remove u from S_w, otherwise we do an appropriate exchange. Once u is removed from S_w, he will not be added to S_w again. It is so since once u has no unsaturated W-agents equally good for him as w on his list, it will stay so. Hence the overall time Algorithm ASBM spends on S_w is $O(|U|)$.

Every list in the priority queue of U-agents matched with w is organized in such a way that subsatellitic U-agents proceed U-agents that are not subsatellitic. Whenever a U-agent u ceases to be subsatellitic we move him to the end of every list in every priority queue he is in. Moving u to the end of every such list takes $O(\sum_{w \in M(u)} \log(b(w))$ time. Every u ceases to be subsatellitic at most once in the course of running the algorithm. This way to see if w is uneasy wrt u, we look at the list containing u in the priority queue and see if the first U-agent on this list is subsatellitic.

Every U-agent u makes a proposal to every W-agent on L_u at most twice and to every W-agent on L'_u at most once.

Summing all the arguments together, we get that the running time of Algorithm ASBM is $O(m \min\{1, \log \max\{b(w) : w \in W\}\})$, where m denotes the number od edges in G. If $\max\{b(v) : u \in U\} < \max\{b(w) : w \in W\}$ then we can swap the roles of U-agents and W-agents. Therefore we can state.

Theorem 2. *The running time of Algorithm ASBM is* $O(m \min\{1, \log c\})$, *where* $c = min\{max\{b(v) : v \in U\}, max\{b(v) : v \in W\}\}$ *and* m *denotes the number of the edges.*

5 Correctness of Algorithm ASBM

The correctness of Algorithm ASBM is proved in a very similar way as the correctness of Algorithm GS Modified.

Lemma 3. *1) If W-agent becomes saturated, she will stay saturated. 2) An insecure W-agent w accepts every proposal. Once a saturated W-agent is not insecure, she cannot become insecure later. 3) A W-agent w matched with u can reject u only if w is saturated and a) u is satellitic wrt to w (w is insecure) or b) w is not insecure and u is the worst U-agents currently matched with w and w receives a proposal from u', who is better for w than u or c) w is not insecure and u is (one of) the worst U-agents currently matched with w and u is subsatellitic and w is uneasy wrt to u' who proposes from $L'_{u'}$ 4) A saturated W-agent w and not insecure can accept a U-agent u only if u' is at least as good for w as the worst U-agent $u \in M(w)$; moreover if u' is equally good for w as u, then w accepts u' only if w is uneasy wrt to u' and u' proposes from $L'_{u'}$.*

The proof is very similar to that of Lemma 1 and follows directly from the description of Algorithm ASBM.

Theorem 3. *Let M denote a b-matching computed by Algorithm ASBM. Then M is a $3/2$-approximation of an optimal stable b-matching.*

Proof. We will show that the graph does not contain blocking pairs and dangerous paths.

Suppose that (u, w) are a blocking pair. u is either unsaturated or there exists $w' \in M(u)$ worse for u than w. It means that at some point u proposed to w from L_u when edge (u, w) was not special. If u's proposal to w was rejected, then at that point w was saturated and not insecure and the worst $u' \in M(w)$ was at least as good as u for w (by Lemma 3) and thus (also by Lemma 3 4)) w could not later become matched to some u'' who is worse for w than u. Therefore u got accepted then and later got rejected. Since at the moment of that proposal edge (u, w) was not special, u was not satellitic wrt to w (and clearly could not become satellitic later.) By Lemma 3 3) at the moment of rejecting u W-agent w was not insecure and the worst U-agent matched with w was u. Therefore by Lemma 3 4) w could not later become matched to some u'' who is worse for her than u. A contradiction.

Suppose now that the graph contains a masculine dangerous path (u', w, u, w') such that $u \in M(w)$. Thus u is satellitic wrt to w and w is insecure. It means that at some point u proposed to w from L_u when edge (u, w) was not special. Then w was either insecure, because she is insecure now, or unsaturated. Therefore u got accepted. Later on he was clearly rejected. However by Lemma 3 3) and the description of the algorithm ABSM, it is impossible because an insecure w rejects only satellitic wrt to w U-agents.

Finally suppose that the graph contains a feminine dangerous path (u', w, u, w') such that $u \in M(w)$. Thus w is uneasy wrt to u' and u is subsatellitic. At some

point u' proposed to w from $L_{u'}$ when edge (u, w) was not special. If he got rejected then, then w was not insecure and the worst U-agent u' she was matched with was equally good for her as u. By Lemma 3 4) at that point w was uneasy wrt to u' and u' added w to the end of list $L'_{u'}$. If he got accepted at that point, then later he was rejected and also had to add w to the end of list $L'_{u'}$. When u' proposed to w from $L'_{u'}$, w was still uneasy wrt to u' (because w is uneasy wrt to u' now), hence u' was accepted (because u' proposing from $L'_{u'}$ is subsatellitic) and could not get rejected later if there were still subsatellitic U-agents matched with w, who were equally good as u' for w. A contradiction. □

Acknowledgements. I would like to thank an anonymous referee for many helpful comments.

References

1. Archer, A., Williamson, D.P.: Faster approximation algorithms for the minimum latency problem. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2003, pp. 88–96 (2003)
2. Abdulkadiroglu, A., Pathak, P.A., Roth, A.E.: Strategy-proofness versus Efficiency in Matching with Indifferences: Redesigning the NYC High School Match. American Economic Review 99(5), 1954–1978 (2009)
3. Erdil, A., Haluk, E.: What's the Matter with Tie-Breaking? Improving Efficiency in School Choice, Working Paper, Department of Economics, University of Oxford (2007)
4. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. American Mathematical Monthly 69, 9–15 (1962)
5. Halldorsson, M.M., Irving, R.W., Iwama, K., Manlove, D., Miyazaki, S., Morita, Y., Scott, S.: Approximability results for stable marriage problems with ties. Theor. Comput. Sci. 306(1-3), 431–447 (2003)
6. Halldorsson, M.M., Iwama, K., Miyazaki, S., Yanagisawa, H.: Improved approximation results for the stable marriage problem. ACM Transactions on Algorithms 3(3) (2007)
7. Irving, R.W., Manlove, D.: Finding large stable matchings. ACM Journal of Experimental Algorithmics 14 (2009)
8. Iwama, K., Miyazaki, S., Okamoto, K.: A $(2-c\frac{\log n}{n})$-Approximation Algorithm for the Stable Marriage Problem. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 349–361. Springer, Heidelberg (2004)
9. Iwama, K., Miyazaki, S., Yamauchi, N.: A $(2-c\frac{1}{\sqrt{(n)}})$-Approximation Algorithm for the Stable Marriage Problem. In: Deng, X., Du, D. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 902–914. Springer, Heidelberg (2005)
10. Iwama, K., Miyazaki, S., Yamauchi, N.: A 1.875 - approximation algorithm for the stable marriage problem. In: Bansal, N., Pruhs, K., Stein, C. (eds.) Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, pp. 288–297. SIAM, New Orleans (2007)
11. Kiraly, Z.: Better and Simpler Approximation Algorithms for the Stable Marriage Problem. Algorithmica 60(1), 3–20 (2011)

12. Lovasz, L., Plummer, M.D.: Matching Theory. Ann. Discrete Math. 29 (1986)
13. Manlove, D., Irving, R.W., Iwama, K., Miyazaki, S., Morita, Y.: Hard variants of stable marriage. Theor. Comput. Sci. 276(1-2), 261–279 (2002)
14. McDermid, E.: A 3/2-Approximation Algorithm for General Stable Marriage. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 689–700. Springer, Heidelberg (2009)
15. Mehlhorn, K.: A Faster Approximation Algorithm for the Steiner Problem in Graphs. Inf. Process. Lett. 27(3), 125–128 (1988)
16. Yanagisawa, H.: Approximation algorithms for stable marriage problems, PhD thesis, Kyoto University, Graduate School of Informatics (2007)

Simpler 3/4-Approximation Algorithms for MAX SAT

Anke van Zuylen

Max Planck Institute for Informatics,
66123, Saarbücken, Germany
anke@mpi-inf.mpg.de

Abstract. We consider the recent randomized $\frac{3}{4}$-algorithm for MAX SAT of Poloczek and Schnitger. We give a much simpler set of probabilities for setting the variables to true or false, which achieve the same expected performance guarantee. Our algorithm suggests a conceptually simple way to get a deterministic algorithm: rather than comparing to an unknown optimal solution, we instead compare the algorithm's output to the optimal solution of an LP relaxation. This gives rise to a new LP rounding algorithm, which also achieves a performance guarantee of $\frac{3}{4}$.

1 Introduction

The maximum satisfiability problem (MAX SAT) is a fundamental NP-hard problem. Given a set of variables, $x_1, \ldots, x_n$, and a set of weighted disjunctive clauses $C_1, \ldots, C_m$ of literals, where a literal is either a variable x_i or its negation $\bar{x}_i$, we want to find a truth assignment to the variables that maximizes the weight of the satisfied clauses.

Let W be the weight of all clauses. A simple approximation algorithm for MAX SAT sets each variable to true with probability $\frac{1}{2}$; by linearity of expectation, the expected weight of the satisfied clauses is at least $\frac{1}{2}W$, and, hence, this is a randomized $\frac{1}{2}$-approximation algorithm. This algorithm can be derandomized using the method of conditional expectation, which gives rise to the following algorithm: Consider the variables one at a time. For a clause C_j with weight w_j that is not yet satisfied by the assignment of the variables considered so far, let c_j be the number of variables occurring in C_j for which the truth assignment has not yet determined. Define the modified weight of C_j as $\mu(C_j) = w_j \left(\frac{1}{2}\right)^{c_j}$. Note that this is the expected weight of clause C_j that is *not* satisfied, if the remaining variables are set to true with probability $\frac{1}{2}$. We now set the next variable x_i to true if the modified weight of the clauses containing x_i is greater than or equal to the modified weight of the clauses containing $\bar{x}_i$, and to false otherwise. This deterministic algorithm is due to Johnson [6] and is known as Johnson's algorithm. The fact that it can be interpreted as the derandomization of the randomized algorithm that sets each variable to true with probability $\frac{1}{2}$ was noted by Yannakakis [9]. Chen, Friesen and Zhang [2] showed that the approximation ratio of the derandomized algorithm is in fact $\frac{2}{3}$; see also Engebretsen [4] for a simplified analysis.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 188–197, 2012.

Better approximation algorithms are known, both for the general case and for certain special cases, but until recently, all of these used the optimal solution to a linear program or semidefinite program. See for example Yannakakis [9] and Goemans and Williamson [5]. The best known approximation algorithm is due to Avidor, Berkovitch and Zwick [1] and achieves a guarantee of 0.7968.

Very recently, Poloczek and Schnitger [8] gave the first approximation algorithm with performance guarantee $\frac{3}{4}$ that is purely combinatorial. They define a randomized variant of Johnson's algorithm, which sets variable x_i to true or false with probability proportional to the modified weight of the clauses containing x_i and $\bar{x}_i$ respectively. They then show how to slightly modify these probabilities so that the expected weight of the clauses satisfied by the algorithm is at least $\frac{3}{4}$ of the weight of the optimal solution.

The probabilities determined by the algorithm are rather complicated, and they depend on previous decisions by the algorithm. Derandomization of this algorithm seems therefore highly non-trivial. In fact, Poloczek [7] shows that, under certain assumptions, no deterministic variant of the algorithm of Poloczek and Schnitger [8] can achieve the same guarantee: Poloczek shows that no deterministic *adaptive priority algorithm* can achieve an approximation ratio of $\frac{3}{4}$. Priority algorithms are a formalization of greedy algorithms, and need to make an irrevocable decision when a data item is revealed. In the setting considered by Poloczek, a data item is the name of a variable, say x; the set of clauses that contain the variable x; and for each such clause, the data item contains the sign of x in the clause, the weight, and the other variables appearing in the clause (but not whether these appear negated or not). Based on this information, the algorithm has to decide whether to set x to true or false. In an adaptive priority algorithm, the algorithm may adaptively change the order in which it considers the data items, but when the data item corresponding to variable x is revealed, it still needs to irrevocably determine the value of x.

It may however still be the case that a deterministic variant, which is not an (adaptive) priority algorithm, achieves a guarantee of $\frac{3}{4}$. In this paper, we give a simple expression for the probability with which to set the next variable to true or false, which gives the same performance guarantee as the algorithm of Poloczek and Schnitger [8]. Our probabilities are not necessarily the same as those given by Poloczek and Schnitger [8], but they do satisfy the inequalities that are required for their analysis (and, by extension, our version of the analysis) to hold. Although the expression of the probabilities is simple, the probabilities still depend on the past decisions made by the algorithm, and, hence, the question whether this algorithm can be derandomized remains non-trivial. However, if we allow our algorithm to use linear programming, derandomization becomes relatively straightforward. Our second result is therefore a new deterministic LP rounding algorithm, which achieves an approximation ratio of $\frac{3}{4}$.

The remainder of this paper is structured as follows: we begin in Section 2 by introducing the notion of a potential function, which is implicitly used in the analysis of Poloczek and Schnitger. We summarize some key ideas of their analysis in terms of the potential function. We then give a new randomized

algorithm which has very simple probabilities of setting the next variable to true or false, and we prove that it satisfies the conditions derived in Section 2. Our new algorithm suggests a conceptually simple way to get a deterministic algorithm: rather than comparing to an unknown optimal solution, we can instead compare the algorithm's output to the optimal solution of an LP relaxation. This gives rise to the new rounding algorithm described in Section 4.

2 Analysis with a Potential Function

Let the input be a set of variables $x_1, \ldots, x_n$, and a set of disjunctive clauses $C_1, \ldots, C_m$ with weights $w_1, \ldots, w_m \geq 0$, where the literals in the clauses are variables or their negation. Let $W = \sum_{j=1}^{m} w_j$. The algorithms we consider iteratively determine the value (either 1 (true) or 0 (false)) to which we set variables $x_1, \ldots, x_n$, and our aim is to prove that the expected weight of the satisfied clauses is at least $\frac{3}{4}$ times the weight of the optimal assignment.

For a given index i, let $SAT(i)$ be the weight of the clauses that are satisfied by the algorithm's values for $x_1, \ldots, x_i$, and let $UNSAT(i)$ be the weight of the clauses which contain only $x_1, \ldots, x_i$, or their negations, and that are not satisfied by the chosen values. Suppose we have already determined the assignment for $x_1, \ldots, x_{i-1}$, and the algorithm now fixes the assignment for x_i. Then $SAT(i) - SAT(i-1)$ is the weight of the clauses that become satisfied by the algorithm's assignment for x_i (and that were not already satisfied by the assignment for $x_1, \ldots, x_{i-1}$), and $UNSAT(i) - UNSAT(i-1)$ is the weight of the clauses that become unsatisfiable by the assignment to x_i. If for all i, we could determine an assignment such that

$$(SAT(i) - SAT(i-1)) - 3(UNSAT(i) - UNSAT(i-1)) \geq 0, \tag{1}$$

then this would imply a $\frac{3}{4}$-approximation algorithm: Note that $\sum_{i=1}^{m} \Big((SAT(i) - SAT(i-1)) - 3(UNSAT(i) - UNSAT(i-1))\Big) = SAT(n) - 3UNSAT(n)$, where $SAT(n)$ is the weight of the clauses satisfied by the algorithm's solution, and $UNSAT(n)$ is the weight of the clauses that the algorithm does not satisfy, i.e. $UNSAT(N) = W - SAT(n)$. So we would get that $SAT(n) - 3(W - SAT(n)) \geq 0$, or $SAT(n) \geq \frac{3}{4}W$.

There does not always exist an assignment to i such that (1) holds, but note that we only need the inequality to hold, summed over all i. We therefore introduce the idea of a potential function Φ. This idea is implicit in the analysis of Poloczek and Schnitger [8]. One can think of Φ as a "bank account" for the algorithm. In the course of the algorithm, we may add or remove some amount to the potential function to allow us to satisfy the inequality $(SAT(i) - SAT(i-1) - 3(UNSAT(i) - UNSAT(i-1)) \geq 0$.

More precisely, let $\Phi(i)$ be the value of the potential function after determining the truth assignment of variable x_i (where $\Phi(0)$ is the potential function at the start of the algorithm). Let OPT be the weight of the satisfied clauses in an

optimal solution. The potential function Φ, combined with the algorithm, must satisfy the following three properties:

(i) $\Phi(0) \leq 3(W - OPT)$;
(ii) $\Phi(n) \geq 0$;
(iii) For each variable x_i, the algorithm (randomly) determines a truth assignment to x_i such that

$$E\big[SAT(i) - SAT(i-1) - 3(UNSAT(i) - UNSAT(i-1))\big] \geq E\big[\Phi(i) - \Phi(i-1)\big].$$

If we have a potential function Φ with an algorithm that together satisfy these three properties, then $E\big[SAT(n) - 3(W - SAT(n))\big] \geq \Phi(n) - \Phi(0) \geq -\Phi(0) \geq 3(OPT - W)$, which gives $E\big[SAT(n)\big] \geq \frac{3}{4}OPT$.

We remark that the potential functions in this paper will in fact have $\Phi(0) = 2(W - OPT)$, which is less than what is allowed by (i), but that increasing it to $3(W - OPT)$ does not help in our analysis.

2.1 Poloczek and Schnitger's Potential Function

Poloczek and Schnitger [8] do not explicitly define the idea of a potential function, but their analysis implicitly uses the following potential function. Let $x_i = x_i^*$ for $i = 1, \ldots, n$ be an optimal solution, where each x_i^* is either 1 (true) or 0 (false). Let x_i^a be the truth assignment to x_i by the algorithm's solution, if x_i has already been determined. Let "time i" be the time when the algorithm has determined the truth assignment to $x_1, \ldots, x_i$. We'll say a clause is alive at time i if it contains some literal from $\{x_{i+1}, \ldots, x_n\}$, and it is not (yet) satisfied by setting $x_1 = x_1^a, \ldots, x_i = x_i^a$. We'll say a live clause is contradictory at time i if it is not satisfied by setting $x_1 = x_1^a, \ldots, x_i = x_i^a$ according to the algorithm's solution, and $x_{i+1} = x_{i+1}^*, \ldots, x_n^*$. We will make sure that at any point in time $\Phi(i)$ is (at least) twice the weight of the clauses that are alive and contradictory at time i. Note that we thus have the $\Phi(0) = 2(W - OPT)$.

Let $W_i, \overline{W}_i$ be the weight of the clauses that are alive at time $i-1$ and contain x_i and $\bar{x}_i$ respectively, but do not contain $x_{i+1}, \ldots, x_n$. Let $F_i, \overline{F}_i$ be the weight of the remaining clauses that are alive at time $i-1$ and that contain x_i and $\bar{x}_i$ respectively. We note that $W_i, \overline{W}_i, F_i, \overline{F}_i$ are random variables that are determined by the algorithm's decisions for $x_1, \ldots, x_{i-1}$. Let 1_A be the indicator function that is 1 if A holds and 0 otherwise. A contradictory clause at time $i-1$ is not contradictory at time i when it is no longer alive at time i because either it becomes satisfied or it has no literals in $x_{i+1}, \ldots, x_n$. We can thus lower bound the weight of the contradictory clauses that are alive at time $i-1$ and not alive at time i by $W_i \mathbf{1}_{\{x_i^*=0\}} + \overline{W}_i \mathbf{1}_{\{x_i^*=1\}}$.

On the other hand, the only clauses that can become contradictory when going from time $i-1$ to time i are clauses that are alive at time $i-1$ and at time i, that contain either x_i or $\bar{x}_i$, and for which the algorithm's setting for x_i is not the same as the setting in the optimal solution. Hence we can upper bound the weight of the clauses that become contradictory by $\mathbf{1}_{\{x_i^*=0\}} \mathbf{1}_{\{x_i=1\}} \overline{F}_i + \mathbf{1}_{\{x_i^*=1\}} \mathbf{1}_{\{x_i=0\}} F_i$.

We thus have that

$$\Phi(i) - \Phi(i-1) \leq 2\left(-W_i + \mathbf{1}_{\{x_i=1\}}\overline{F}_i\right)\mathbf{1}_{\{x_i^*=0\}} + 2\left(-\overline{W}_i + \mathbf{1}_{\{x_i=0\}}F_i\right)\mathbf{1}_{\{x_i^*=1\}}.$$

We note that the expression $E\left[c' - c\right]$ in the analysis of Poloczek and Schnitger [8] is equal to $E\left[\Phi(i) - \Phi(i-1)\right]$, and that a similar inequality is given in their Lemma 2.2.

On the other hand,

$$\begin{aligned} &SAT(i) - SAT(i-1) - 3(UNSAT(i) - UNSAT(i-1)) \\ &\quad = \mathbf{1}_{\{x_i=1\}}(W_i + F_i - 3\overline{W}_i) + \mathbf{1}_{\{x_i=0\}}(\overline{W}_i + \overline{F}_i - 3W_i) \end{aligned}$$

Let p be the probability that the algorithm set x_i to 1. Then, in order to satisfy property (iii), we need:

$$\begin{aligned} &p(W_i + F_i - 3\overline{W}_i) + (1-p)(\overline{W}_i + \overline{F}_i - 3W_i) \\ &\quad - 2\left(-W_i + p\overline{F}_i\right)\mathbf{1}_{\{x_i^*=0\}} - 2\left(-\overline{W}_i + (1-p)F_i\right)\mathbf{1}_{\{x_i^*=1\}} \geq 0. \end{aligned} \tag{2}$$

3 A New Combinatorial Randomized Algorithm

In the following lemma and its proof, we will define $\frac{c}{0} = \infty$ if $c \geq 0$ and $\frac{c}{0} = -\infty$ if $c < 0$.

Lemma 1. *Consider the randomized algorithm that iteratively determines the assignment to $x_1, \ldots, x_n$ as follows: Given the assignment of $x_1, \ldots, x_{i-1}$, let $W_i, \overline{W}_i$ be the weight of the clauses that are not yet satisfied and contain x_i and $\bar{x}_i$ respectively, but do not contain $x_{i+1}, \ldots, x_n$. Let $F_i, \overline{F}_i$ be the weight of the remaining clauses that are not yet satisfied and that contain x_i and $\bar{x}_i$ respectively. Let $\alpha = \frac{W_i + F_i - \overline{W}_i}{F_i + \overline{F}_i}$, and let x_i be set to 1 with probability*

$$p = \begin{cases} 0 & \text{if } \alpha \leq 0, \\ \alpha & \text{if } \alpha \in (0,1), \\ 1 & \text{if } \alpha \geq 1. \end{cases}$$

Then the expected weight of the clauses satisfied by the algorithm is at least $\frac{3}{4}OPT$.

Proof. We will show that inequality (2) holds, by giving a lower bound B on

$$2\left(W_i - p\overline{F}_i\right)\mathbf{1}_{\{x_i^*=0\}} + 2\left(\overline{W}_i - (1-p)F_i\right)\mathbf{1}_{\{x_i^*=1\}}, \tag{3}$$

in the case when $\alpha \leq 0, \alpha \geq 1$ and $\alpha \in (0,1)$, and showing that for each of these cases, $p(W_i + F_i - 3\overline{W}_i) + (1-p)(\overline{W}_i + \overline{F}_i - 3W_i) + B \geq 0$.

We first consider the case when $\alpha \leq 0$, i.e., when $W_i + F_i \leq \overline{W}_i$. Then $p = 0$ and $W_i - p\overline{F}_i = W_i \leq \overline{W}_i - F_i = \overline{W}_i - (1-p)F_i$, so (3) is at least $2W_i - p\overline{F}_i = 2W_i$. Therefore, the lefthand side of (2) is at least $\overline{W}_i + \overline{F}_i - 3W_i + 2W_i = \overline{W}_i + \overline{F}_i - W_i$.

Note that this cannot be negative, since combined with $W_i + F_i - \overline{W}_i \leq 0$ this would give $F_i + \overline{F}_i < 0$.

If $\alpha \geq 1$, then $W_i + F_i - \overline{W}_i \geq F_i + \overline{F}_i$, i.e., $W_i - \overline{F}_i \geq \overline{W}_i$. Since $p = 1$, (3) is at least $2\overline{W}_i$. So the lefthand side of (2) is at least $W_i + F_i - \overline{W}_i$ and this cannot be negative, as this would imply $F_i + \overline{F}_i < 0$ by the fact that $\overline{W}_i + \overline{F}_i - W_i \leq 0$.

Finally, if $\alpha \in (0, 1)$, then we have that $p = \alpha$ and, by definition of α, $-W_i + p\overline{F}_i = -\overline{W}_i + (1-p)F_i$. Hence, the quantity in (3) does not depend on whether x_i^* is zero or one, since it is either $2W_i - 2p\overline{F}_i$ or $2\overline{W}_i - 2(1-p)F_i$ which are equal. Thus (3) is also equal to $p(2W_i - 2p\overline{F}_i) + (1-p)(2\overline{W}_i - 2(1-p)F_i)$. Plugging this into (2) gives

$$\begin{aligned}
&p(W_i + F_i - 3\overline{W}_i) + (1-p)(\overline{W}_i + \overline{F}_i - 3W_i) + \\
&2p(W_i - p\overline{F}_i) + 2(1-p)(\overline{W}_i - (1-p)F_i) \\
=&(6p-3)W_i - (6p-3)\overline{W}_i + (5p - 2p^2 - 2)F_i - (2p^2 + p - 1)\overline{F}_i \\
=&(2p-1)(3W_i + (1-p)F_i - 3\overline{W}_i - p\overline{F}_i + F_i - \overline{F}_i) \\
=&(2p-1)(2W_i + F_i - 2\overline{W}_i - \overline{F}_i),
\end{aligned}$$

where the first two equalities follow by rearranging terms, and the last equality uses the fact that $W_i + (1-p)F_i = \overline{W}_i + p\overline{F}_i$. Now, either $p \geq \frac{1}{2}$ in which case $2p - 1 \geq 0$ and $2W_i + F_i \geq 2W_i + 2(1-p)F_i = 2\overline{W}_i + 2p\overline{F}_i \geq 2\overline{W}_i + \overline{F}_i$, so $2W_i + F_i - 2\overline{W}_i - \overline{F}_i \geq 0$. Otherwise, $p < \frac{1}{2}$, in which case $2p - 1 < 0$ and also $2\bar{W}_i + \overline{F}_i > 2W_i + F_i$. Hence in either case the inequality (2) holds. □

Remark 2. Let α be defined as in Lemma 1. If we let

$$p = \begin{cases} 0 & \text{if } \alpha \leq \frac{1}{3}, \\ \alpha & \text{if } \alpha \in (\frac{1}{3}, \frac{2}{3}), \\ 1 & \text{if } \alpha \geq \frac{2}{3}, \end{cases}$$

then the expected weight of the clauses satisfied by the algorithm is also at least $\frac{3}{4}OPT$.

Proof. We only need to verify that inequality (2) holds for this choice of p, if $\alpha \in (0, \frac{1}{3}]$ or if $\alpha \in [\frac{2}{3}, 1)$. If $\alpha \in (0, \frac{1}{3}]$ then $p = 0$, and we note that $W_i - p\overline{F}_i \geq W_i - \alpha\overline{F}_i = \overline{W}_i - (1-\alpha)F_i \geq \overline{W}_i - (1-p)F_i$. Hence (3) is at least $2\overline{W}_i - 2(1-p)F_i$, and therefore the lefthand side of (2) is at least

$$\overline{W}_i + \overline{F}_i - 3W_i + 2\overline{W}_i - 2F_i.$$

Now, note that $3\overline{W}_i + \overline{F}_i \geq 3\overline{W}_i + 3\alpha\overline{F}_i = 3W_i + 3(1-\alpha)F_i \geq 3W_i + 2F_i$ hence (2) holds.

Similarly, if $\frac{2}{3} \leq \alpha < 1$, then setting $p = 1$ will give that (3) is at least $2W_i - 2\overline{F}_i$, and hence the lefthand side of (2) is at least $3W_i + F_i - 3\overline{W}_i - 2\overline{F}_i$ and this is nonnegative by the fact that $\alpha \geq \frac{2}{3}$. □

Note that one way to view an iteration of the algorithm is as a 2-player-zero-sum game. We get to choose p, our probability of playing $x_i = 1$, and the opponent

gets to choose q, which is the optimum's probability of playing $x_i = 1$. We are trying to maximize

$$p(W_i+F_i-3\overline{W}_i)+(1-p)(\overline{W}_i+\overline{F}_i-3W_i)+2(1-q)(W_i-p\overline{F}_i)+2q(\overline{W}_i-(1-p)F_i)$$

and the opponent is trying to minimize this quantity. We show that the value of this game is nonnegative by showing that there exists a randomized strategy p such that for any strategy q the outcome is nonnegative. When $W_i + F_i < \overline{W}_i$ then $\overline{W}_i - (1-p)F_i \geq W_i - p\overline{F}_i$ for any $p \geq 0$, and hence $q = 0$ is an optimal strategy for the opponent. It is easily verified that, given $q = 0$, $p = 0$ is an optimal strategy for the algorithm. Similarly, when $\overline{W}_i + \overline{F}_i < W_i$, then $q = 1, p = 1$ are a pair of optimal strategies. In all other cases, the proof of Lemma 1 shows that $q = (1-p)$ is an optimal strategy for the opponent, given our strategy.

Note that we thus achieve an expected non-negative value even if we allow fractional values $q \in [0, 1]$. Hence, our algorithm achieves at least $\frac{3}{4}$ of the weight of any fractional assignment as well; something that was recently shown by Poloczek [7] for the algorithm in [8].

In fact, allowing the opponent to use fractional assignments makes it easy to derandomize the algorithm: we can compute the optimum's probability q of playing $x_i = 1$ by solving a linear program. Given this information, there exists a pure strategy p that achieves a nonnegative value. This gives rise to the deterministic algorithm in the next section.

4 A New Deterministic LP Rounding Algorithm

Let q_i be the variable in the linear program corresponding to the decision $x_i = 1$, and let z_j be a variable corresponding to the j-th clause, and let w_j be the weight of the j-th clause. We let P_j be the indices of the literals i such that x_i appears in the clause, and N_j the indices of the literals such that $\bar{x}_i$ appears in the clause. Then the linear programming relaxation is:

$$\begin{aligned}
&\min \sum_j w_j z_j \\
&\text{s.t.} \sum_{i \in P_j} q_i + \sum_{i \in N_j} (1 - q_i) \geq z_j && \text{for } j = 1, \ldots, m \\
&0 \leq z_j \leq 1 && \text{for } j = 1, \ldots, m \\
&0 \leq q_i \leq 1 && \text{for } i = 1, \ldots, n
\end{aligned}$$

For ease of notation, we again define $\frac{c}{0} = \infty$ if $c \geq 0$ and $\frac{c}{0} = -\infty$ if $c < 0$.

Lemma 3. *Let q^* be an optimal LP solution, with objective value OPT_{LP}. Using the parameters defined in Lemma 1, let α again be defined as $\frac{W_i+F_i-\overline{W}_i}{F_i+\overline{F}_i}$, and let x_i be set to 1 with probability*

$$p = \begin{cases} 0 & \text{if } \alpha \leq 0, \text{ or if } \alpha \in (0,1) \text{ and } q_i^* < (1-\alpha)/2\alpha \\ 1 & \text{if } \alpha \geq 1, \text{ or if } \alpha \in (0,1) \text{ and } q_i^* \geq (1-\alpha)/2\alpha. \end{cases}$$

Then the weight of the clauses satisfied by the algorithm is at least $\frac{3}{4}OPT_{LP}$.

Proof. We'll again say a clause is alive at time i if it contains some literal from $\{x_{i+1}, \ldots, x_n\}$, and it is not satisfied yet by the algorithm's solution on $x_1, \ldots, x_i$. We will say the contradictory weight of a live clause j at time i is $w_j(1 - \min\left\{1, \sum_{i' \in P_j : i' \geq i} q^*_{i'} + \sum_{i' \in N_j : i' \geq i}(1 - q^*_{i'})\right\})$.

We define the potential function $\Phi(i)$ to be twice the contradictory weight of the live clauses. Initially, $\Phi(0) = 2(W - OPT_{LP}) \leq 2(W - OPT)$, since all clauses are alive at time 0, and the contradictory weight of clause j at time 0 is $w_j(1 - z_j)$.

We now consider $\Phi(i) - \Phi(i-1)$. Note that $\Phi(i)$ does not contain any contradictory weight for clauses that are alive at time $i-1$ that are not alive at time i. Hence Φ drops by at least $2W_i(1 - q^*_i) + 2\overline{W}_i q^*_i$. On the other hand, the contradictory weight for any clause that is still alive at time i will increase only if the clause contains x_i or $\bar{x}_i$ (i.e. the clause is contained in F_i or $\overline{F}_i$ respectively) and it is not satisfied by the algorithm's setting (i.e. if we set $x_i = 0$ or $x_i = 1$ respectively). The increase in the contradictory weight is thus at most $2q^*_i F_i$ if we set $x_i = 0$, and $2(1 - q^*_i)\overline{F}_i$ if we set $x_i = 1$.

Hence we get that

$$\Phi(i) - \Phi(i-1) \leq 2\left(-W_i + \mathbf{1}_{\{x_i=1\}}\overline{F}_i\right)(1 - q^*_i) + 2\left(-\overline{W}_i + \mathbf{1}_{\{x_i=0\}}F_i\right)q^*_i.$$

At time n, there are no live clauses, and hence the contradictory weight of the live clauses is zero, or, $\Phi(n) \geq 0$.

As before,

$$\begin{aligned} &SAT(i) - SAT(i-1) - 3(UNSAT(i) - UNSAT(i-1)) \\ &\quad = \mathbf{1}_{\{x_i=1\}}(W_i + F_i - 3\overline{W}_i) + \mathbf{1}_{\{x_i=0\}}(\overline{W}_i + \overline{F}_i - 3W_i) \end{aligned}$$

Let p be the probability with which we set x_i to 1 (which is 1 if $\alpha \geq 1$ or if $\alpha \in (0,1)$ and $q^*_i \geq (1-\alpha)/(2\alpha)$ and 0 otherwise). Then, we need to show that p satisfies

$$\begin{aligned} &p(W_i + F_i - 3\overline{W}_i) + (1-p)(\overline{W}_i + \overline{F}_i - 3W_i) \\ &\quad - 2\left(-W_i + p\overline{F}_i\right)(1 - q^*_i) - 2\left(-\overline{W}_i + (1-p)F_i\right)q^*_i \geq 0. \end{aligned} \tag{4}$$

This is the same as (2), except that we replaced $\mathbf{1}_{\{x^*_i=1\}}$ by q^*_i and $\mathbf{1}_{\{x^*_i=0\}}$ by $(1 - q^*_i)$. Note that the proof of Lemma 1 shows that if $\alpha \leq 0$ or $\alpha \geq 1$, then (4) holds for our choice of p, for any $q^*_i \in [0,1]$. Hence, we only need to check the case when $\alpha \in (0,1)$.

If we set $p = 0$ then the lefthand side of (4) becomes

$$\begin{aligned} &\overline{W}_i + \overline{F}_i - 3W_i + 2W_i(1 - q^*_i) + 2\overline{W}_i q^*_i - 2F_i q^*_i \\ &\quad = (1 + 2q^*_i)\left(\overline{W}_i + \frac{1}{1 + 2q^*_i}\overline{F}_i - W_i - \frac{2q^*_i}{1 + 2q^*_i}F_i\right). \end{aligned}$$

To see that this is non-negative, note that, since $p = 0$, and $\alpha \in (0,1)$, we have that $q_i^* < \frac{1-\alpha}{2\alpha}$. Therefore, $\frac{1}{1+2q_i^*} > \alpha$, and $\frac{2q_i^*}{1+2q_i^*} < 1-\alpha$. So, (4) is at least $(1+2q_i^*)\left(\overline{W}_i + \alpha\overline{F}_i - W_i - \alpha F_i\right)$. Finally, note that $\overline{W}_i + \alpha\overline{F}_i - W_i - (1-\alpha)F_i = 0$, by the definition of α.

Similary, if we set $p = 1$ then the lefthand side of (4) becomes

$$\begin{aligned} &W_i + F_i - 3\overline{W}_i + 2W_i(1-q_i^*) + 2\overline{W}_i q_i^* - 2\overline{F}_i(1-q_i^*) \\ &\quad = (3-2q_i^*)\left(W_i + \frac{1}{3-2q_i^*}F_i - \overline{W}_i - \frac{2-2q_i^*}{3-2q_i^*}\overline{F}_i\right). \end{aligned}$$

We claim that for any $\alpha \in (0,1)$

$$\frac{2-3\alpha}{2-2\alpha} \leq \frac{1-\alpha}{2\alpha}.$$

This can be seen by noting that $\frac{(2\alpha-1)^2}{\alpha(1-\alpha)} \geq 0$, and

$$\frac{(2\alpha-1)^2}{\alpha(1-\alpha)} = \frac{4\alpha^2-4\alpha+1}{\alpha(1-\alpha)} = -\frac{2\alpha-3\alpha^2}{\alpha(1-\alpha)} + \frac{\alpha^2-2\alpha+1}{\alpha(1-\alpha)} = -\frac{2-3\alpha}{1-\alpha} + \frac{1-\alpha}{\alpha}.$$

Hence, since $p = 1$ implies that $q_i^* \geq \frac{1-\alpha}{2\alpha}$, we also have $q_i^* \geq \frac{2-3\alpha}{2-2\alpha}$. Therefore, $\frac{1}{3-2q_i^*} \geq 1-\alpha$. So, we get that

$$\begin{aligned} &(3-2q_i^*)\left(W_i + \frac{1}{3-2q_i^*}F_i - \overline{W}_i - \frac{2-2q_i^*}{3-2q_i^*}\overline{F}_i\right) \\ &\quad \geq (3-2q_i^*)\left(W_i + \alpha F_i - \overline{W}_i - (1-\alpha)\overline{F}_i\right) \\ &\quad \geq 0. \end{aligned}$$

where the final inequality follows from the fact that $3-2q_i^* \geq 1$ and $W_i + \alpha F_i - \overline{W}_i - (1-\alpha)\overline{F}_i = 0$. □

5 Conclusion and Future Directions

The question remains whether there exists a deterministic algorithm that achieves an approximation ratio of $\frac{3}{4}$, which does not use sophisticated techniques such as linear programming. Poloczek and Schnitger [8] gave the first randomized algorithm that achieves this, and our simplified analysis makes it easier to see the need for randomization in their algorithm to "foil an adversarial optimum". We also show that it is possible to derandomize (our version of) their algorithm if one has an optimal solution to a linear programming relaxation. The upcoming paper of Poloczek [7] shows that no adaptive priority algorithm can achieve a guarantee of $\frac{3}{4}$, but this does not completely exclude the existence of a deterministic combinatorial $\frac{3}{4}$-approximation algorithm. For instance, an algorithm that looks at all data items and then chooses the next variable to be determined is not an adaptive priority algorithm, and the upper bound of Poloczek [7] does

not apply. Moreover, there seems to be some evidence that carefully choosing the next variable to be determined could lead to improved results by a recent result of Costello, Shapira and Tetali [3]: They showed that Johnson's algorithm has a guarantee strictly better than $\frac{2}{3}$ if the variables are considered in a random order, whereas the best possible guarantee is $\frac{2}{3}$ if the variables are considered in a fixed order.

Acknowledgements. The author thanks David Williamson for pointing her to the question whether a deterministic variant of the algorithm of Poloczek and Schnitger exists.

References

1. Avidor, A., Berkovitch, I., Zwick, U.: Improved Approximation Algorithms for MAX NAE-SAT and MAX SAT. In: Erlebach, T., Persinao, G. (eds.) WAOA 2005. LNCS, vol. 3879, pp. 27–40. Springer, Heidelberg (2006)
2. Chen, J., Friesen, D.K., Zheng, H.: Tight bound on Johnson's algorithm for maximum satisfiability. J. Comput. Syst. Sci. 58, 622–640 (1999)
3. Costello, K.P., Shapira, A., Tetali, P.: Randomized greedy: new variants of some classic approximation algorithms. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 647–655. SIAM (2011)
4. Engebretsen, L.: Simplified tight analysis of Johnson's algorithm. Inf. Process. Lett. 92(4), 207–210 (2004)
5. Goemans, M.X., Williamson, D.P.: New $\frac{3}{4}$ -approximation algorithms for the maximum satisfiability problem. SIAM J. Discrete Math. 7(4), 656–666 (1994)
6. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. System Sci. 9, 256–278 (1974); Fifth Annual ACM Symposium on the Theory of Computing, Austin, Tex. (1973)
7. Poloczek, M.: Bounds on Greedy Algorithms for MAX SAT. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 37–48. Springer, Heidelberg (2011)
8. Poloczek, M., Schnitger, G.: Randomized variants of Johnson's algorithm for MAX SAT. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 656–663. SIAM (2011)
9. Yannakakis, M.: On the approximation of maximum satisfiability. Journal of Algorithms 17, 475–502 (1994)

On Online Algorithms with Advice for the k-Server Problem

Marc P. Renault[1] and Adi Rosén[2,⋆]

[1] LIAFA, Univerité Paris Diderot - Paris 7; and UPMC
mrenault@liafa.jussieu.fr
[2] CNRS and Univerité Paris Diderot - Paris 7
adiro@lri.fr

Abstract. We consider the model of online computation with advice [5]. In particular, we study the k-server problem under this model. We prove two upper bounds for this problem. First, we show a $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$-competitive online algorithm for general metric spaces with b bits of advice per request, where $3 \leq b \leq \log k$. This improves upon the recent result of [1]. Moreover, we believe that our algorithm and our analysis are more intuitive and simpler than those of [1]. Second, we give a 1-competitive online algorithm for trees which uses $2 + 2\lceil \log(p+1) \rceil$ bits of advice per request, where p is the caterpillar dimension of the tree.

Keywords: online computation with advice, k-server problem, online algorithms, competitive analysis.

1 Introduction

Online algorithms have been the subject of intense research activity over the past decades. The traditional setting is that of an online algorithm that does not have any knowledge about the future and that of a worst-case analysis using competitive analysis (cf. [3]). In the present paper we consider a model recently introduced by Emek et al. [5], dubbed *online computation with advice*, which is aimed at relaxing the "absolutely no knowledge about the future" setting and at giving a general framework to quantify the interplay between the amount of knowledge about the future and the possible improvement in the competitive ratio. Roughly speaking, this model augments the power of the online algorithm by a series of queries. Each query is issued by the online algorithm when it receives a new request. These queries map the whole request sequence, including future requests, to some domain of *advice*. Thus, they provide the online algorithm with some information about the future. One is typically interested in the interplay between the size of the domain of advice, i.e., how many bits of advice are received with each request, and the attainable competitive ratio. For a formal definition of this model, see Section 2.

⋆ Research supported in part by ANR projects QRAC and ALADDIN.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 198–210, 2012.

A number of results for various online problems have been obtained in the above model and in a variant thereof introduced by Böckenhauer et al. [2]. In the present paper, we consider the k-server problem under the model of online computation with advice. Emek et al. [5] gave an upper bound of $k^{O(\frac{1}{b})}$ on the competitive ratio of deterministic algorithms on general metric spaces, where b is the number of bits of advice per request. This upper bound was recently improved to $2\left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil$ by Böckenhauer et al. [1]. Better bounds for specific metric space where also given (see the paragraph "Related Work" below).

In this paper, we improve the upper bound for deterministic k-server algorithms with advice on general metric spaces by giving a deterministic online algorithm with b bits of advice per request, for $b \geq 3$, whose competitive ratio is $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$. While the improvement over the previous result is only about a factor of 2, we believe that our algorithm and analysis are more intuitive and simpler than previous ones, and may lead to further improvements in the upper bound. Also, we consider the class of metric spaces of finite trees, and give a 1-competitive deterministic online algorithm. The number of bits of advice per request used by this algorithm is $2 + 2\lceil \log(p+1) \rceil$, where p is the caterpillar dimension of the tree (cf. [8]). We use this measure for the tree since it is at most the height of the tree, and it is at most $\log N$, where N is the number of nodes in the tree [8]. This measure is preferable over other measures, such as height, because it remains constant for degenerate trees, such as the line, the spider and the caterpillar.

Related Work. The model of online computation with advice considered in the present paper was introduced by Emek et al. [5]. In that paper, the authors gave tight bounds of $\Theta(\log n/b)$ for deterministic and randomized online algorithms with advice for Metrical Task Systems, where n is the number of states of the systems and b is the number of bits of advice per request. They also gave a deterministic online algorithm with advice for the k-server problem which is $k^{O(\frac{1}{b})}$-competitive, where $\Theta(1) \leq b \leq \log k$. This was improved by Böckenhauer et al. [1] who gave a deterministic online algorithm with advice for general metric spaces with a competitive ratio of $2\left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil$. Böckenhauer et al., also, gave a deterministic algorithm for the Euclidean plane with a competitive ratio of $\frac{1}{1-2\sin(\frac{\pi}{2^b})}$, where $b \geq 3$ is the number of bits of advice per request. For the uniform metric space (the problem of paging), a 1-competitive deterministic online algorithm with 1 bit of advice per request is implicit in [4].

Böckenhauer et al. [2] introduced a somewhat similar model for online algorithms with advice, where the advice is a single tape of bits instead of being given separately for each request. This allows an algorithm to read a different number of bits of advice per request, but it requires that the online algorithm knows how many bits of advice to read with each request. Thus, the two models are, in general, incomparable. We note that upper bounds in the model of [5], as those given in the present paper, carry over to the model of [2]. Several results were given in this model [4,2,6,7,1]. For example, in [4,2], the authors explore

the number of bits of advice required for deterministic and randomized paging algorithms, scheduling algorithms and routing algorithms to be 1-competitive.

2 Preliminaries

Online algorithms receive their input piece by piece. Each piece, or request, is an element of some set R, and the algorithm receives a *request sequence* denoted $\sigma = r_1, \ldots, r_n$, where $n = |\sigma|$ and r_i is the ith request. An online algorithm must perform all of the actions pertaining to a request before receiving the subsequent requests. These actions incur some cost to the online algorithm. In this paper, we consider only minimization problems.

We use the definition of *deterministic online algorithms with advice* as presented in [5]. An online algorithm with advice is defined as a request-answer game that consists of a request set, R; a sequence of finite nonempty answer sets, $A_1, A_2, \ldots$; and a sequence of cost functions, $\text{cost}_n : R^n \times A_1 \times A_2 \times \cdots \times A_n \to \mathbb{R}^+ \cup \{\infty\}$ for $n = 1, 2, \ldots$. In addition, online algorithms with advice have access via a query to an advice space, U, which is a finite set. The advice space has a size of 2^b, where $b \geq 0$ is the number of bits of advice provided to the algorithm with each request. With each request, the online algorithm receives some advice that is defined by a function, $u_i : R^* \to U$, that is applied to the whole request sequence, including future requests. A deterministic online algorithm with advice can, thus, be represented as a sequence of pairs (g_i, u_i), where $g_i : R^i \times U^i \to A_i$ for $i = 1, 2, \ldots$. The action that the online algorithm takes upon receiving request r_i is a function of the first i requests, $r_1, \ldots, r_i$, and the advice received so far, $u_1(\sigma), \ldots, u_i(\sigma)$.

The cost of the online algorithm is defined as $\text{ALG}(\sigma) = \text{cost}_n(\sigma, \text{ALG}[\sigma])$, where $\text{ALG}[\sigma] = < a_1, \ldots, a_n > \in A_1 \times \cdots \times A_n$ and $a_j = g_j(r_1, \ldots, r_j, u_i, \ldots, u_j)$ for $j = 1, \ldots, n$. At the risk of a slight abuse of notation, we will denote the cost of a subsequence of σ as $\text{ALG}(r_i, \ldots, r_j)$, where the prefix is understood implicitly.

For a minimization problem, we say that an algorithm is *c-competitive*, or has a *competitive ratio* of c, if, for every finite request sequence σ, $\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + \zeta$, where ζ is not dependent on the request sequence and $\text{OPT}(\sigma)$ is the optimum cost over σ. If $\zeta = 0$, we say that an online algorithm is *strictly c-competitive*.

The *k-server problem* consists of a metric space, $\mathcal{M}$, k mobile servers and a finite request sequence, σ. Let $\mathcal{M} = (M, d)$, where M is a set of nodes, $d : M \times M \to \mathbb{R}^+$ is a distance function on M and $|M| = N > k$. Each request of σ will be to a node of $\mathcal{M}$, and a server must be moved to the requested node before the algorithm will receive the subsequent request. The goal is to minimize the distance travelled by the k servers over σ. A *lazy* k-server algorithm is an algorithm that, upon each request, only moves a single server to the request if it is uncovered.

For a metric space which is a tree, we say that a server, s, is *adjacent* to a request, r_i, if, along the shortest path between the position of s and r_i, there are no other servers.

The *caterpillar dimension* of a rooted tree, T, with root r, denoted $\mathrm{cdim}(T)$, is defined as in [8]. For a tree, T, composed of a single node, $\mathrm{cdim}(T) = 0$. For a tree, T, with two or more nodes, $\mathrm{cdim}(T) \leq k+1$ if there exists edge disjoint paths, $P_1, \ldots, P_q$, beginning at the root r such that each component T_j of $T - E(P_1) - \cdots - E(P_q)$ has $\mathrm{cdim}(T_j) \leq k$, where $E(P_i)$ are the edges of P_i. The components T_j are rooted at their unique vertex lying on some P_i. The decomposition of T into these edge disjoint paths is called the *caterpillar decomposition* of the tree. All the nodes of P_i, $1 \leq i \leq q$, except the root, are assigned path level $k+1$. The root is assigned path level $k+2$. Note that the root of the tree has a path level one more than the caterpillar dimension of the tree.

Given an unrooted tree, G, we define the caterpillar dimension of G as the minimum over all nodes, $v \in G$, of the caterpillar dimension of G when rooted at v. In what follows, we refer to the caterpillar dimension of unrooted trees as defined here.

3 An Upper Bound for General Metric Spaces

In this section, we present a $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$-competitive deterministic online algorithm with advice, called CHASE, for the k-server problem on general metric spaces, with b bits of advice per request, where $b \geq 3$. For convenience of notation, we use $\alpha = \left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$.

In order to clearly present the algorithm and proof, we will first design and analyze the algorithm such that it gets a variable number of bits of advice with each request. The algorithm will receive at least 2 bits of advice with each request, and the total number of advice bits will not exceed bn for any prefix of n requests. Afterwards, we will show how to adapt the algorithm so that it gets at most b bits of advice with each request using a transformation of [1].

Roughly speaking, our algorithm works as follows: given a request sequence, σ, we consider an optimal algorithm for this sequence. Based on this optimal algorithm, we partition σ into k subsequences, σ^s, such that all the requests of σ^s are served according to the optimal algorithm by server s. With $\log k$ bits of advice per request, we can indicate, with each request of σ^s, the identity of the server s, and, thus, our online algorithm with advice would precisely follow the optimum algorithm. If, however, we have only $b < \log k$ bits of advice per request, we do that only roughly every $\log k/b$ requests of σ^s. We call these requests "anchors". The rest of the requests of σ^s are served in a greedy manner, i.e., they are served by the closest server to the request which then returns to its previous position. By serving requests in this way, server s always stays at its last anchor. Thus, the cost of serving the $(\log k/b) - 1$ non-anchor requests between any two anchors is bounded from above by $2 \log k/b$ times the distance from the last anchor to the furthest non-anchor request. This gives us a competitive ratio of $O(\log k/b)$. Some fine tuning of the above ideas gives us our result. In what follows, we formally define the algorithm and prove its competitive ratio.

Algorithm CHASE: At the beginning, all servers are unmarked.
Given a request, r_j, and the advice, do:

- If the advice is 00, serve r_j with the closest server to r_j and return it to its previous position.
- If the advice is 10, serve r_j with the closest unmarked server and mark this server. Do not return the server to its previous position.
- If the advice is $11t$, where t is a server number encoded in $\lceil \log k \rceil$ bits, serve the request with server number t.

In order to define the advice, we will fix a optimum algorithm, OPT, that we assume to be a lazy algorithm. Henceforth, we refer to it as the lazy optimum. We will then partition the request sequence into k subsequences, $\sigma^1, \ldots, \sigma^k$, where σ^s is the trace of the server s in OPT. In other words, σ^s consists of the requests served by server s in the lazy optimum. It should be noted that the requests of σ^s are not necessarily consecutive requests in σ. Let r^s_j be the jth request served by server s over σ^s. Recall that $\alpha = \left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$. Independently, for each server, we choose an index $1 \leq q^s \leq \alpha$. The way to choose this index will be defined later. The request sequence σ^s is divided into α-length cycles starting at $r^s_{q^s+1}$. We will denote the ith cycle of σ^s by c^s_i. The first cycle, c^s_1, which starts at request r^s_1 and ends at request $r^s_{q_s}$, may have a length less than α. Let C^s be the total number of cycles in σ^s.

The advice will be defined as follows for request r^s_j:

- 10, if $j = q^s$, i.e., the last request of the first cycle.
- $11t$, if $j = q^s + i\alpha$ for some $i \geq 1$, i.e., the last request of all cycles except the first one. Here, t is the server number that serves request $r^s_{q^s}$ in CHASE encoded in $\lceil \log k \rceil$ bits.
- 00, if $j \neq q^s + i\alpha$, i.e., everywhere else.

The first two bits of the advice per request will be referred to as the control bits.

First, we state a technical lemma that we will use in our proof.

Lemma 1. *Given a sequence of α non-negative values, $a_1, \ldots, a_\alpha$, there is an integral value, q, where $1 \leq q \leq \alpha$, such that*

$$\sum_{i=1}^{q}(2(q-i)+1)a_i + \sum_{i=q+1}^{\alpha} 2(\alpha+q-i)a_i \leq \alpha \sum_{i=1}^{\alpha} a_i \ .$$

Proof. Summing the expression over all possible values of q, we get

$$\sum_{q=1}^{\alpha}\left[\sum_{i=1}^{q}(2(q-i)+1)a_i + \sum_{i=q+1}^{\alpha} 2(\alpha+q-i)a_i\right] = \left[\sum_{q=1}^{\alpha}\big(2(\alpha-q)+1\big)\right] \cdot \sum_{i=1}^{\alpha} a_i$$
$$= \alpha^2 \sum_{i=1}^{\alpha} a_i \ .$$

It follows that one of the α possible values of q gives at most the average value, i.e., $\alpha \sum_{i=1}^{\alpha} a_i$. The lemma follows.

Now, we prove the main theorem of this section.

Theorem 1. *For every $b \geq 3$, algorithm CHASE is an $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$-competitive k-server algorithm for general metric spaces with b bits of advice per request .*

Proof. For the proof, we will compare the cost of CHASE and OPT separately for every subsequence σ^s, and cycle by cycle within each σ^s. Recall that $\alpha = \left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$. Note that the first cycle and the last cycle may be of length less than α.

Consider the ith cycle of server s in OPT for $i > 1$ (we will deal with the first cycle later). Let t be the server in CHASE that serves request $r^s_{q^s}$. We will denote $r^s_{(i-2)\alpha+q^s}$, the last request of the previous cycle, by INIT^s_i. We claim that, just before the cycle starts, both OPT and CHASE will have a server at INIT^s_i. This is true because the advice for request $r^s_{(i-2)\alpha+q^s}$ indicated to CHASE to bring server t to INIT^s_i and, by the definition of the algorithm, t will always return to INIT^s_i between $r^s_{(i-2)\alpha+q^s}$ and $r^s_{(i-2)\alpha+q^s+1}$. For OPT, by definition of the subsequence σ^s, OPT serves $r^s_{(i-2)\alpha+q^s}$ with s and does not move s between request $r^s_{(i-2)\alpha+q^s}$ and request $r^s_{(i-2)\alpha+q^s+1}$.

Also, observe that just before each of the requests between $r^s_{(i-2)\alpha+q^s+1}$ and $r^s_{(i-1)\alpha+q^s}$ inclusive, i.e., the requests of the ith cycle, server t of CHASE is at INIT^s_i. Recall that CHASE serves these requests except the last one by using the closest server and, then, returns it to its prior position. Therefore, the cost to CHASE for any request $r^s_{(i-2)\alpha+q^s+j}$, where $1 \leq j \leq \alpha - 1$, i.e., the requests of cycle i except the last one, is

$$\text{CHASE}(r^s_{(i-2)\alpha+q^s+j}) \leq 2d(\text{INIT}^s_i, r^s_{(i-2)\alpha+q^s+j}) \ . \tag{1}$$

By the triangle inequality and Equation (1),

$$\text{CHASE}(r^s_{(i-2)\alpha+q^s+j}) \leq 2\sum_{l=1}^{j} d(r^s_{(i-2)\alpha+q^s+l-1}, r^s_{(i-2)\alpha+q^s+l}) \ . \tag{2}$$

For request $r^s_{(i-1)\alpha+q^s}$, i.e., the last request of cycle i, CHASE serves the request using server t that is at $r^s_{(i-2)\alpha+q^s}$. We have, by the triangle inequality,

$$\begin{aligned} \text{CHASE}(r^s_{(i-1)\alpha+q^s}) &= d(\text{INIT}^s_i, r^s_{(i-1)\alpha+q^s}) \\ &\leq \sum_{l=1}^{\alpha} d(r^s_{(i-2)\alpha+q^s+l-1}, r^s_{(i-2)\alpha+q^s+l}) \ . \end{aligned} \tag{3}$$

Observe that the cost of OPT to serve $r^s_{(i-2)\alpha+q^s+j}$ for $1 \leq j \leq \alpha$, i.e., the requests of cycle i, is $d(r^s_{(i-2)\alpha+q^s+j-1}, r^s_{(i-2)\alpha+q^s+j})$. Using this fact and

Equations (2) and (3), we can bound the cost of CHASE over a cycle by the cost of OPT as follows:

$$\sum_{j=1}^{\alpha} \mathrm{CHASE}(r^s_{(i-2)\alpha+q^s+j}) \leq \sum_{j=1}^{\alpha-1} \left(2 \sum_{l=1}^{j} \mathrm{OPT}(r^s_{(i-2)\alpha+q^s+l}) \right) + \sum_{l=1}^{\alpha} \mathrm{OPT}(r^s_{(i-2)\alpha+q^s+l}) = \sum_{j=1}^{\alpha} [2(\alpha - j) + 1] \mathrm{OPT}(r^s_{(i-2)\alpha+q^s+j}) \ . \quad (4)$$

The analysis of the first cycle is, essentially, the same as the analysis of the ith cycle, $i > 1$, with the exception that an additive constant is introduced per request of the first cycle. The additive constant results from the fact that, during the first cycle of σ^s, CHASE does not necessarily maintain a server at the initial position of s. Nevertheless, by the definition of CHASE, there will always be an unmarked server in one of the locations of the initial configuration. Let Δ be the diameter of the initial configuration. Therefore, for any request of the first cycle, r^s_l, of σ^s, analogously to Equation (2), we have

$$\mathrm{CHASE}(r^s_l) \leq 2\Big(\Delta + \sum_{m=1}^{l} d(r^s_{m-1}, r^s_m) \Big) \ , \quad (5)$$

where r^s_0 is the initial position of s. Analogous to Equation (4), summing Equation (5) over all requests of the first cycle of s, gives

$$\sum_{l=1}^{q^s} \mathrm{CHASE}(r^s_l) \leq \sum_{l=1}^{q^s} [2(q^s - l) + 1] \mathrm{OPT}(r^s_l) + 2\alpha\Delta \ . \quad (6)$$

If we assume the cost for requests with indexes less than 0 to be 0 for both OPT and CHASE, we can rewrite Equation (6) to be more congruent with Equation (4) as follows:

$$\sum_{j=1}^{\alpha} \mathrm{CHASE}(r^s_{-\alpha+q^s+j}) \leq \sum_{j=1}^{\alpha} [2(\alpha - j) + 1] \mathrm{OPT}(r^s_{-\alpha+q^s+j}) + 2\alpha\Delta \ . \quad (7)$$

Using Equations (4) and (7), and summing over all cycles, gives

$$\mathrm{CHASE}(\sigma^s) \leq \sum_{i=1}^{C^s} \sum_{j=1}^{\alpha} [2(\alpha - j) + 1] \mathrm{OPT}(r^s_{(i-2)\alpha+q^s+j}) + 2\alpha\Delta \ . \quad (8)$$

Define $a_1, \ldots, a_\alpha$ such that $a_j = \sum_{i=1}^{C^s} \mathrm{OPT}(r^s_{(i-1)\alpha+j})$, i.e., the cost of OPT for the requests in σ^s in jumps of α requests. We can rewrite Equation (8) as

$$\mathrm{CHASE}(\sigma^s) \leq \sum_{i=1}^{q} (2(q - i) + 1) a_i + \sum_{i=q+1}^{\alpha} 2(\alpha + q - i) a_i + 2\alpha\Delta \ . \quad (9)$$

By Lemma 1, there is a value $1 \leq q^s \leq \alpha$ such that

$$\text{CHASE}(\sigma^s) \leq \alpha \sum_{i=1}^{\alpha} a_i + 2\alpha\Delta = \alpha\text{OPT}(\sigma^s) + 2\alpha\Delta \ .$$

We chose this q^s separately for each server s in order to define the cycles. Summing over all k subsequences σ^s concludes the proof of the competitive ratio.

Finally, we show that the algorithm uses at most bn bits over any prefix of n requests. There are 2 control bits with each request. Let t be the server in CHASE that serves $r^s_{q_s}$, i.e., the last request of the first cycle of σ^s. There are at least α requests of σ^s between any two requests, where the id of t is given in the advice. Since $\alpha = \left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$, the claim follows.

In order to adapt the algorithm so that it receives b bits of advice per request, we use a transformation of [1]. Two control bits will be given with each request, and the remaining $b-2$ bits will contain portions of server ids. The control bits will be as defined previously. We then define a string as the concatenation of all server ids given for the whole sequence. This string will be broken into $(b-2)$-bit chunks and a single chunk will be given with each request. The algorithm can store these $(b-2)$-bit chunks in a FIFO queue and will have $\lceil \log k \rceil$ bits available to be read from the queue when dictated by the control bits.

4 *k*-Server with Advice on Trees

In this section, we describe a deterministic online algorithm with advice for the k-server problem on finite trees, called PATH-COVER, that is 1-competitive and uses $2 + 2\lceil \log(p+1) \rceil$ bits of advice per request, where p denotes the minimal caterpillar dimension of the tree. Similar results can be obtained if other measures of the tree, such as its height, are used instead of the caterpillar dimension. We chose this measure since it gives a 1-competitive algorithm with a constant number of bits of advice per request for degenerate trees such as the line or a caterpillar. Furthermore, the caterpillar dimension is at most the height of the tree, and is at most $\log N$, where N is the number of nodes in the tree [8].

The algorithm and advice are based on the actions of a non-lazy optimum algorithm with certain properties for the given sequence. First, we describe this non-lazy algorithm and show that it has optimum cost.

4.1 Non-lazy Optimum

We show that, for every sequence of requests, there is an algorithm, OPT_{nl}, with optimal cost that, also, has the following three properties given that the algorithm can chose its initial configuration.

1. Between r_i and just before r_{i+1}, OPT_{nl} moves at most a single server, s. Note that s may make multiple moves.

2. Just before r_i, s is at the same path level or higher than r_i. The path level is according to the caterpillar decomposition of the tree.
3. s is adjacent to r_i just before r_i.

Given the caterpillar decomposition of T that minimizes $\mathrm{cdim}(T)$ and a lazy optimum, OPT_l, we first build OPT'_{nl} which has the first two properties above. For $u, v \in T$, let $\mathrm{maxPath}(u, v)$ be the node nearest u on the highest path level on the path between u and v. We choose the initial configuration of OPT'_{nl} as follows: for each of the k servers s_i, place s_i at $\mathrm{maxPath}(u_i, v_i)$, where u_i is the initial position of s_i, and v_i is the position of the first request served by s_i in OPT_l. Then, each request, r_j, is served in OPT'_{nl} with the same server as OPT_l. After serving r_j, place the server, t, used for r_j at $\mathrm{maxPath}(r_j, r_q)$, where r_q is the next request served by t in OPT_l. Observe that the first two properties above hold for OPT'_{nl}.

Claim. For any σ, $\mathrm{OPT}'_{nl}(\sigma) = \mathrm{OPT}_l(\sigma)$.

Proof. The claim follows from the fact that the trajectories followed by each of the servers according to OPT'_{nl} and OPT_l are the same. The only difference being that some of the moves are done earlier in OPT'_{nl} than in OPT_l.

Now, we construct OPT_{nl} based on OPT'_{nl} to satisfy property 3 along with the first two properties without increasing the cost.

OPT_{nl} will be defined by induction on the request sequence. Let $T^* = (t^*_1, q_1), \ldots, (t^*_m, q_m)$ be all the server moves, in order, performed by OPT'_{nl} such that the ordered pair (x, y) defines a move of a server from position x to position y, and $m \geq n$ is the total number of server moves performed by OPT_{nl}. Let $S^* = (s^*_1, r_1), \ldots, (s^*_n, r_n)$ be the subsequence of T^*, where the ith ordered pair indicates that the server at position s^*_i serves r_i in OPT'_{nl}. We build T^j and S^j, $j \geq 0$, inductively, where $T^0 = T^*$ and $S^0 = S^*$. $T^j = (t^j_1, q_1), \ldots, (t^j_m, q_m)$ will have all three properties above until after serving request r_j (in fact Property 1 will hold for the entire sequence T^j), and $S^j = (s^j_1, r_1), \ldots, (s^j_n, r_n)$ will have the property that $s^j_1, \ldots, s^j_j$ are adjacent to their respective requests.

Assume that S^{i-1} and T^{i-1} are well defined. This is trivially true for T^0 and S^0. In order to construct S^i and T^i, we need to consider s^{i-1}_i which is either adjacent or not to r_i. If s^{i-1}_i is adjacent to r_i, then $S^i := S^{i-1}$ and $T^i := T^{i-1}$. Otherwise, if s^{i-1}_i is not adjacent to r_i, there exists a server at node x on the path between s^{i-1}_i and r_i which is adjacent to r_i. In this case, the relevant moves in both T and S will be modified such that the server at x serves request r_i, and the server at s^{i-1}_i will be used the very next time that the server at x would have been used. To formally define T^i and S^i we proceed as follows: S^i is defined as S^{i-1} from (s^i_1, r_1) to (s^i_{i-1}, r_{i-1}), and T^i is defined as T^{i-1} from (t^{i-1}_1, q_1) to (t^{i-1}_j, q_{j-1}), where q_j is r_i. The next moves in S^i and T^i are defined as $(s^i_i, r_i) := (t^i_j, q_j) := (x, r_i)$. From (t^i_{j+1}, q_{j+1}) to (t^i_m, q_m), T^i is defined as T^{i-1} except for the next occurrence of (x, q_l) in T^{i-1}, $l < m$. Set $(t^i_l, q_l) := (s^{i-1}_i, q_l)$ in T^i. If q_l is a request, say r_p, then we, also, set $(s^i_p, r_p) := (s^{i-1}_i, r_p)$ for S^i. The rest of S^i is defined as S^{i-1}.

Lemma 2. *On the tree, the cost of OPT_{nl} is no more than the cost of OPT'_{nl}, and OPT_{nl} has the following three properties:*

1. *Between r_i and just before r_{i+1}, OPT_{nl} moves at most a single server, s.*
2. *Just before r_i, s is at the same path level or higher than r_i.*
3. *s is adjacent to r_i just before r_i.*

Proof. First, we show that the cost of OPT_{nl} is no more than the cost of OPT'_{nl}. This is done by induction on the construction steps of T^i (and S^i). For $i = 0$ the claim is trivial. For the inductive step, we note that we change at most two moves between T^{i-1} and T^i as in the construction above. With the notations of the construction above, we note that the cost of T^i for q_i and q_l is at most $d(s_i^i, q_j) + d(s_i^{i-1}, s_i^i) + d(s_i^{i-1}, q_l) = d(s_i^{i-1}, q_j) + d(s_i^i, q_l)$ which is the cost paid by T^{i-1} for q_i and q_l.

Property 1 holds for OPT_{nl} because it holds for OPT'_{nl}, no moves are added in the construction, and the only changes are to the first move after a request is issued.

We now show by induction on i that property 2 holds for S^i. This is true for S^0 since property 2 holds for OPT'_{nl}. For the induction step, let s' be the server used by S^{i-1} to serve r_i, and let s be the server used by S^i to serve r_i . Let $\ell(v)$ denote the path level of a node v. We know by the induction hypothesis that $\ell(s') \geq \ell(r_i)$. Also, s lies on the path between s' and r_i, and, by the recursive nature of the caterpillar decomposition, it follows that $\ell(s) \geq \ell(r_i)$. If q_l is not a request, there is nothing else to prove. If q_l is a request, by the induction hypothesis, we know that $\ell(s) \geq \ell(q_l)$, and we have that $\ell(s') \geq \ell(s)$. Therefore, $\ell(s') \geq \ell(q_l)$. Thus, property 2 holds for S^i.

Property 3 is immediate from the inductive construction.

4.2 The Algorithm

There will be two stages to the algorithm. The initial stage will be for the first k requests and will be used to match the configuration of PATH-COVER to that of OPT_{nl} as defined in the previous section. Over the remaining requests, PATH-COVER will be designed to act exactly as OPT_{nl}. PATH-COVER will receive $2(l + 1)$ bits of advice per request, where $l = \lceil \log(p + 1) \rceil$ and p is the minimal caterpillar dimension of the tree. The advice will be of the form $wxyz$, where w and x will be 1 bit in length, and y and z will be l bits in length.

Algorithm and Advice for $r_1, \ldots, r_k$. From r_1 to r_k, PATH-COVER will serve each request with the nearest server regardless of the advice. As for the advice, for request r_i, where $1 \leq i \leq k$,

- if $w = 1$, the algorithm stores the node nearest r_i which has the path of level y.
- if $x = 1$, the algorithm stores the node nearest the initial position of the ith server which has the path of level z.

Note that both w and x can be 1 for request r_i. Immediately after serving r_k, PATH-COVER will use the first k stored nodes as a server configuration and will move to this configuration at minimal cost (minimum matching).

For $1 \leq i \leq k$, the advice for r_i will be defined as follows:

- w: 1, if the server used for r_i in OPT_{nl} does not serve another request up to r_k.
  0, otherwise
- x: 1, if the ith server does not serve any of the first k requests in OPT_{nl}.
  0, otherwise
- y: A number in binary indicating the path level to which the server used for r_i is moved to in OPT_{nl} after serving r_i.
- z: A number in binary indicating the path level to which the ith server is moved to before r_1 in OPT_{nl}.

Over the first k requests, w and x will be 1 a total of k times (once for each of the k servers of OPT_{nl}). For r_i, when w is 1, this means that, immediately after r_k, OPT_{nl} will have a server at the path of level y between r_i and the root. When x is 1, this means that, immediately after r_k, the ith server of OPT_{nl} will be at the path of level z between the initial position of the ith server and the root. This is the server configuration of OPT_{nl} immediately after r_k encoded in the bits of advice over the first k requests.

Algorithm and Advice for $r_{k+1}, \ldots, r_n$. From r_{k+1} to r_n, given a request, r_i, where $k+1 \leq i \leq n$, and the advice, let P be the unique path of level y between r_i and the root. Now, define a path, Q, on the tree as follows:

- if $x = 1$, Q runs from r_i to the end of P nearest the root.
- if $x = 0$, Q runs from r_i to the end of P furthest from the root.

PATH-COVER will serve r_i with the closest server along Q. After serving r_i, PATH-COVER will move this server to the node of the path of level z nearest to r_i.

For $k+1 \leq i \leq n$, the advice for r_i will be defined as follows:

- w: 0 (not used)
- x: Let s be the server used by OPT_{nl} to serve r_i, and let c be the node of the same path level as the location of s closest to r_i.
  1, if s is between c and the root of the tree.
  0, otherwise.
- y: A number in binary indicating the path level to which of the server that OPT_{nl} uses for r_i.
- z: A number in binary indicating the path level to which the server used for r_i is moved to in OPT_{nl} immediately after serving r_i.

Analysis

Theorem 2. *PATH-COVER is* 1*-competitive on finite trees.*

Proof. From r_1 to r_k, all the requests are served by the closest server. This cost can be bounded by $k\Delta$, where Δ is the diameter of the tree. Immediately after r_k, PATH-COVER matches the configuration of OPT_{nl}. The cost to match a configuration can be bounded by $k\Delta$.

According to the definition of the advice and the algorithm, the configuration of PATH-COVER matches the configuration of OPT_{nl} after serving r_k. We show that starting at request r_{k+1}, the configurations of PATH-COVER and OPT_{nl} match just before serving each request, and that PATH-COVER serves the request with a server from the same position as does OPT_{nl}. Assume that the configurations of PATH-COVER and OPT_{nl} match until immediately before serving r_i for some $k+1 \leq i \leq n$. Let s be the server used by OPT_{nl} for r_i. We claim that PATH-COVER and OPT_{nl} will use a server from the same position for r_i. The induction assumption that the configurations of PATH-COVER and OPT_{nl} match immediately before serving r_i and the third property of OPT_{nl} guarantee that there is no server between the position of s and r_i in PATH-COVER. The second property of OPT_{nl} guarantees that s is at the same path level or higher than r_i. This implies that the advice provided to PATH-COVER specifies a unique server that must be at the same position in PATH-COVER as s in OPT_{nl}. Immediately after serving r_i, PATH-COVER and OPT_{nl} move the server used for r_i to the same node in the tree as per their definitions. So, immediately before serving r_{i+1}, the configurations of OPT_{nl} and PATH-COVER are the same.

Therefore,

$$\text{PATH-COVER}(\sigma) \leq \mathrm{OPT}(\sigma) + 2k\Delta$$

4.3 Special Metric Spaces and PATH-COVER

This section presents some variations and implication of the previously described algorithm, PATH-COVER, for special metric spaces. More detailed proofs of each case can be found in [9].

The Line. The caterpillar dimension for the line is 1 which implies that our algorithm PATH-COVER requires 4 bits of advice. However, as the servers essentially do not change path levels on the line, a single bit of advice indicating the direction of the server to be used is all that is needed for a strictly 1-competitive algorithm.

The Circle. Applying the algorithm for the line to the circle provides a strictly 1-competitive algorithm with 1 bit of advice. The key is to define a clockwise orientation on the cycle. The 1 bit of advice will indicate whether to use the adjacent server in the clockwise direction or the counter-clockwise direction.

The Spider. A spider graph consists of a single fork, the centre, and 0 or more branches without forks connected to the centre. The caterpillar dimension is 1

implying 4 bits of advice for PATH-COVER. We can define a variant of PATH-COVER for the spider that uses 2 bits of advice. One bit indicates the direction of the adjacent server, s, for the request while the second bit indicates if s should be moved to the centre after serving the request. This algorithm is 1-competitive.

5 Conclusions

We give an improved upper bound for the k-server problem with advice on general metric spaces. Moreover, we believe that our algorithm and our analysis are more intuitive and simpler than previous ones, and may, thus, lead to further improvements in the upper bound. We, also, give a 1-competitive k-server algorithm with advice for finite trees, using a number of bits of advice which is a function of the caterpillar dimension of the tree. The obvious open problem that remains is to give tight bounds for the k-server problem with advice on general metric spaces or for specific metric spaces.

Acknowledgements. We thank Yuval Emek, Pierre Fraigniaud, Amos Korman, and Manor Mendel for useful discussions.

References

1. Böckenhauer, H.-J., Komm, D., Královic, R., Královic, R.: On the Advice Complexity of the k-Server Problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011) see also technical report at, ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/7xx/703.pdf
2. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the Advice Complexity of Online Problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
3. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)
4. Dobrev, S., Královič, R., Pardubská, D.: How Much Information About the Future is Needed? In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 247–258. Springer, Heidelberg (2008)
5. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theor. Comput. Sci. 412(24), 2642–2656 (2011)
6. Hromkovič, J., Královič, R., Královič, R.: Information Complexity of Online Problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
7. Komm, D., Královič, R.: Advice Complexity and Barely Random Algorithms. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 332–343. Springer, Heidelberg (2011)
8. Matoušek, J.: On embedding trees into uniformly convex banach spaces. Israel Journal of Mathematics 114, 221–237 (1999), http://dx.doi.org/10.1007/BF02785579, doi:10.1007/BF02785579
9. Renault, M.: Online Algorithms with Advice. Master's thesis, MPRI – Université Paris Diderot - Paris 7 (2010)

Improved Lower Bound for Online Strip Packing
(Extended Abstract)

Rolf Harren[1] and Walter Kern[2]

[1] Max-Planck-Institut für Informatik (MPII)
Campus E1 4, 66123 Saarbrücken, Germany
rharren@mpi-inf.mpg.de
[2] University of Twente, Department of Applied Mathematics
P.O. Box 217, 7500 AE Enschede, The Netherlands
w.kern@utwente.nl

1 Introduction

In the two-dimensional strip packing problem a number of rectangles have to be packed without rotation or overlap into a strip such that the height of the strip used is minimal. The width of the rectangles is bounded by 1 and the strip has width 1 and infinite height.

We study the online version of this packing problem. In the online version the rectangles are given to the online algorithm one by one from a list, and the next rectangle is given as soon as the current rectangle is irrevocably placed into the strip. To evaluate the performance of an online algorithm we employ competitive analysis. For a list of rectangles L, the height of a strip used by online algorithm ALG and by the optimal solution is denoted by $\mathrm{ALG}(L)$ and $\mathrm{OPT}(L)$, respectively. The optimal solution is not restricted in any way by the ordering of the rectangles in the list. Competitive analysis measures the absolute worst-case performance of online algorithm ALG by its competitive ratio

$$\rho_{\mathrm{ALG}} = \sup_L \left\{ \frac{\mathrm{ALG}(L)}{\mathrm{OPT}(L)} \right\}.$$

Known Results. Regarding the upper bound on the competitive ratio for online strip packing, recent advances have been made by Ye, Han & Zhang[6] and Hurink & Paulus[3]. Independently they showed that a modification of the well-known shelf algorithm yields an online algorithm with competitive ratio $7/2 + \sqrt{10} \approx 6.6623$. We refer to these two papers for a more extensive overview of the literature.

In the early 80s, Brown, Baker & Katseff[1] derived a lower bound $\rho \geq 2$ on the competitive ratio of any online algorithm by constructing certain (adversary) sequences in a fairly straightforward way. These sequences, that we call BBK sequences in the sequel, were further studied by Johannes[4] and Hurink & Paulus[2], who derived improved lower bounds of 2.25 and 2.43, respectively. (Both results are computer aided and presented in terms of online parallel machine scheduling, a closely related problem.) The paper of Hurink & Paulus[2] also presents an upper bound of $\rho \leq 2.5$ for packing BBK sequences. Kern & Paulus[5] finally settled the question how well the BBK sequences can be packed by providing a matching upper and lower bound of $\rho_{\mathrm{BBK}} = 3/2 + \sqrt{33}/6 \approx 2.457$.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 211–218, 2012.

Our Contribution. Using modified BBK sequences we show an improved lower bound of $2.589\ldots$ on the absolute competitive ratio of this problem. The modified sequences that we use consist solely of two types of items, namely, *thin* items that have negligible width (and thus can all be packed in parallel) and *blocking* items that have width 1. The advantage of these sequences is that the structure of the optimal packing is simple, i.e., the optimal packing height is the sum of the heights of the blocking items plus the maximal height of the thin items. Therefore, we call such sequences *primitive*.

On the positive side, we present an online algorithm for packing primitive sequences with competitive ratio $(3+\sqrt{5})/2 = 2.618\ldots$. This upper bound is especially interesting as it not only applies to the concrete adversary instances that we use to show our lower bound. Thus to show a new lower bound for strip packing that is greater than $2.618\ldots$ (and thus reduce the gap to the general upper bound of 6.6623), new techniques are required that take instances with more complex optimal solutions into consideration.

Organization. We start our presentation with a description of the Brown-Baker-Katseff sequences and their modification. Afterwards we present our lower bound based on these modifications, and finally we describe our algorithm for packing primitive sequences.

2 Sequence Construction

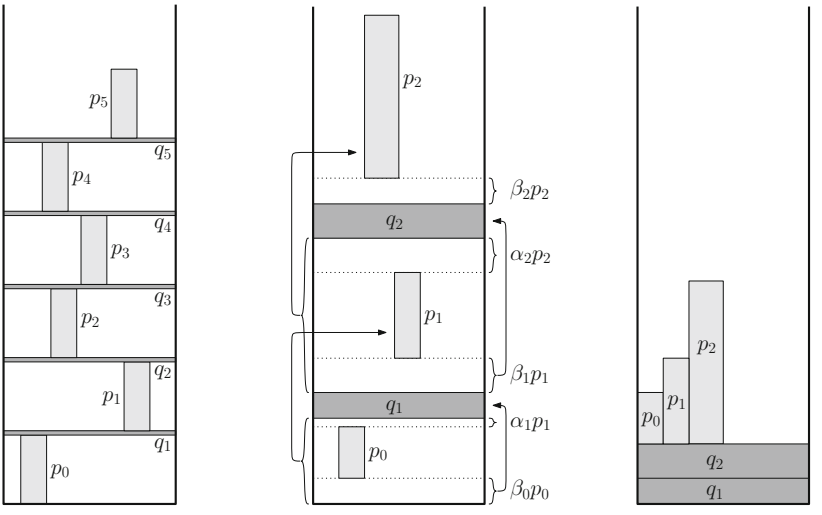
In this paper we denote the thin items by p_i and the blocking items by q_i (adopting the notation from [5]). As already mentioned in the introduction, we assume that the width of the thin items is negligible and thus all thin items can be packed next to each other. Moreover, the width of the blocking items q_i is always 1, so that no item can be packed next to any blocking item in parallel. Therefore, all items are characterized by their heights and we refer to their heights by p_i and q_i as well. By definition, for any list $L = q_1, q_2, \ldots, q_k, p_1, p_1, \ldots, p_\ell$ consisting of thin and blocking items we have

$$\mathrm{OPT}(L) = \sum_{i=1}^{k} q_i + \max_{i=1,\ldots,\ell} p_i.$$

To prove the desired lower bound we assume the existence of a ρ-competitive algorithm ALG for some $\rho < 2.589\ldots$ (the exact value of this bound is specified later) and construct an adversary sequence depending on the packing that ALG generates.

To motivate the construction, let us first consider the GREEDY algorithm for online strip packing, which packs every item as low as possible—see Figure 1a. This algorithm is not competitive (i.e., has unbounded competitive ratio): Indeed, consider the list $L_n = p_0, q_1, p_1, q_2, p_2, \ldots, q_n, p_n$ of items with

$$\begin{aligned} p_0 &:= 1, \\ q_i &:= \varepsilon && \text{for } 1 \le i \le n, \\ p_i &:= p_{i-1} + \varepsilon && \text{for } 1 \le i \le n \end{aligned}$$



(a) A greedy packing (b) A packing of a BBK sequence (c) An optimal packing of a BBK sequence

Fig. 1. Online and optimal packings

for some $\varepsilon > 0$. GREEDY would pack each item on top of the preceding ones and thus generate a packing of height $\text{GREEDY}(L_n) = \sum_{i=0}^{n} p_i + \sum_{i=1}^{n} q_i = n + 1 + \Omega(n^2\varepsilon)$, whereas the optimum clearly has height $1 + 2n\varepsilon$.

The GREEDY algorithm illustrates that any competitive online algorithm needs to create gaps in the packing. These gaps work as a buffer to accommodate small blocking items—or, viewed another way, force the adversary to release larger blocking items.

BBK sequences. The idea of Brown, Baker & Katseff[1] was to try to cheat an arbitrary (non-greedy) online packing algorithm ALG in a similar way by constructing an alternating sequence $p_0, q_1, p_1, \ldots$ of thin and blocking items. The heights p_i respectively q_i are determined so as to force the online algorithm ALG to put each item above the previous ones—see Figure 1b for an illustration. To describe the heights of the items formally, we consider the gaps that ALG creates between the items. We distinguish two types of gaps, namely gaps below and gaps above a blocking item, and refer to theses gaps as α- and β-gaps, respectively. These gaps also play an important role in our analysis of the modified BBK sequences. We describe the height of the gaps around the blocking item q_i relative to the thin item p_i. Thus, we denote the height of the α-gap below q_i by $\alpha_i p_i$ and the height of the β-gap above q_i by $\beta_i p_i$. Using this notation, we are ready to formally describe the BBK sequences $L = p_0, q_1, p_1, q_2, \ldots$ with

$$
\begin{aligned}
p_0 &:= 1, \\
q_1 &:= \beta_0 p_0 + \varepsilon, \\
p_i &:= \beta_{i-1} p_{i-1} + p_{i-1} + \alpha_i p_i + \varepsilon && \text{for } i \geq 1, \\
q_i &:= \max\left(\alpha_{i-1} p_{i-1}, \beta_{i-1} p_{i-1}, q_{i-1}\right) + \varepsilon && \text{for } i \geq 2.
\end{aligned}
$$

As mentioned in the introduction, Brown, Baker & Katseff[1] used these sequences to derive a lower bound of 2 before Kern & Paulus[5] recently showed that the competitive ratio for packing them is $\rho_{\mathrm{BBK}} = 3/2 + \sqrt{33}/6 \approx 2.457$.

The optimal online algorithm for BBK sequences that Kern & Paulus[5] describe generates packings with striking properties: No gaps are created except the first possible gap $\beta_0 = \rho_{\mathrm{BBK}} - 1$ and the second α-gap $\alpha_2 = 1/(\rho_{\mathrm{BBK}} - 1)$, which are chosen as large as possible while remaining ρ_{BKK}-competitive. Observing this behavior of the optimal algorithm led us to the modification of the BBK sequences.

Modified BBK sequences. When packing BBK sequences, a good online algorithm should be eager to enforce blocking items of relatively large size (as each blocking item of size q increases the optimal packing by q as well). These blocking items are enforced by generating corresponding gaps.

Modified BBK sequences are designed to counter this strategy: Each time the online algorithm places a blocking item q_i, the adversary, rather than immediately releasing a thin item p_{i+1} (of height defined as in standard BBK sequences) that does not fit in between the last two blocking items, generates a whole sequence of slowly growing thin items, which "continuously" grow from p_i to p_{i+1}. Packing this subsequence causes additional problems for the online algorithm: If the algorithm fits the whole subsequence into the last interval between q_{i-1} and q_i, it would fill out the whole interval and create an α-gap of 0. On the other extreme, if ALG would pack a thin item of height roughly p_i above q_i, then the (relative) β-gap it can generate is much less compared to what it could have achieved with a thin item of larger height p_{i+1}. The next blocking item q_{i+1} will be released as soon as the sequence of thin items has grown from p_i to p_{i+1}.

This general concept of modified BBK sequences applies after the first blocking item q_1 is released. Since subsequences of thin items and single blocking items are released alternately, we refer to this phase as the *alternating* phase. Before that, we have a *starting* phase which ends with the release of the first blocking item q_1. This starting phase needs special attention as we have no preceding interval height as a reference.

The optimal online algorithm by Kern & Paulus[5] generates an initial gap $\beta_0 = \rho_{\mathrm{BBK}} - 1$ of maximal size to enforce a large first blocking item q_1. In the starting phase, we seek to prevent the algorithm from creating a large β_0-gap in the following way. Assume that the online algorithm places p_0 "too high" (i.e., β_0 is "too large"). Then the adversary, instead of releasing q_1, would continue generating higher and higher thin items and observe how the algorithm places them. As long as the algorithm places these thin items next to each other (overlapping in their packing height), the size of the gap below these items decreases monotonically relative to the height where items are packed. Eventually, β_0 has become sufficiently small—in which case the starting phase comes to an end with the release of q_1—or the online algorithm decides to "jump" in the sense that one of the items in this sequence of increasing height thin items is put strictly above all previously packed thin items, creating a new gap (distance between the last two items) and a significantly increased new packing height. Once a jump has occurred, the adversary continues generating thin items of slowly growing height until a next jump occurs or until the ratio of the largest current gap to the current packing height (the modified analogue to the standard β_0-gap) is sufficiently small and the starting phase comes to an end.

Summarizing, a *modified BBK sequence* simply consists of a sequence of thin items, continuously growing in height, interleaved with blocking items which (by definition of their height) must be packed above all preceding items, and are released as described above, *i.e.*, when the thin item size has grown up to the largest gap between two blocking items, *c.f.* the full paper for more details.

In the next section we use these modified BBK sequences to show the following theorem.

Theorem 1. *There exists no algorithm for online strip packing with competitive ratio*

$$\rho < \hat{\rho} = \frac{17}{12} + \frac{1}{48}\sqrt[3]{22\,976 - 768\sqrt{78}} + \frac{1}{12}\sqrt[3]{359 + 12\sqrt{78}} \approx 2.589\ldots.$$

3 Lower Bound

For the sake of contradiction, we assume that ALG is a ρ-competitive algorithm for online strip packing with $\rho < \hat{\rho}$. Let $\delta = \hat{\rho} - \rho > 0$. W.l.o.g. we assume that δ is sufficiently small.

We distinguish between the thin items p_i (whose height matches the height of the previous interval plus an arbitrarily small excess) and the subsequences of gradually growing thin items by denoting the whole sequence of thin items by $r_1, r_2, \ldots$ and designating certain thin items as p_i.

Our analysis (cf section 5) distinguishes two phases. In the first phase, the *starting* phase, we consider the following problem that the online algorithm faces. Given an input that consists only of thin items $r_1, r_2, \ldots$ (in this phase no blocking items are released), minimize the competitive ratio while retaining a free gap of maximal size (relative to the current packing height). More specifically, let

$$\frac{h(\mathrm{maxgap}_{\mathrm{ALG}}(r_i))}{\mathrm{ALG}(r_i)}$$

be the *max-gap-to-height* ratio after packing r_i where $h(\mathrm{maxgap}_{\mathrm{ALG}}(r_i))$ denotes the height of the maximal gap that algorithm ALG created up to item r_i and $\mathrm{ALG}(r_i)$ denotes the height algorithm ALG consumed up to item r_i. We say ALG is (ρ, c)-competitive in the starting phase if ALG is ρ-competitive (i.e., $\mathrm{ALG}(r_i) \leq \rho\mathrm{OPT}(r_i)$) and retains a max-gap-to-height ratio of c (i.e., $h(\mathrm{maxgap}_{\mathrm{ALG}}(r_i))/\mathrm{ALG}(r_i) \geq c$ for $i \geq 1$) for all lists $L = r_1, r_2, \ldots$ of thin items.

In the analysis of the starting phase we show that our modified BBK sequences force any ρ-competitive algorithm to reach a state with max-gap-to-height ratio less than

$$\hat{c} = \frac{\hat{\rho} - 2\sqrt{\hat{\rho} - 1}}{\hat{\rho} - 1}.$$

Thus no $(\rho, \hat{c})$-competitive algorithm exists for $\rho < \hat{\rho}$. In the moment ALG packs an item r_i and hereby reaches a max-gap-to-height ratio of less than $\hat{c}$, the starting phase ends with the release of the first blocking item q_1 of height $\hat{c} \cdot \mathrm{ALG}(r_i)$.

In the analysis of the *alternating phase* we show that no ρ-competitive algorithm can exist if the first blocking item after the starting phase has height $\hat{c}$ times the current packing height for

$$\hat{c} = \frac{1 - \sqrt{4\hat{\rho}^2 - 12\hat{\rho} + 5}}{2(\hat{\rho} - 1)}.$$

Thus our two phases fit together for

$$\hat{c} = \frac{\hat{\rho} - 2\sqrt{\hat{\rho} - 1}}{\hat{\rho} - 1} = \frac{1 - \sqrt{4\hat{\rho}^2 - 12\hat{\rho} + 5}}{2(\hat{\rho} - 1)},$$

which is satisfied for

$$\hat{\rho} = \frac{17}{12} + \frac{1}{48}\sqrt[3]{22\,976 - 768\sqrt{78}} + \frac{1}{12}\sqrt[3]{359 + 12\sqrt{78}} \approx 2.589\ldots.$$

The corresponding value of $\hat{c}$ is $\hat{c} \approx 0.04275\ldots$. We skip the proof of Theorem 1.

Algorithm 1. Online Algorithm for Restricted Instances

```
Initially the packing is considered to be blocked
whenever a rectangle r_j is released do
    if r_j is a blocking item then
        Pack r_j at the lowest possible height
    else if r_j is a thin item then
        if the packing is open then
            Pack r_j bottom-aligned with the top thin item
        else if the packing is blocked then
            Try to pack r_j below the top item
            If this is not possible, pack r_j at distance (ρ − 2)r_j above the packing
```

4 Upper Bound

In this section we present the online algorithm ONL for packing instances that consist solely of thin and blocking items. We prove that the competitive ratio of ONL is $\rho = (3 + \sqrt{5})/2$. We distinguish two kinds of packings according to the item on top: If the item on top of the packing is a blocking item, we have a *blocked* packing, otherwise we have an *open* packing. Initially, we have a blocked packing by considering the bottom of the strip as a blocking item of height 0.

The general idea of the algorithm ONL is pretty straight-forward: Generate a β-gap of relative height $\rho - 2$ whenever a jump is unavoidable and pack arriving blocking items as low as possible. Since we neglect the starting phase, $\beta = \rho - 2$ is the maximal β-gap that we can ensure. This leads to the following algorithm—see also Algorithm 1.

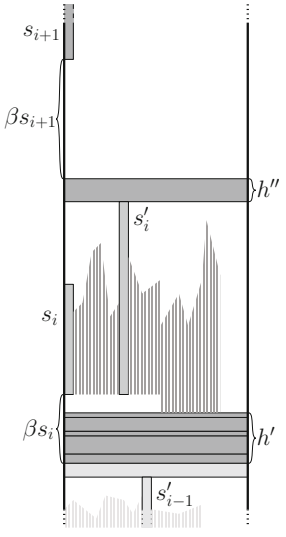


Fig. 2. Packing after the $(i+1)$-th jump. The blocking items that arrived after s_i are shown in darker shade. By definition, s_i is the first item that does not fit into the previous interval. Thus we have $s_{i+1} > s'_i + \beta\, s_i - h'$.

- If a blocking item r_j arrives, we pack r_j at the lowest possible height. This can be inside the packing, if a sufficiently large gap is available, or directly on top of the packing. In the latter case, the packing is blocked afterwards.
- If a thin item r_j arrives at an open packing, we bottom-align r_j with the top item.
- If, finally, a thin item r_j arrives at a blocked packing, we try to pack r_j below the blocking item on top. If this is not possible, i.e., r_j exceeds the height of all intervals for thin items, we pack r_j at distance $\beta\, r_j = (\rho - 2) r_j$ above the top of the packing. This changes the packing to an open packing again.

We show that ONL is ρ-competitive for $\rho = (3 + \sqrt{5})/2$. Actually, this is only questionable in one case, namely, when we pack a thin item r_j with distance $(\rho - 2) r_j$ above the packing. All other cases are trivial since if the packing height increases, then the optimal height increases by the same value (for thin items the packing height only increases if r_j is the new maximal item).

We denote the thin items that are packed when generating a new gap by s_i for the i-th jump. Let s'_{i-1} be the highest thin item that is bottom-aligned with s_{i-1}. Note that the blocking item that blocks the packing after the i-th jump is packed directly above s'_{i-1}. See Figure 2 for an illustration.

It is obvious that the first jump item s_1, that is actually the first thin item that arrives, can be packed.

For the induction step we assume $\mathrm{ONL}(s_i) \leq \rho\, \mathrm{OPT}(s_i)$. Before a jump can become unavoidable, new blocking items of total height greater than $\beta\, s_i$ need to arrive as otherwise the gap below s_i could accommodate all of them. Let h' be the height of the blocking items that are packed into the β-gap below s_i and let h'' be the total height

of blocking items that arrive between s_i and s_{i+1} and are packed above s_i. We have $h' \leq (\rho-2)s_i$ and $h'+h'' > (\rho-2)s_i$ as otherwise no blocking item would be packed on top. As further blocking items could be packed even below s'_{i-1} we get

$$\begin{aligned}\mathrm{OPT}(s_{i+1}) &\geq \mathrm{OPT}(s_i) + h' + h'' + s_{i+1} - s_i\\ \mathrm{ONL}(s_{i+1}) &= \mathrm{ONL}(s_i) + s'_i - s_i + h'' + \beta s_{i+1} + s_{i+1}.\end{aligned}$$

And thus we have

$$\begin{aligned}&&\mathrm{ONL}(s_{i+1}) &\leq \rho\,\mathrm{OPT}(s_{i+1})\\ \Leftarrow\quad &\mathrm{ONL}(s_i) + s'_i - s_i + h'' + \beta s_{i+1} + s_{i+1} &\leq \rho\big(\mathrm{OPT}(s_i) + h' + h'' + s_{i+1} - s_i\big)\\ \Leftarrow\quad &(\rho-1)s_i + s'_i - \rho h' - (\rho-1)h'' &\leq (\rho-1-\beta)s_{i+1}.\end{aligned}$$

As $\rho - 1 - \beta = 1$ and $s_{i+1} > s'_i + (\rho-2)s_i - h'$ this is satisfied if

$$\begin{aligned}&&(\rho-1)s_i + s'_i - \rho h' - (\rho-1)h'' &\leq s'_i + (\rho-2)s_i - h'\\ \Leftrightarrow\quad && s_i &\leq (\rho-1)(h'+h'')\\ \Leftarrow\quad && s_i &\leq (\rho-1)(\rho-2)s_i = s_i.\end{aligned}$$

The last equality holds since $\rho = (3+\sqrt{5})/2$ and thus $(\rho-1)(\rho-2) = 1$. Summarizing, we arrive at

Theorem 2. ONL *is a ρ-competitive algorithm for packing primitive sequences with*

$$\rho = \frac{3+\sqrt{5}}{2} \approx 2.618.$$

So the true best possible competitive ratio for packing primitive sequences is somewhere in between the two values specified by Theorems 1 and 2. We have reasons to believe that it is strictly in between these two. But perhaps an even more challenging question is whether or not (or to what extent) primitive sequences provide worst case instances for online packing in general.

References

1. Brown, D.J., Baker, B.S., Katseff, H.P.: Lower bounds for online two-dimensional packing algorithms. Acta Informatica 18, 207–225 (1982)
2. Hurink, J., Paulus, J.: Online scheduling of parallel jobs on two machines is 2-competitive. Operations Research Letters 36(1), 51–56 (2008)
3. Hurink, J.L., Paulus, J.J.: Online Algorithm for Parallel Job Scheduling and Strip Packing. In: Kaklamanis, C., Skutella, M. (eds.) WAOA 2007. LNCS, vol. 4927, pp. 67–74. Springer, Heidelberg (2008)
4. Johannes, B.: Scheduling parallel jobs to minimize the makespan. Journal of Scheduling 9(5), 433–452 (2006)
5. Kern, W., Paulus, J.: A tight analysis of Brown-Baker-Katseff sequences for online strip packing. Submitted (J. Combinatorial Opt.)
6. Ye, D., Han, X., Zhang, G.: A note on online strip packing. Journal of Combinatorial Opt. 17(4), 417–423 (2009)

Competitive Router Scheduling with Structured Data*

Yishay Mansour[1,**], Boaz Patt-Shamir[2,***], and Dror Rawitz[2]

[1] School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
mansour@cs.tau.ac.il
[2] School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel
{boaz,rawitz}@eng.tau.ac.il

Abstract. We consider the task of transmitting structured information over bounded-capacity links. Our information model is a stream of basic units called *superpackets* that are broken into k packets each. To model the possible structure and redundancy of the superpackets, we assume that for each superpacket there is a collection of minimal subsets of packets whose delivery makes the superpacket *useful*. This very general model encompasses, for example, MPEG streams, where one can think of a group of pictures (GoP) as a superpacket. The fundamental difficulty is that networks can forward only the primitive packets, but applications can use only superpackets, and thus if no minimal subset is delivered, the whole superpacket becomes useless. Our aim is to maximize goodput (number of useful superpackets) in the face of overloaded communication links, where we are forced to drop some packets.

Specifically, we assume that an arbitrary stream of packets arrives at a router with multiple bounded-capacity outgoing links. An on-line algorithm needs to decide, for each superpacket, which outgoing link to use (all packets of the same superpacket must use the same link) and, in case of an overload at a link, which packets to drop and which to transmit so as to maximize goodput. We analyze a simple randomized competitive algorithm to the general case and provide a nearly matching lower bound on the competitive ratio of any randomized on-line algorithm.

1 Introduction

Consider a video stream encoded in MPEG-2 [1]. Grossly oversimplifying, the structure of the stream is as follows. The stream is broken into *Groups of Pictures* (GoP), which may last a few minutes each. A GoP consists of a single *I-frame*, a few *P-frames*, and many *B-frames*. An I-frame is a stand-alone picture that

* Research supported in part by the Next Generation Video Consortium, Israel.
** Supported in part by a grant from the Israel Science Foundation (grant No. 709/09) and grant No. 2008-321 from the United States-Israel Binational Science Foundation (BSF), and by Israel Ministry of Science and Technology.
*** Supported in part by the Israel Science Foundation (grant 1372/09) and by Israel Ministry of Science and Technology.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 219–232, 2012.

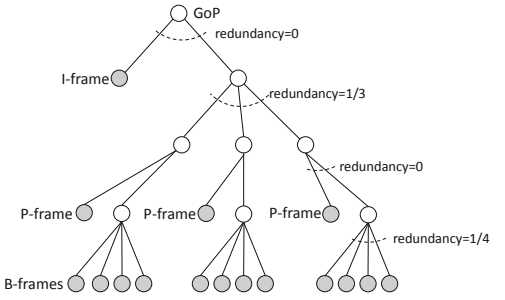


Fig. 1. A tree representation of a GoP. Gray nodes represent data. A node with redundancy β is deemed useful if no more than a fraction β of its children is non-useful.

requires no other information for decoding; decoding a P-frame requires its preceding I-frame; and decoding a B-frame requires its preceding "reference frame" (be it I- or P-frame).[1] The implication of this structure is that if an I-frame is lost, then the whole GoP is lost, and if a B-frame is lost, then only a fraction of a second is lost. But then again, if too many B-frames are lost (where "too many" is defined subjectively), the GoP should be considered again worthless. This structure can be modeled by a tree. Figure 1 illustrates a simple example.

The root represents the GoP; if either the I-frame (left child) or the other data (right subtree) are lost, then the GoP is lost; however, the right subtree may be considered useful even if one of its children is lost; and similarly, each of these (depth 2) nodes is useful only if both its P-frame child and at least $3/4$ of its B-frames are not lost. While this is not an accurate description of MPEG, we note that the hierarchical tree structure is very natural and appears in many other formats (e.g., XML documents [2]), with or without redundancy. Conceivably, more complex forms of redundancy are also used.

So suppose now that we need to manage a router that delivers multiple video streams, such that each stream may use any of a number of outgoing links (see Figure 2). At every step, some packets arrive at the router, and the router needs to decide which outgoing link is used for each packet, and, in case of an overflow in that link, which packets to discard. Note that in our example, if we drop an I-frame from each GoP, then all GoP's are useless at the receiving ends, even if the link has delivered all P- and B-frames (this is an instance of a high throughput, low goodput situation). In this paper we study, from the theoretical viewpoint, algorithms that decide which packets to drop so as to maximize the goodput of bounded-capacity links.

To this end, we consider the following abstract model. Senders generate basic information units, called *superpackets*, that are broken into packets by the network protocol at the senders. The router needs to decide which link is used by each new superpacket: all subsequent packets of that superpacket must use

[1] In fact, P-frames depend on the previous reference frame; and B-frames depend on both their immediate surrounding frames. In addition, MPEG partitions frames into "slices," which are transmitted in network packets.

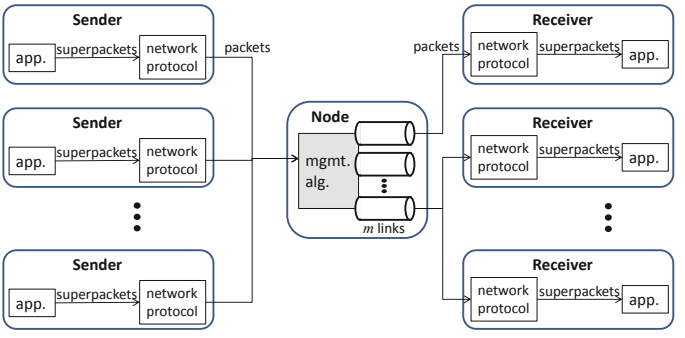


Fig. 2. Basic system setup. Our focus is on the link management algorithm (shaded). All packets belonging to the same superpacket must use the same link.

the same link.[2] If the number of packets assigned to a link exceeds its capacity, the management algorithm needs to decide which packet to drop and which to forward. To allow for arbitrary structure and redundancy, we assume that each superpacket is associated with a collection of *feasible subsets* that is closed under set inclusion (i.e., if $S \supset S'$ and S' is feasible, then so is S). A superpacket is considered useful only if the set of its delivered packets is one of its feasible subsets. The goal of the algorithm is to maximize the number useful superpackets at the receivers.

Our Contribution. Following [3,4], we study the fundamental PRIORITY algorithm for link management, augmented with a simple randomized strategy that allocates superpackets to links based on the links capacities. Algorithm PRIORITY assigns to each superpacket a random priority (based only on its weight, cf. Section 2.2), and in case of overflows, low-priority packets are dropped. This algorithm enjoys many nice features: in particular let us mention that it is highly distributed in the sense that it can be employed consistently in multiple locations without any communication overhead (see [3]). For our context, we note that the algorithm works without any knowledge of the feasible sets.

Our main result is competitive analysis of this simple algorithm. Specifically, suppose that at most σ packets arrive in a step and that each link can serve at most one packet per step; suppose further that each superpacket contains k packets. We prove that the algorithm guarantees expected goodput of $\Omega(\text{OPT}/(k\sqrt{\sigma/m}))$, where OPT denotes the maximal number of superpackets that can be delivered in the given input sequence and m is the number of links. In fact, we prove our result in a more general setting: first, we consider *weighted* goodput (i.e., when each superpacket has a different value, and the goal is to maximize the total value of useful superpackets), where the competitive ratio is not affected; and second, we consider *capacitated* links, where each link i has capacity c_i. In this case the expected weight of superpackets delivered by our

[2] This requirement, referred to as "stickiness" or "persistence" is typical in communication protocols, e.g. TCP.

algorithm is $\Omega(\text{OPT}/(k\sqrt{\sigma/c}))$, where $c = \sum_{i=1}^{m} c_i$. Notice that the competitive ratio depends only on the total available bandwidth c, regardless of the way it is broken into links. Also note that the competitive ratio improves linearly with $\sqrt{c}$. Finally, we provide a refined analysis that takes into account a parameter we define, called the *effective redundancy* of the input sequence.

We present a lower bound on the competitive ratio for the case of m unit capacity links without redundancy. Based on [3], we show that in this case, no randomized on-line algorithm can improve on our results by more than a polylogarithmic factor.

We show that our results extend to more general models. In some cases, there may be more than just two values for a superpacket (no value or full value). Superpackets may be structured so that there are a few "service levels," with different values, so that the value of a delivered superpacket is the value of the highest satisfied service level. We show that our algorithm is competitive in this model as well. We also show that our upper bound applies to the *instantaneous network* model, where we are given a network, a source s and a destination t, and the algorithm needs to choose a path from s to t for each superpacket: all packets of a superpacket must follow the same path. A conflict between two superpackets occurs, if the routes of the superpackets intersect, and there exists a time step in which packets from both arrive. The motivation for this model is the case where a superpacket is a set of short virtual circuits between s and t over a network of unit-capacity links.

Related Work. Buffer overflow management has been studied extensively in the last decade from the competitive analysis viewpoint (starting with [5,6]: see [7] for a recent survey). The simplest superpacket model, in which each superpacket consists of k packets that need all be delivered, was introduced in [8]. Emek at al. [3] consider the basic problem (k-packet superpackets, single link, no redundancy) under the name Online Set Packing, and introduce the PRIORITY algorithm (based on Turan's Theorem [9]). They prove an upper bound on the competitive ratio of PRIORITY and a lower bound on the competitive ratio of any on-line algorithm for that problem. In [4], basic redundancy is introduced: in our terms, there is a constant $0 \leq \beta < 1$ such that any subset of at least $(1-\beta)k$ packets is feasible (in other words, a super packet is useful if at most a β-fraction of its packets are lost). A general technique for dealing with buffers is also introduced in [4].

The offline version of single link management, without redundancy and superpacket structure, is equivalent to the Set Packing problem (SP), where each superpacket corresponds to a set and each time step corresponds to an element. SP is as hard as Maximum Independent Set even when all elements are contained in at most two sets (i.e., $\sigma \leq 2$), and therefore cannot be approximated to within an $O(n^{1-\epsilon})$-factor, for any $\epsilon > 0$, where n is the number of sets [10]. Letting T denote the number of time steps (elements), SP is $O(\sqrt{T})$-approximable, and hard to approximate within $T^{1/2-\epsilon}$, for any $\epsilon > 0$ [11]. When set size is at most k, SP is approximable within $\frac{k}{2} + \epsilon$, for any $\epsilon > 0$ [12] and within $\frac{k+1}{2}$ in the weighted case [13], but known to be hard to approximate to within $O(k/\log k)$-factor [14].

Paper Organization. The remainder of the paper is organized as follows. In Section 2 we formalize the model and present our algorithm. The analysis of our algorithm and the lower bound are given in Section 3. Extensions are given in Section 4.

2 Preliminaries

2.1 Models

Data Model. Our basic concept is a *superpacket*, typically denoted S, which is comprised of k *packets*. The complete set of superpackets is denoted $\mathcal{C}$. Each superpacket $S \in \mathcal{C}$ is associated with a *feasibility collection* $\mathcal{F}_S \subseteq 2^S$. $\bar{S} \in \mathcal{F}_S$ is called a *feasible subset* of S. We assume that $\mathcal{F}_S$ is closed under set inclusion, or *monotone*, for any superpacket S, namely that if $\bar{S} \in \mathcal{F}_S$, then $S' \in \mathcal{F}_S$ for any S' such that $\bar{S} \subseteq S'$. The case where $\mathcal{F}_S = \{S\}$, for every $S \in \mathcal{S}$, is referred to as *all-or-nothing*. In this case a superpacket is lost if even one of its packets is dropped.

Each superpacket $S \in \mathcal{C}$ has a *weight* $w(S) > 0$. In the *unweighted* model, $w(S) = 1$ for all $S \in \mathcal{C}$. Given a set of superpackets $\mathcal{C}' \subseteq \mathcal{C}$ we define $w(\mathcal{C}') \stackrel{\text{def}}{=} \sum_{S \in \mathcal{C}'} w(S)$. The input is a sequence of packets that arrive online. We assume that the online algorithm can associate packets with superpackets (e.g., packets contain their parent superpacket ID in their headers). We stress, however, that the algorithm has no knowledge on feasibility collections of superpackets. The system progresses in discrete time steps, where the time horizon is denoted by T. In each step t, a set of $\sigma(t)$ packets arrive. (We assume that no superpacket has two packets arriving at the same step.) The arrival time of a packet p is denoted by $\text{arr}(p)$. The set of superpackets whose packet arrive at time t is denoted $\mathcal{C}(t)$, i.e., $\mathcal{C}(t) = \{S \in \mathcal{C} : \exists p \in S \text{ s.t. } \text{arr}(p) = t\}$. The *burst size* at time t is denoted $\sigma(t) = |\{p : \text{arr}(p) = t\}|$; the *weighted* burst size is denoted $\sigma_\$(t) = \sum_{S \in \mathcal{C}(t)} w(S)$.

System Model. In the *single-link model*, we have an integer *capacity* $c \geq 1$, and the algorithm selects, at each time step t, c packets to forward. All other packets that arrived at time t are lost (possibly causing the loss of their superpackets). In the *multiple links* model, there are m links with capacities $c_1, \ldots, c_m$, where $c_i \geq 1$ for $i = 1, \ldots, m$. We denote $c = \sum_{i=1}^{m} c_i$. The algorithm selects a single link for each superpacket (for all its packets), and then, in each time step, the algorithm does, for each link, the single link task: select which packets will be forwarded, subject to that link capacity constraint.

Given an algorithm ALG and an instance $\mathcal{I}$, we denote the set of completed superpackets by $\text{ALG}(\mathcal{I})$ (or simply by ALG), and call it the *goodput* of the algorithm. If the algorithm is randomized, its goodput for a given instance is a random variable, and we shall refer to its expected value. We measure the performance of algorithms using competitive analysis: The *competitive ratio* of an algorithm is the supremum, over all instances $\mathcal{I}$, of $w(\text{OPT}(\mathcal{I}))/w(\text{ALG}(\mathcal{I}))$, where $w(\text{OPT}(\mathcal{I}))$ is the maximum possible goodput for $\mathcal{I}$.

Additional Notation. We define for every set of packets S,

$$N[S] \stackrel{\text{def}}{=} \{S' \in \mathcal{C} : \exists p \in S, p' \in S' \text{ s.t. } \text{arr}(p) = \text{arr}(p')\}$$

and $N(S) \stackrel{\text{def}}{=} N[S] \setminus \{S\}$. Notice that $N(S) = N[S]$ if $S \notin \mathcal{C}$. For a finite sequence of values $x_1, \ldots, x_n$, we denote $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $x_{\max} = \max\{x_1, \ldots, x_n\}$. We use the notation $\overline{xy} = \frac{1}{n}\sum_{i=1}^{n} x_i y_i$.

2.2 Algorithm

The following algorithm was proposed in [3] for the Online Set Packing problem.

Algorithm 1. PRIORITY

1: For each superpacket S with weight $w(S)$: select a priority $r(S) \in [0,1]$ independently at random by the cumulative distribution function $\Pr[r(S) < x] = x^{w(S)}$.
2: **for all** time step t **do**
3: Receive $\sigma(t)$ packets
4: Deliver the c packets whose superpackets have the largest priority.
5: **end for**

Note that, if $w(S)$ is integral, the priority $r(S) \in [0,1]$ of a superpacket S is distributed like the maximum of $w(S)$ independent $U[0,1]$ random variables.

For our case, where we have m links, we use the following simple link allocation algorithm. Each link is then managed by its own replica of PRIORITY.

Algorithm 2. PLINK

1: Whenever a packet p from superpacket S arrives:
2: **if** p is the first packet from S **then**
3: set $\ell(S) \in \{1, \ldots, m\}$ randomly by $\Pr[\ell(S) = i] = \frac{c_i}{c}$
4: **end if** ▷ *else $\ell(S)$ was set previously*
5: send p to link $\ell(S)$ (managed by Algorithm PRIORITY$_{\ell(S)}$)

3 Multiple Capacitated Links

In this section we study the case where weighted superpackets arrive at a server with multiple links. We show that the competitive ratio of PLINK is $O(k\sqrt{\overline{\sigma\sigma_\$}/c \cdot \overline{\sigma_\$}})$. (Recall that c is the total capacity of all links.) We also present an almost matching lower bound for the all-or-nothing case with unit capacity links that is based on the lower bound for single unit capacity link from [3]. Our lower bound applies even to unweighted input sequences. We conclude the section with a discussion on the difference between the effects of many links and of large capacity.

3.1 Analysis of Algorithm PLink

We start by stating lower bounds on the survival probability of a superpacket S under PRIORITY, in the single link case. (We abuse notation by using PRIORITY to denote the set of surviving superpackets.) Note that the bounds hold for arbitrary feasible subsets of S. (Similar results appear in [3], but assuming that the only feasible set is S itself). Due to lack of space, the proofs are omitted.

Lemma 1. *Let $c = 1$. For any superpacket $S \in \mathcal{C}$ and for any feasible subset $\bar{S}$ of S we have* $\Pr[S \in \text{PRIORITY}] \geq \frac{w(S)}{w(N(\bar{S}))+w(S)}$.

Lemma 2. *Let $c > 1$. For any superpacket $S \in \mathcal{C}$ and for any feasible subset $\bar{S}$ of S we have* $\Pr[S \in \text{PRIORITY}] \geq \frac{1}{2} \cdot \min\left\{\frac{c \cdot w(S)}{w(N(\bar{S}))+w(S)}, 1\right\}$.

We now extend the lemmas to the multiple links case. We abuse notation once again by using PLINK to denote the set of surviving superpackets.

Lemma 3. *For any $S \in \mathcal{C}$ and for any feasible subset $\bar{S}$ of S we have*

$$\Pr[S \in \text{PLINK}] \geq \frac{c}{4} \cdot \frac{w(S)}{2w(N(\bar{S})) + c \cdot w(S)} .$$

Proof. Let $\ell(S)$ denote the link that was selected for S by PLINK. Also let $N_i(S) = \{S' \in N(S) : \ell(S') = i\}$. By the independence of the random choices we get that $\mathbb{E}[w(N_i(\bar{S}))] = \frac{c_i}{c} \cdot w(N(\bar{S}))$, and by Markov's Inequality we have that $\Pr\left[w(N_i(\bar{S})) > \frac{2c_i}{c} \cdot w(N(\bar{S}))\right] < \frac{1}{2}$.

According to Lemmas 1 and 2 it follows that

$$\begin{aligned}
\Pr\left[S \in \text{PLINK} \mid \begin{smallmatrix}\ell(S)=i \text{ and} \\ w(N_i(\bar{S})) \leq \frac{2c_i w(N(\bar{S}))}{c}\end{smallmatrix}\right] &\geq \frac{1}{2} \min\left\{\frac{c_i w(S)}{\frac{2c_i w(N(\bar{S}))}{c} + w(S)}, 1\right\} \\
&\geq \frac{1}{2} \min\left\{\frac{c_i w(S)}{\frac{2c_i w(N(\bar{S}))}{c} + c_i w(S)}, 1\right\} \\
&= \frac{1}{2} \min\left\{\frac{cw(S)}{2w(N(\bar{S})) + cw(S)}, 1\right\} \\
&= \frac{1}{2} \cdot \frac{cw(S)}{2w(N(\bar{S})) + cw(S)}
\end{aligned}$$

Hence

$$\begin{aligned}
\Pr[S \in \text{PLINK} \mid \ell(S) = i] \geq & \Pr\left[w(N_i(\bar{S})) \leq \frac{2c_i w(N(\bar{S}))}{c} \mid \ell(S) = i\right] \cdot \\
& \Pr\left[S \in \text{PLINK} \mid \begin{smallmatrix}\ell(S)=i \text{ and} \\ w(N_i(\bar{S})) \leq \frac{2c_i w(N(\bar{S}))}{c}\end{smallmatrix}\right] \\
\geq & \frac{c}{4} \cdot \frac{w(S)}{2w(N(\bar{S})) + cw(S)} .
\end{aligned}$$

Therefore,

$$\Pr[S \in \text{PLINK}] \;=\; \sum_i \frac{c_i}{c} \cdot \Pr[S \in \text{PLINK} \mid \ell(S) = i] \;\geq\; \frac{c}{4} \cdot \frac{w(S)}{2w(N(\bar{S})) + cw(S)} \;.$$

□

The following lemma states the property that allows us to bound the competitive ratio of PLINK.

Lemma 4. *Let $\mathcal{C}' \subseteq \mathcal{C}$ be a collection of superpackets, and for every $S \in \mathcal{C}'$ let $\bar{S} \subseteq S$ be a feasible subset of S. Then, either (i) $\mathbb{E}[w(\text{PLINK})] \geq \frac{w(\mathcal{C}')}{8}$, or (ii) $\mathbb{E}[w(\text{PLINK})] \geq \frac{c}{16} \frac{w(\mathcal{C}')^2}{\sum_{S \in \mathcal{C}'} w(N(S))}$.*

Proof. By linearity of expectation we have

$$\begin{aligned}
\mathbb{E}[w(\text{PLINK})] \;&=\; \sum_{S \in \mathcal{C}} w(S) \cdot \Pr[S \in \text{PLINK}] \\
&\geq\; \sum_{S \in \mathcal{C}'} \frac{w(S)}{4} \frac{c \cdot w(S)}{2w(N(\bar{S})) + cw(S)} \\
&=\; \frac{c}{4} \sum_{S \in \mathcal{C}} \frac{w(S)^2}{2w(N(\bar{S})) + cw(S)} \\
&\geq\; \frac{c}{4} \cdot \frac{w(\mathcal{C}')^2}{2\sum_{S \in \mathcal{C}'} w(N(\bar{S})) + cw(\mathcal{C}')} \;,
\end{aligned}$$

where the first inequality follows from Lemma 3 and the second is due to the following implication of the Cauchy-Schwarz Inequality: for any positive reals $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$, it holds that $\sum_i \frac{a_i^2}{b_i} \geq \frac{\left(\sum_i a_i\right)^2}{\sum_i b_i}$.

If $c \cdot w(\mathcal{C}') \geq 2\sum_{S \in \mathcal{C}'} w(N(\bar{S}))$, then $\mathbb{E}[w(\text{PLINK})] \geq \frac{c}{4} \cdot \frac{w(\mathcal{C}')^2}{2cw(\mathcal{C}')} = \frac{w(\mathcal{C}')}{8}$, and otherwise, $\mathbb{E}[w(\text{PLINK})] \;>\; \frac{c}{16} \cdot \frac{w(\mathcal{C}')^2}{\sum_{S \in \mathcal{C}'} w(N(\bar{S}))}$. □

Lemmas 5 and 6 below apply Lemma 4 with two different collections $\mathcal{C}'$.

Lemma 5. *Either $\mathbb{E}[w(\text{PLINK})] \geq \frac{w(\text{OPT})}{8}$, or $\mathbb{E}[w(\text{PLINK})] \geq \frac{w(\text{OPT})^2}{16kw(\mathcal{C})}$.*

Proof. For each superpacket $S \in \text{OPT}$ fix $\bar{S}$ to be the subset of S which contains the packets delivered by OPT. Clearly $\bar{S}$ is a feasible subset of S. Hence, by Lemma 4 with $\mathcal{C}' = \text{OPT}$ we have that either $\mathbb{E}[w(\text{PLINK})] \geq \frac{w(\text{OPT})}{8}$ or $\mathbb{E}[w(\text{PLINK})] \geq \frac{c}{16} \frac{w(\text{OPT})^2}{\sum_{S \in \text{OPT}} w(N(S))}$. In the latter case, observe that each superpacket in $\mathcal{C}$ intersects at most ck superpackets in OPT. Hence, $\sum_{S \in \text{OPT}} w(N(\bar{S})) \;\leq\; ckw(\mathcal{C})$. It follows that $\mathbb{E}[w(\text{PLINK})] \geq \frac{w(\text{OPT})^2}{16kw(\mathcal{C})}$. □

Lemma 6. *Either $\mathbb{E}[w(\text{PLINK})] \geq \frac{w(\mathcal{C})}{8}$, or $\mathbb{E}[w(\text{PLINK})] \geq \frac{cw(\mathcal{C})^2}{16T \cdot \bar{\sigma}\sigma_{\$}}$.*

Proof. Fix a superpacket S and let $\bar{S} = S$. By Lemma 4 with $\mathcal{C}' = \mathcal{C}$ we have that either $\mathbb{E}[w(\text{PLINK})] \geq \frac{w(C)}{8}$, or $\mathbb{E}[w(\text{PLINK})] \geq \frac{c}{16} \cdot \frac{w(\mathcal{C})^2}{\sum_{S \in \mathcal{C}} w(N(S))}$. Summing over the superpackets we get that

$$\sum_{S \in \mathcal{C}} w(N(S)) \;<\; \sum_{t} \sigma(t)\sigma_{\$}(t) \;=\; T \cdot \overline{\sigma\sigma_{\$}}\,, \tag{1}$$

and the lemma follows. □

Combining Lemmas 5 and 6 we obtain our main result.

Theorem 1. *The competitive ratio of* PLINK *is at most* $16k\sqrt{\frac{\overline{\sigma\sigma_{\$}}}{c\overline{\sigma_{\$}}}}$.

Proof. If either $w(\text{PLINK}) \geq w(\text{OPT})/8$ or $w(\text{PLINK}) \geq w(\mathcal{C})/8$, then we are done. Otherwise, we have that

$$\mathbb{E}[w(\text{PLINK})] \geq \frac{w(\text{OPT})^2}{16kw(\mathcal{C})} \qquad \text{and} \qquad \mathbb{E}[w(\text{PLINK})] \geq \frac{cw(\mathcal{C})^2}{16T \cdot \overline{\sigma\sigma_{\$}}}\,.$$

The maximum of these bounds is minimized when $w(\text{OPT}) = \sqrt{\frac{ck \cdot w(\mathcal{C})^3}{T \cdot \overline{\sigma\sigma_{\$}}}}$, and therefore, for any instance

$$\mathbb{E}[w(\text{PRIORITY})] \;\geq\; w(\text{OPT}) \cdot \frac{1}{16}\sqrt{\frac{cw(\mathcal{C})}{k \cdot T \cdot \overline{\sigma\sigma_{\$}}}}.$$

Finally, since $T \cdot \overline{\sigma_{\$}} \;=\; \sum_t \sigma_{\$}(t) \;\leq\; \sum_{S \in \mathcal{C}} k \cdot w(S) \;=\; k \cdot w(\mathcal{C})$, it follows that

$$\mathbb{E}[(\text{PRIORITY})] \;\geq\; w(\text{OPT}) \cdot \frac{1}{16}\sqrt{\frac{c \cdot \overline{\sigma_{\$}}}{k^2\overline{\sigma\sigma_{\$}}}} \;=\; w(\text{OPT}) \cdot \frac{1}{16k}\sqrt{\frac{c \cdot \overline{\sigma_{\$}}}{\overline{\sigma\sigma_{\$}}}}\,.$$

□

Note that the upper bound we provide in Theorem 1 does not depend on the number of links, but rather on the input sequence and on the total capacity of the links.

Corollary 1. *The competitive ratio of* PLINK *is at most* $16k\sqrt{\sigma_{\max}/c}$.

Proof. Follows from the fact that $\overline{\sigma\sigma_{\$}} \leq \overline{\sigma_{\$}} \cdot \sigma_{\max}$. □

3.2 A Lower Bound

We now present a lower bound for the multiple links case. It uses the simple scenario of unweighted, unit-capacity per link (i.e., $m = c$) instances, and thus it applies to more general setting a fortiori. However, we assume that the only feasible subset of a superpacket is all packets, i.e., no redundancy is considered.

Our lower bound uses, as a black box, the following lower bound from [3] for Online Set Packing (OSP).

Theorem 2 ([3]). *For any randomized online algorithm, there exists an infinite family of unweighted, unit-capacity instances of* OSP *for which the competitive ratio is* $\tilde{\Omega}(k\sqrt{\sigma_{\max}})$.

Next building on Theorem 2 we obtain a lower bound for the multiple uncapacitated links case.

Theorem 3. *For any online randomized algorithm there exists an infinite family of unweighted, instances for which, under the* m *unit capacity link model, the competitive ratio is* $\tilde{\Omega}(k\sqrt{\sigma_{\max}/c})$.

Proof. Let $\mathcal{I}'$ be the instance whose existence is promised by Theorem 2. Define an instance $\mathcal{I}$ where each superpacket in $\mathcal{I}'$ is replicated c times. (Note that $c = m$ in this case.) Clearly, $|\textsc{opt}(\mathcal{I})| \geq c \cdot |\textsc{opt}(\mathcal{I}')|$, since it is possible to route the ith copy of each set to link i. We show that given any randomized online algorithm ALG for the multiple links case, one can obtain an algorithm ALG$'$ for the single link case such that $\mathbb{E}[\,|\textsc{alg}'|\,] \geq \mathbb{E}[\,|\textsc{alg}|\,]/c$. Hence, $|\textsc{opt}|/\mathbb{E}[\,|\textsc{alg}|\,] \geq |\textsc{opt}'|/\mathbb{E}[\,|\textsc{alg}'|\,]$, and the theorem follows.

Given an algorithm ALG, define $\textsc{alg}_i$ to be the set of completed superpackets that were routed to link i. Let ℓ be the link that maximizes performance, i.e., $\ell = \operatorname{argmax}_i \mathbb{E}[|\textsc{alg}_i|]$. Clearly, $\mathbb{E}[|\textsc{alg}_i|] \geq \mathbb{E}[|\textsc{alg}|]/c$. We construct an algorithm ALG$'$ for OSP that simulates link ℓ. More specifically, given an input sequence, ALG$'$ makes m copies of each superpacket, and executes ALG on the new instance. Let $\textsc{alg}_\ell(t)$ be the set of superpackets whose packets were transmitted by ALG on link ℓ at time t. If $S_j \in \textsc{alg}_\ell(t)$, where S^j is a copy of S, then ALG$'$ transmits a corresponding packet from S, namely $\textsc{alg}'(t) = \{S \,:\, \exists j,\ S^j \in \textsc{alg}_\ell(t)\}$. Since no two copies of S can be completed by ALG, it follows that $|\textsc{alg}'| \geq |\textsc{alg}_\ell|$. Hence, $\mathbb{E}[|\textsc{alg}'|] \geq \mathbb{E}[|\textsc{alg}|]/c$, as required. □

We note that our lower bound shows that our upper bound is essentially tight—for the case of unit capacity links.

3.3 The Effect of Many Links and Large Capacity

As we mentioned above, it is interesting to note that the competitive ratio of Algorithm PLink depends only on the total available bandwidth, regardless of how it is partitioned among the links. However, the lower bound of Theorem 3 is proved specifically for the case of unit capacity links. It is natural to ask whether link capacity plays an important role in algorithm performance. The answer is positive, as demonstrated by the following scenario. Consider two models, one with m unit-capacity links and another with a single link with capacity m. Suppose that there are $n \geq m$ superpackets without any redundancy, and let the arrival sequence consists of all possible $\binom{n}{m}$ bursts of size m in arbitrary order. In the m unit-capacity links model, only m superpackets can be completely delivered, because each channel can deliver only one complete superpacket. However, in the single-link capacity m model, all superpackets are delivered. This means that the optimum may change dramatically when links are consolidated.

We conclude this section with two observations about the effect of sufficiently many unit capacity links. The proofs are omitted.

First, we consider the effect on the optimal solution.

Observation 7. *If* $m > k(\sigma_{\max} - 1)$*, then* $\text{OPT} = \mathcal{C}$*.*

Next, we consider the effect on the competitive ratio,

Theorem 4. *Suppose that* ALG *is an algorithm that assigns superpackets to links uniformly at random, and consider unweighted instances without redundancy. Then if* $m \geq \frac{k\overline{\sigma^2}}{\epsilon\overline{\sigma}}$ *then* $\mathbb{E}[\,|\text{ALG}|\,] \geq (1-\epsilon)|\text{OPT}|$*.*

We note that since $\frac{\overline{\sigma^2}}{\overline{\sigma}} \leq \sigma_{\max}$, the same result holds for $m \geq \frac{k\sigma_{\max}}{\epsilon}$.

4 Extensions

In this section we present a refinement of the analysis of PLink, and then we extend the analysis to more general settings. More specifically, we provide a refined analysis of PLink that takes into account the *effective redundancy* of the input sequence (Section 4.1). We show that PLink can be used in the case where there are several feasibility collections for each superpacket, and each collection is associated with a different service level (Section 4.2). We also extend our results to the *instantaneous network* model (Section 4.3).

4.1 Effective Redundancy

To refine the analysis of Algorithm PLink, we defined the following concepts. The *burstiness* of a superpacket S is defined as $\mathrm{B}(S) = \sum_{p \in S} \sigma_{\$}(\mathrm{arr}(p))$. The minimal burstiness of S is $\mathrm{B}_{\min}(S) = \min_{\bar{S} \in \mathcal{F}_S} \mathrm{B}(\bar{S})$. Let $\rho_S = \mathrm{B}_{\min}(S)/\mathrm{B}(S)$ and $\rho = \max_{S \in \mathcal{C}} \rho_S$. ρ is called the *effective redundancy* of the input sequence.

We now refine Lemma 6 to include ρ as follows.

Lemma 8. *Either* $\mathbb{E}[w(\text{PLink})] \geq \frac{w(\mathcal{C})}{8}$*, or* $\mathbb{E}[w(\text{PLink})] \geq \frac{cw(\mathcal{C})^2}{16\rho T \cdot \overline{\sigma\sigma_{\$}}}$*.*

Proof. We follow the proof of Lemma 6, but we take $\bar{S}$ to be a feasible subset of S with minimal burstiness, namely such that $\mathrm{B}(\bar{S}) = \mathrm{B}_{\min}(S)$. Equation (1) is replaced with $\sum_{S \in \mathcal{C}} w(N(\bar{S})) < \sum_{S \in \mathcal{C}} \mathrm{B}(\bar{S}) \leq \sum_{S \in \mathcal{C}} \rho \mathrm{B}(S) = \rho \cdot T \cdot \overline{\sigma\sigma_{\$}}$. □

We can therefore conclude that in this case we have an improvement of $\sqrt{\rho}$ factor over Theorem 1:

Theorem 5. *The competitive ratio of* PLink *is at most* $16k\sqrt{\frac{\rho\overline{\sigma\sigma_{\$}}}{c\overline{\sigma_{\$}}}}$*.*

We note that ρ decreases if bursts are roughly the same weight or if there are no packets whose delivery is essential to the survival of superpackets.

To motivate the parameter ρ, consider the case where superpackets are hierarchically structured. Specifically, we assume that the feasible collection of a superpacket S is defined by a *structure tree* T_S, whose leaves are the packets, and whose root is identified with the superpacket. A structure tree is a rooted tree with a *redundancy parameter* assigned to each node, subject to the following restriction for internal nodes: Let $d(v)$ denote the number of children of a node v. The redundancy parameter of a node v is $\beta_v^S \in \{0, \frac{1}{d_v}, \ldots, \frac{d_v-1}{d_v}\}$. (Assume w.l.o.g. that $\beta_v^S = 0$ if $d(v) \leq 1$.) The interpretation of redundancy is defined recursively as follows. A subset S' of the leaves of a structure tree T is said to be *feasible* if either of the following conditions hold: (i) S' contains one packet and T contains one leaf; or (ii) Let $v_1, \ldots, v_d$ be the children of the root of T, with structure trees $T_1, \ldots, T_d$, respectively. Let $S'_1, \ldots, S'_d$ be the subsets of S' corresponding to $T_1, \ldots, T_d$, resp. Then S' is feasible if at least $(1 - \beta_r^S)d$ of the subsets $S'_1, \ldots, S'_d$ are feasible.

Consider a superpacket S with its structure tree T_S. We define the *redundancy* of a leaf v in T_S as $\hat{\rho}_v \stackrel{\text{def}}{=} \prod_{i=0}^{\ell}(1 - \beta_{v_i}^S)$, where $v = v_0, v_1, \ldots, v_\ell = r$ is the path from v to the root r. The redundancy of the superpacket S is defined as $\hat{\rho}_S \stackrel{\text{def}}{=} \max_{v \text{ is a leaf}} \rho_v$. Notice that $\hat{\rho}_S$ depends on T_S and on β^S, but not on the input sequence.

Observation 9. *$\rho_S \leq \hat{\rho}_S$ for every superpacket $S \in \mathcal{C}$.*

It follows that we can replace ρ with $\hat{\rho} = \max_S \hat{\rho}_S$ in Theorem 5. We note that $\hat{\rho}_S = 1$ in the GoP example (see Figure 1), since the I-frame is contained in any feasible set. However, the competitive ratio will improve, if we send the I-frame twice.

4.2 Multiple Service Levels

In the model considered in Section 3, each superpacket S has a single weight (value) $w(S)$ that is collected if a feasible subset of S is delivered. In some cases, there may be more than just two values for the superpacket (no value or full value). We consider this case here. Intuitively, we consider superpackets structured so that there are a few "service levels," with different values, so that the value of a delivered superpacket is the value of the highest satisfied service level. We show that Algorithm PLink is competitive in this case as well.

Formally, we assume that with each superpacket i there are ℓ feasibility collections $\mathcal{F}_S^1 \supset \mathcal{F}_S^2 \cdots \supseteq \mathcal{F}_S^\ell$ and a weight $w(S)$. There are also ℓ payment levels $0 < \alpha_1 < \cdots < \alpha_\ell \leq 1$, such that the profit obtained from a superpacket S with delivered packets S' is $\alpha_i \cdot w(S)$, where i is the maximal service level i such that $S' \subseteq \mathcal{F}_S^i$.

Let $w_i(S) \stackrel{\text{def}}{=} w(S)(\alpha_i - \alpha_{i-1})$, for $i = 1, \ldots, \ell$, namely $w_i(S)$ stands for the marginal profit obtained by going from service level $i-1$ to service level i. Let OPT_i denote the optimal value with respect to the instance with the weight function w_i.

Lemma 10. *Executing Algorithm* PLink *with the original weights results in* $\mathbb{E}[w_i(\text{PLink})] \geq 16k\sqrt{\frac{\overline{\sigma\sigma_\$}}{c\overline{\sigma_\$}}} \cdot w_i(\text{OPT}_i)$ *for a service level* i*, where* ρ_i *is defined by the feasible collections for service level* i*.*

Proof. Since w_i is proportional to the original weight function w, for every i, it follows that the analysis of Algorithm PLink continues to hold with respect to w_i, even if the random priorities are chosen according to w (see Lemmas 11-3). Therefore $\mathbb{E}[w_i(\text{PLink})] \geq 16k\sqrt{\overline{\sigma\sigma_{i\$}}/c\overline{\sigma_{i\$}}} \cdot w_i(\text{OPT}_i) = 16k\sqrt{\overline{\sigma\sigma_\$}/c\overline{\sigma_\$}} \cdot w_i(\text{OPT}_i)$. □

Theorem 6. *The competitive ratio of* PLink *is at most* $16k\sqrt{\frac{\overline{\sigma\sigma_\$}}{c\overline{\sigma_\$}}}$.

Proof. By linearity of expectation and Observation 10 we have that

$$\begin{aligned}
\mathbb{E}[w(\text{PLink})] &= \sum_i \mathbb{E}[w_i(\text{PLink})] \\
&\geq \sum_i 16k\sqrt{\frac{\overline{\sigma\sigma_\$}}{c\overline{\sigma_\$}}} \cdot w_i(\text{OPT}_i) \\
&\geq 16k\sqrt{\frac{\overline{\sigma\sigma_\$}}{c\overline{\sigma_\$}}} \sum_i w_i(\text{OPT}) \qquad = 16k\sqrt{\frac{\overline{\sigma\sigma_\$}}{c\overline{\sigma_\$}}} \cdot w(\text{OPT}) \ .
\end{aligned}$$

□

4.3 Instantaneous Network Model

We can extend our results to the following scenario we call the instantaneous network model. In this model we are given a graph with unit capacity edges and two distinguished nodes, a source s and a destination t, and the algorithm needs to choose a path from s to t for each superpacket: all packets of a superpacket must follow the same path. A conflict between superpackets S and S' occurs if the routes of S and S' intersect, and there exists a time step in which packets from both S and S' arrive.

We observe that the instantaneous network model can be reduced to the unit capacity multiple links model.

Lemma 11. *There exists a reduction from the instantaneous network model to the unit capacity multiple links model.*

Proof. Consider any feasible solution. For each superpacket S, let $p(S)$ be the path from s to t that is used for S by the solution. Let $C = \{e_1, ..., e_f\}$ be a minimum s,t-cut in the network, and for each superpacket S, let $e(S)$ be some edge in C that is contained in $p(S)$, namely $e(S) \in C \cap p(S)$. Define $\mathcal{C}_i = \{S : e(S) = e_i\}$. Clearly, $\cup_i \mathcal{C}_i = \mathcal{C}$. Let $p_1, ..., p_f$ be f simple edge disjoint paths from s to t, where $e_i \in p_i$. We reassign superpackets to paths as follows: $p'(S) = p_i$ if $e(S) = e_i$, namely if $S \in \mathcal{C}_i$. Since the superpackets in $\mathcal{C}_i$ intersect at e_i using p, no new conflict is introduced by the new assignment p'. □

It follows that

Theorem 7. *There exists a randomized algorithm for the instantaneous network model whose competitive ratio is at most* $16k\sqrt{\frac{\overline{\sigma\sigma_{\$}}}{c\overline{\sigma}_{\$}}}$.

Theorem 8. *For any online randomized algorithm for the instantaneous network model there exists an infinite family of unweighted, instances for which the competitive ratio is* $\tilde{\Omega}(k\sqrt{\sigma_{\max}/c})$.

References

1. International Organization for Standardization: MPEG-2 standard, ISO/IEC 13818-2:2000 (2000)
2. World Wide Web Consortium: Extensible markup language (XML) 1.0. W3C Recommendation (November 2008), http://www.w3.org/TR/REC-xml/
3. Emek, Y., Halldórsson, M.M., Mansour, Y., Patt-Shamir, B., Radhakrishnan, J., Rawitz, D.: Online set packing and competitive scheduling of multi-part tasks. In: 29th Annual ACM Symposium on Principles of Distributed Computing (2010)
4. Mansour, Y., Patt-Shamir, B., Rawitz, D.: Overflow management with multipart packets. In: IEEE INFOCOM (2011)
5. Mansour, Y., Patt-Shamir, B., Lapid, O.: Optimal smoothing schedules for real-time streams. In: 19th Annual ACM Symposium on Principles of Distributed Computing, pp. 21–30 (2000)
6. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. In: 33rd Annual ACM Symposium on Theory of Computing, pp. 520–529 (2001)
7. Goldwasser, M.H.: A survey of buffer management policies for packet switches. SIGACT News 41(1), 100–128 (2010)
8. Kesselman, A., Patt-Shamir, B., Scalosub, G.: Competitive buffer management with packet dependencies. In: 23rd IPDPS, pp. 1–12 (2009)
9. Alon, N., Spencer, J.H.: The Probabilistic Method. 3rd edn. Wiley Interscience (2008)
10. Håstad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. Acta Mathematica 182(1), 105–142 (1999)
11. Halldórsson, M.M., Kratochvíl, J., Telle, J.A.: Independent sets with domination constraints. Discrete Applied Mathematics 99(1-3), 39–54 (2000)
12. Hurkens, C.A.J., Schrijver, A.: On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. SIAM J. Discrete Math. 2(1), 68–72 (1989)
13. Berman, P.: A $d/2$ approximation for maximum weight independent set in d-claw free graphs. Nord. J. Comput. 7(3), 178–184 (2000)
14. Hazan, E., Safra, S., Schwartz, O.: On the Complexity of Approximating k-Dimensional Matching. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 83–97. Springer, Heidelberg (2003)

Approximation with a Fixed Number of Solutions of Some Biobjective Maximization Problems*

Cristina Bazgan[1,2,3], Laurent Gourvès[1,2], and Jérôme Monnot[1,2]

[1] Université Paris-Dauphine, LAMSADE,
Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France
[2] CNRS, UMR 7243
[3] Institut Universitaire de France
{bazgan,laurent.gourves,monnot}@lamsade.dauphine.fr

Abstract. We investigate the problem of approximating the Pareto set of biobjective optimization problems with a given number of solutions. This task is relevant for two reasons: (*i*) Pareto sets are often computationally hard so approximation is a necessary tradeoff to allow polynomial time algorithms; (*ii*) limiting explicitly the size of the approximation allows the decision maker to control the expected accuracy of approximation and prevents him to be overwhelmed with too many alternatives. Our purpose is to exploit general properties that many well studied problems satisfy. We derive existence and constructive approximation results for the biobjective versions of Max Bisection, Max Partition, Max Set Splitting and Max Matching.

1 Introduction

In multiobjective combinatorial optimization a solution is evaluated considering several objective functions and a major challenge in this context is to generate the set of efficient solutions or the Pareto set (see [8] about multiobjective combinatorial optimization). However, it is usually difficult to identify the efficient set mainly due to the fact that the number of efficient solutions can be exponential in the size of the input and moreover the associated decision problem is NP-complete even if the underlying single-objective problem can be solved in polynomial time. To handle these two difficulties, researchers have been interested in developing approximation algorithms with an a priori provable guarantee such as polynomial time constant approximation algorithms. Considering that all objectives have to be maximized, and for a positive $\rho \leq 1$, a ρ-approximation of Pareto set is a set of solutions that includes, for each efficient solution, a solution that is at least at a factor ρ on all objective values. Intuitively, the larger the size of the approximation set, the more accurate it can be.

* This research has been supported by the project ANR-09-BLAN-0361 GUaranteed Efficiency for PAReto optimal solutions Determination (GUEPARD).

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 233–246, 2012.

It has been pointed out by Papadimitriou and Yannakakis [18] that, under certain general assumptions, there always exists a $(1-\varepsilon)$-approximation, with any given accuracy $\varepsilon > 0$, whose size is polynomial both in the size of the instance and in $1/\varepsilon$ but exponential in the number of criteria. In this result, the accuracy $\varepsilon > 0$ is given explicitly but the size of the approximation set is not given explicitly. When the number of solutions in the approximation set is limited, not every level of accuracy is possible. So, once the number of solutions is fixed in the approximation set of a multiobjective problem, the following questions are raised: What is the accuracy for which an approximation is guarantee to exist? Which accuracy can be obtained in polynomial time?

In this paper we are interested in establishing for biobjective maximization problems the best approximation ratio of the set of efficient solutions when the size of the approximation set is given explicitly. We give two approaches that deal with biobjective problems that allow us to obtain approximations of the set of efficient solutions with one or several solutions. More precisely, in a first approach, we consider a general maximization problem (denoted by Π_1 in the following) and establish a sufficient condition that guarantees the construction of a constant approximation of the Pareto set with an explicitly given number of solutions. As a corollary, we can construct a $(1-\varepsilon)$-approximation of the Pareto set with $O(\frac{1}{\varepsilon})$ solutions. In a second approach, we establish a necessary and sufficient condition for the construction of a constant approximation of the Pareto set with one solution.

Properties defined in these two approaches apply to several problems previously studied in single-objective approximation. Then we derive polynomial time constant approximations with one solution for Biobjective Max Bisection, Biobjective Max Partition, Biobjective Max Cut, Biobjective Max Set Splitting, Biobjective Max Matching. Some instances show that the given biobjective appromixation ratios are the best we can expect. In addition Biobjective Max Partition, Biobjective Max Cut, Biobjective Max Set Splitting admit a $(1-\varepsilon)$-approximation of the Pareto set with $O(\frac{1}{\varepsilon})$ solutions.

Several results exist in the literature on the approximation of multiobjective combinatorial optimization problems. One can mention the existence of fully polynomial time approximation schemes for biobjective shortest path [12,22,21], knapsack [9,5], minimum spanning tree [18], scheduling problems [4], randomized fully polynomial time approximation scheme for matching [18], and polynomial time constant approximation for max cut [2], a biobjective scheduling problem [20] and the traveling salesman problem [3,16]. Note that [2] and [20] are approximations with a single solution.

This article is organized as follows. In Section 2, we introduce basic concepts about multiobjective optimization and approximation. Section 3 is devoted to an approach for approximating some biobjective problems with one or several solutions. Section 4 presents a necessary and sufficient condition for approximating within a constant factor some biobjective problems with one solution. Conclusions are provided in a final section. Due to space limitation, some proofs are omitted.

2 Preliminaries on Multi-objective Optimization and Approximation

Consider an instance of a multi-objective optimization problem with k criteria or objectives where X denotes the finite set of feasible solutions. Each solution $x \in X$ is represented in the objective space by its corresponding objective vector $w(x) = (w_1(x), \ldots, w_k(x))$. We assume that each objective has to be maximized.

From these k objectives, the dominance relation defined on X states that a feasible solution x dominates a feasible solution x' if and only if $w_i(x) \geq w_i(x')$ for $i = 1, \ldots, k$ with at least one strict inequality. A solution x is *efficient* if and only if there is no other feasible solution $x' \in X$ such that x' dominates x, and its corresponding objective vector is said to be *non-dominated*. Usually, we are interested in finding a solution corresponding to each non-dominated objective vector, set that is called Pareto set.

For any $0 < \rho \leq 1$, a solution x is called a ρ-approximation of a solution x' if $w_i(x) \geq \rho \cdot w_i(x')$ for $i = 1, \ldots, k$. A set of feasible solutions X' is called a ρ-approximation of a set of efficient solutions if, for every feasible solution $x \in X$, X' contains a feasible solution x' that is a ρ-approximation of x. If such a set exists, we say that the multi-objective problem admits a ρ-approximate Pareto set with $|X'|$ solutions.

An algorithm that outputs a ρ-approximation of a set of efficient solutions in polynomial time in the size of the input is called a ρ-approximation algorithm. In this case we say that the multi-objective problem admits a polynomial time ρ-approximate Pareto set.

Consider in the following a single-objective maximization problem P defined on a ground set $\mathcal{U}$. Every element $e \in \mathcal{U}$ has a non negative weight $w(e)$. The goal is to find a feasible solution (subset of $\mathcal{U}$) with maximum weight. The weight of a solution S must satisfy the following scaling hypothesis: if $opt(I)$ denotes the optimum value of I, then $opt(I') = t \cdot opt(I)$, where I' is the same instance as I except that $w'(e) = t \cdot w(e)$. For example, the hypothesis holds when the weight of S is defined as the sum of its elements' weights, or $\min w(e) : e \in S$, etc.

In the biobjective version, called biobjective P, every element $e \in \mathcal{U}$ has two non negative weights $w_1(e), w_2(e)$ and the goal is to find a Pareto set within the set of feasible solutions. Given an instance I of biobjective P, we denote by $opt_i(I)$ (or simply opt_i) the optimum value of I restricted to objective i, $i = 1, 2$. Here, the objective function on objective 1 is not necessarily the same as on objective 2, but both satisfy the scaling hypothesis.

3 Approximation with a Given Number of Solutions

Papadimitriou and Yannakakis [18] proved the existence of at least one $(1 - \varepsilon)$-approximation of size polynomial in the size of the instance and $\frac{1}{\varepsilon}$. In this result, the accuracy $\varepsilon > 0$ is given explicitly but the size of the approximation set is not given explicitly. In this section we consider a general maximization problem Π_1 and establish a sufficient condition that guarantees the construction of

a constant approximation of the Pareto set with an explicitly given number of solutions for Π_1. This result allows to construct a $(1-\varepsilon)$-approximation of the Pareto set with $O(\frac{1}{\varepsilon})$ solutions but not necessarily in polynomial time. Moreover, if the single objective problem is polynomial time constant approximable and the sufficient condition is strengthened then the biobjective version is also polynomial time constant approximable with one solution. Thus we obtain constant approximations and polynomial time constant approximations with one solution for Biobjective Max Partition, Biobjective Max Cut, Biobjective Max Set Splitting, Biobjective Max Matching.

In the following, we are interested in particular cases of biobjective maximization problems, Biobjective Π_1, which satisfy the following property.

Property 1. Given any two feasible solutions S_1 and S_2, and any real α satisfying $0 < \alpha \leq 1$, if $w_2(S_1) < \alpha w_2(S_2)$ and $w_1(S_2) < \alpha w_1(S_1)$ then there exists a feasible solution S_3 which satisfies $w_1(S_3) > (1-\alpha)w_1(S_1)$ and $w_2(S_3) > (1-\alpha)w_2(S_2)$.

We say that *Biobjective Π_1 satisfies polynomially Property 1* if S_3 can be constructed in polynomial time.

Property 1 means that if S_1 is not an α-approximation of S_2 and S_2 is not an α-approximation of S_1 for both objective functions w_1 and w_2, then there exists a feasible solution S_3 which simultaneously approximates S_1 and S_2 with performance guarantee $1-\alpha$.

Given a positive integer ℓ, consider the equations $x^{2\ell} = 1 - x^{\ell}$ and $x^{2\ell-1} = 1 - x^{\ell}$. Denote by α_ℓ and β_ℓ their respective solutions in the interval $[0,1)$. Remark that $\alpha_\ell = \left(\frac{\sqrt{5}-1}{2}\right)^{1/\ell}$ and $\alpha_\ell < \beta_{\ell+1} < \alpha_{\ell+1}, \ell \geq 1$.

Theorem 1. *If Biobjective Π_1 satisfies Property 1, then it admits a β_ℓ-approximate Pareto set (resp. an α_ℓ-approximate Pareto set) containing at most p solutions, where p is a positive odd integer such that $p = 2\ell - 1$ (resp. a positive even integer such that $p = 2\ell$).*

Proof. Let S_1 (resp. S_2) be a solution optimal for the first objective (resp. second one). In the following, *opt* denotes the optimal value on the first objective and also on the second objective. This can be assumed without loss of generality because a simple rescaling can make the optimal values coincide (e.g. we can always assume that $opt_2 \neq 0$, thus by multiplying each weight $w_2(e)$ by $\frac{opt_1}{opt_2}$ we are done). Then $w_1(S_1) = w_2(S_2) = opt$. If p is odd then $\rho = \beta_\ell$ with $p = 2\ell - 1$, otherwise $\rho = \alpha_\ell$ with $p = 2\ell$. Subdivide the bidimensionnal value space with coordinates $\{0\} \cup \{\rho^i opt : 0 \leq i \leq p\}$. See Figure 1 for an illustration.

Given $i, 1 \leq i \leq p$, the *strip* $s(i,.)$ is the part of the space containing all couples (w_1, w_2) satisfying $\rho^i opt < w_1 \leq \rho^{i-1} opt$ and $0 \leq w_2 \leq opt$. The strip $s(p+1,.)$ is the part of the space containing all couples (w_1, w_2) satisfying $0 \leq w_1 \leq \rho^p opt$ and $0 \leq w_2 \leq opt$. Given $j, 1 \leq j \leq p$, the *strip* $s(.,j)$ is the part of the space containing all couples (w_1, w_2) satisfying $\rho^j opt < w_2 \leq \rho^{j-1} opt$ and

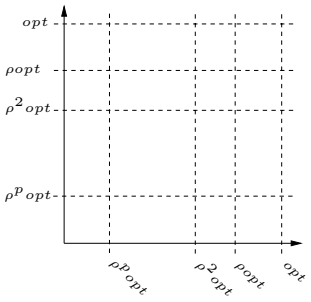


Fig. 1. Illustration of Theorem 1

$0 \leq w_1 \leq opt$. The strip $s(.,p+1)$ is the part of the space containing all couples (w_1, w_2) satisfying $0 \leq w_2 \leq \rho^p opt$ and $0 \leq w_1 \leq opt$.

Suppose that $w_2(S_1) < \rho^p opt$ and $w_1(S_2) < \rho^p opt$. In other words $S_1 \in s(1,.) \cap s(.,p+1)$ and $S_2 \in s(.,1) \cap s(p+1,.)$. Using Property 1 there exists a solution S_3 satisfying $w_1(S_3) > (1-\rho^p)opt$ and $w_2(S_3) > (1-\rho^p)opt$. For the case $\rho = \beta_\ell$ and $p = 2\ell - 1$, we get that $1 - \rho^p = 1 - \beta_\ell^{2\ell-1} = \beta_\ell^\ell = \rho^\ell$. For the case $\rho = \alpha_\ell$ and $p = 2\ell$, we get that $1 - \rho^p = 1 - \alpha_\ell^{2\ell} = \alpha_\ell^\ell = \rho^\ell$. Then S_3 is a ρ-approximation of any solution S satisfying $\max\{w_1(S), w_2(S)\} \leq \rho^{\ell-1} opt$.

One can construct a ρ-approximate Pareto set P as follows: $P = \{S_3\}$ at the beginning and for $j = \ell - 1$ down to 1, pick a feasible solution s with maximum weight w_1 in $s(.,j)$ (if $s(.,j)$ contains at least one value of a feasible solution) and set $P = P \cup \{S\}$. Afterwards, for $i = \ell - 1$ down to 1, pick a feasible solution S with maximum weight w_2 in $s(i,.)$ (if $s(i,.)$ contains at least one value of a feasible solution) and set $P = P \cup \{S\}$. For every strip the algorithm selects a solution which ρ-approximates (on both objective functions) any other solution in the strip. Since the solutions of P approximate the whole bidimensionnal space, P is a ρ-approximate Pareto set containing at most $2\ell - 1$ solutions. Here $2\ell - 1$ is equal to p when p is odd, otherwise it is equal to $p - 1$.

Now suppose that $w_2(S_1) \geq \rho^p opt$ (the case $w_1(S_2) \geq \rho^p opt$ is treated similarly). Solution S_1 must be in $s(.,j^*)$ for $1 \leq j^* \leq p$. Since $w_1(S_1) = opt$, S_1 is a ρ-approximation of any solution S in $s(.,p) \cup s(.,p+1)$. One can build an ρ-approximate Pareto set P as follows: $P = \{S_1\}$ at the beginning and for $j = j^* - 1$ down to 1, pick a feasible solution S with maximum weight w_1 in $s(.,j)$ (if $s(.,j)$ contains at least one value of a feasible solution) and set $P = P \cup \{S\}$. Since the strips form a partition of the space, the algorithm returns an ρ-approximate Pareto set containing at most p solutions. □

Corollary 1. *If Biobjective Π_1 satisfies Property 1, then it admits a $(1-\varepsilon)$-approximate Pareto set containing $O(\frac{1}{\varepsilon})$ solutions.*

Property 1 can be relaxed in the following way:

Property 2. We are given two feasible solutions S_1 and S_2, and a real α satisfying $0 < \alpha \leq 1$. If $w_2(S_1) < \alpha w_2(S_2)$ and $w_1(S_2) < \alpha w_1(S_1)$ then there exists

a feasible solution S_3 which satisfies $w_1(S_3) > (c-\alpha)w_1(S_1)$ and $w_2(S_3) > (c-\alpha)w_2(S_2)$, where $0 < c \leq 1$.

We define similarly that Biobjective Π_1 satisfies polynomially Property 2.

Given a positive integer ℓ, consider the equations $x^{2\ell} = c - x^{\ell}$ and $x^{2\ell-1} = c - x^{\ell}$. Denote by γ_ℓ and δ_ℓ their respective solutions in the interval $[0, 1)$. Remark that $\gamma_\ell = (\frac{\sqrt{1+4c}-1}{2})^{1/\ell}$ and $\gamma_\ell < \delta_\ell < \gamma_{\ell+1}, \ell \geq 1$.

Theorem 2. *If Biobjective Π_1 satisfies Property 2, then it admits a δ_ℓ-approximate Pareto set (resp. an γ_ℓ-approximate Pareto set) containing at most p solutions, where p is a positive odd integer such that $p = 2\ell - 1$ (resp. a positive even integer such that $p = 2\ell$).*

Proof. The proof is similar with the proof of Theorem 1. Suppose that $w_2(S_1) < \rho^p opt$ and $w_1(S_2) < \rho^p opt$. Using Property 2 there exists a solution S_3 satisfying $w_1(S_3) > (c-\rho^p)opt$ and $w_2(S_3) > (c-\rho^p)opt$. For the case $\rho = \delta_\ell$ and $p = 2\ell-1$, we get that $c - \rho^p = c - \delta_\ell^{2\ell-1} = \delta_\ell^{\ell} = \rho^{\ell}$. For the case $\rho = \gamma_\ell$ and $p = 2\ell$, we get that $c - \rho^p = c - \gamma_\ell^{2\ell} = \gamma_\ell^{\ell} = \rho^{\ell}$. Then S_3 is a ρ-approximation of any solution S satisfying $\max\{w_1(S), w_2(S)\} \leq \rho^{\ell-1} opt$. □

The previous results of this section consider the construction, not necessarily in polynomial time, of an approximate Pareto set with a fixed number of solutions. We give in the following some conditions on the construction in polynomial time of an approximate Pareto set with one solution.

Proposition 1. *If Π_1 is polynomial time ρ-approximable and Biobjective Π_1 satisfies polynomially Property 1 (resp. 2), then Biobjective Π_1 is polynomial time $\frac{\rho}{2}$-approximable (resp. $\frac{c\rho}{2}$-approximable) with one solution.*

Proof. Let S_1 (resp. S_2) be a polynomial time ρ-approximation solution for the first objective (resp. second one). In the following, opt_1 (resp. opt_2) denotes the optimal value on the first objective (resp. second one). If $w_2(S_1) \geq \frac{w_2(S_2)}{2}$ then $w_2(S_1) \geq \frac{\rho}{2} opt_2$ and thus S_1 is a $\frac{\rho}{2}$-approximate Pareto set. If $w_1(S_2) \geq \frac{w_1(S_1)}{2}$ then $w_1(S_2) \geq \frac{\rho}{2} opt_1$ and thus S_2 is a $\frac{\rho}{2}$-approximate Pareto set. Otherwise, $w_2(S_1) < \frac{w_2(S_2)}{2}$ and $w_1(S_2) < \frac{w_1(S_1)}{2}$ and since Biobjective Π_2 satisfies polynomially Property 1, we can construct in polynomial time a feasible solution S_3 which satisfies $w_1(S_3) \geq \frac{w_1(S_1)}{2}$ and $w_2(S_3) \geq \frac{w_2(S_2)}{2}$, that is a $\frac{\rho}{2}$-approximate Pareto set. □

We consider in Sections 3.1, 3.2, and 3.3 several examples of problems Π_1 that satisfy the scaling hypothesis and such that Biobjective Π_1 satisfy Property 1 or Property 2.

3.1 Max Pos NAE

The Max Pos NAE problem consists of a set of clauses $\mathcal{C}$ defined on a set of boolean variables $x_1, \ldots, x_n$. The clauses are composed of two or more positive

variables and they are endowed with a non negative weight. The Max Pos NAE problem consists of finding an assignment of the variables such that the total weight of the clauses that are satisfied is maximum, where a positive clause is satisfied by an assignment if it contains at least a true variable and at least a false variable. Max Pos NAE generalizes Max Cut and so it is NP-hard and 0.7499-approximable [24]. Max Pos NAE is also known under the name Max Set Splitting or Max Hypergraph Cut [24].

Lemma 1. *Biobjective* Max Pos NAE *satisfies polynomially Property 1.*

Proof. Let $\alpha \in (0,1]$ and S_1, S_2 two solutions of an instance of biobjective Max Pos NAE satisfying the inequalities: $w_2(S_1) < \alpha w_2(S_2)$ and $w_1(S_2) < \alpha w_1(S_1)$. Consider $S_3 = (S_1 \setminus S_2) \cup (S_2 \setminus S_1)$. Let $c(S)$ be the set of clauses satisfied by assigning variables from S to true and those from $\bar{S}$ to false. Clearly $c(S) = \{C_i = x_{i_1} \vee \ldots \vee x_{i_t} : \exists x_{i_j} \in S, \exists x_{i_\ell} \in \bar{S}\}$. In the following a clause C_i is identified by the set of variables that it contains $\{x_{i_1}, \ldots, x_{i_t}\}$. Then $c(S_1) \setminus c(S_2) = \{C : C \cap S_1 \neq \emptyset \text{ and } C \cap \bar{S}_1 \neq \emptyset\} \cap \{C : C \subseteq S_2 \text{ or } C \subseteq \bar{S}_2\}$. Let $C \in c(S_1) \setminus c(S_2)$. If $C \subseteq S_2$ then since $C \cap \bar{S}_1 \neq \emptyset$ we have $\emptyset \neq C \cap (S_2 \setminus S_1) \subseteq C \cap S_3$. Moreover $C \cap \bar{S}_3 \neq \emptyset$ since $C \cap S_1 \cap S_2 \neq \emptyset$. Thus $C \in c(S_3)$. If $C \subseteq \bar{S}_2$ then since $C \cap S_1 \neq \emptyset$ we have $\emptyset \neq C \cap (S_1 \setminus S_2) \subseteq C \cap S_3$. Moreover $C \cap \bar{S}_3 \neq \emptyset$ since $C \cap \bar{S}_3 \subseteq C \cap \bar{S}_1 \cap \bar{S}_2 \neq \emptyset$. Thus $c(S_1) \setminus c(S_2) \subseteq C(S_3)$. In the similar way we can prove $c(S_2) \setminus c(S_1) \subseteq C(S_3)$. Thus, $c(S_1) \Delta c(S_2) = \big(c(S_1) \setminus c(S_2)\big) \cup \big(c(S_2) \setminus c(S_1)\big)$ is contained in $c(S_3)$.

The inequality $w_2(S_1) < \alpha w_2(S_2)$ can be rewritten as follows:

$$\sum_{C \in c(S_1)} w_2(C) < \alpha \sum_{C \in c(S_2)} w_2(C)$$

$$\sum_{C \in c(S_1) \setminus c(S_2)} w_2(C) + (1-\alpha) \sum_{C \in c(S_1) \cap c(S_2)} w_2(C) < \alpha \sum_{C \in c(S_2) \setminus c(S_1)} w_2(C)$$

We can use it to get

$$w_2(S_3) \geq \sum_{C \in c(S_1) \setminus c(S_2)} w_2(C) + \sum_{C \in c(S_2) \setminus c(S_1)} w_2(C) =$$

$$= \sum_{C \in c(S_1) \setminus c(S_2)} w_2(C) + \alpha \sum_{C \in c(S_2) \setminus c(S_1)} w_2(C) + (1-\alpha) \sum_{C \in c(S_2) \setminus c(S_1)} w_2(C) >$$

$$> 2 \sum_{C \in c(S_1) \setminus c(S_2)} w_2(C) + (1-\alpha) \sum_{C \in c(S_1) \cap c(S_2)} w_2(C) + (1-\alpha) \sum_{C \in c(S_2) \setminus c(S_1)} w_2(C) \geq$$

$$\geq (1-\alpha) \sum_{C \in c(S_2)} w_2(C) = (1-\alpha) w_2(S_2).$$

Using the same technique we can show that $w_1(S_3) > (1-\alpha) w_1(S_1)$. □

Corollary 2. *Biobjective* Max Pos NAE *admits a*
(i) β_ℓ*-approximate Pareto set (resp. an* α_ℓ*-approximate Pareto set) containing at most* p *solutions, where* $p = 2\ell - 1$ *(resp.* $p = 2\ell$*).*
(ii) $(1-\varepsilon)$*-approximate Pareto set containing* $O(\frac{1}{\varepsilon})$ *solutions.*

As indicated above, Corollary 2 deals with the possibility to reach some approximation bounds when the number of solutions in the Pareto set is fixed. We give in the following an approximation bound that we can obtain in polynomial time with one solution.

Corollary 3. *Biobjective* Max Pos NAE *admits a polynomial time* 0.374*-approximate Pareto set with one solution.*

Proof. The results follows from Lemma 1 and Proposition 1 with $\rho = 0.7499$. □

We consider in the following a particular case of Max Pos NAE in which every clause contains exactly k variables, denoted Max Pos kNAE. Max Pos 3NAE is 0.908-approximable [25]. For $k \geq 4$, Max Pos kNAE is $(1 - 2^{1-k})$-approximable [1,14] and this is the best possible since it is hard to approximate within a factor of $1 - 2^{1-k} + \varepsilon$, for any constant $\varepsilon > 0$ [13].

Corollary 4. *Biobjective* Max Pos 3NAE *admits a polynomial time* 0.454*-approximate Pareto set with one solution. For* $k \geq 4$, Max Pos kNAE *admits a polynomial time* $1/2 - 2^{-k}$*-approximate Pareto set with one solution.*

Proof. The results follows from Lemma 1 and Proposition 1 with $\rho = 0.908$ and $\rho = 1 - 2^{1-k}$. □

We consider in the following another particular case of Max Pos NAE in which every clause contains exactly 2 variables, that is exactly Max Cut which is 0.878-approximable [10].

Corollary 5. *Biobjective* Max Cut *admits a*
(i) β_ℓ*-approximate Pareto set (resp. an* α_ℓ*-approximate Pareto set) containing at most* p *solutions, where* $p = 2\ell - 1$ *(resp.* $p = 2\ell$*).*
(ii) $(1-\varepsilon)$*-approximate Pareto set containing* $O(\frac{1}{\varepsilon})$ *solutions.*

Corollary 6. *Biobjective* Max Cut *admits a polynomial time* 0.439*-approximate Pareto set with one solution.*

Proof. The results follows from Lemma 1 and Proposition 1 with $\rho = 0.878$ [10]. □

Clearly this last result is the same as the one given in [2] but we use a different method. We remark that Biobjective Max Cut is not $(1/2 + \varepsilon)$-approximable with one solution [2], meaning that we are close to the best possible approximation result.

3.2 Max Partition

The Max Partition problem is defined as follows: given a set J of n items $1, \ldots, n$, each item j of positive weight $w(j)$, find a solution S that is a bipartition $J_1 \cup J_2$ of the n items such that $w(S) = \min\{\sum_{j \in J_1} w(j), \sum_{j \in J_2} w(j)\}$ is maximized. This NP-hard problem was also studied in the context of scheduling, where the number of partitions is not fixed, and consists of maximizing the earliest machine completion time [23].

Lemma 2. *Biobjective* Max Partition *satisfies polynomially Property 7.*

Corollary 7. *Biobjective* Max Partition *admits a*
(i) β_ℓ-approximate Pareto set (resp. an α_ℓ-approximate Pareto set) containing at most p solutions, where $p = 2\ell - 1$ (resp. $p = 2\ell$).
(ii) $(1-\varepsilon)$-approximate Pareto set containing $O(\frac{1}{\varepsilon})$ solutions.

Corollary 8. *Biobjective* Max Partition *admits a polynomial time $(1/2-\varepsilon)$-approximate Pareto set with one solution, for every $\varepsilon > 0$.*

Proof. Max Partition is a particular case of the Max Subset Sum problem. An input of Max Subset Sum is formed by a set J of n items $1, \ldots, n$, each item j has a positive weight $w(j)$, and an integer t. The problem consists of finding a subset S of J whose sum $w(S)$ is bounded by t and maximum. Max Subset Sum has a fptas [6]. We can obtain a fptas for Max Partition using the previous fptas for $t = \sum_{i=1}^{n} w(i)/2$.

The results follows from Lemma 2 and Proposition 1 with $\rho = 1 - 2\varepsilon$. □

Observe that Biobjective Max Partition is not $(1/2 + \varepsilon)$-approximable with one solution. In order to see this, consider 3 items of weights $w_1(1) = 2, w_2(1) = 1, w_1(2) = 1, w_2(2) = 2, w_1(1) = 1, w_2(3) = 1$. The two efficient solutions S_i, $i = 1, 2$ consists of placing i in a part and the other items in the other part and have weights $w_1(S_1) = 2, w_2(S_1) = 1, w_1(S_2) = 1, w_2(S_2) = 2$. Any other solution is either dominated by one of these two or has weights equal to 1 on both criteria.

3.3 Max Matching

Given a complete graph $G = (V, E)$ with non negative weights on the edges, the Max Matching problem is to find a matching of the graph of total weight maximum. Max Matching is solvable in polynomial time [7]. We study in this part the biobjective Max Matching problem and consider instances where the graph is a collection of complete graphs inside which the weights satisfy the triangle inequality, since otherwise the biobjective Max Matching problem is not at all approximable with one solution. In order to see this, consider a complete graph on 3 vertices with weights $(1, 0), (0, 1), (0, 0)$. The optimum value on each objective is 1. Nevertheless, any solution has value 0 on at least one objective. Clearly Property 1 is not satisfied in this case.

Biobjective Max Matching problem is NP-hard [19]. It remains NP-hard even on instances where the graph is a collection of complete graphs inside which the weights satisfy the triangle inequality.

Lemma 3. *Biobjective* Max Matching *satisfies polynomially Property 2 with* $c = 1/3$.

Corollary 9. *Biobjective* Max Matching *admits a* δ_ℓ*-approximate Pareto set (resp. an* γ_ℓ*-approximate Pareto set) containing at most* p *solutions, where* $p = 2\ell - 1$ *(resp.* $p = 2\ell$*).*

Corollary 10. *Biobjective* Max Matching *admits a polynomial time* $\frac{1}{6}$*-approximate Pareto set with one solution.*

Proof. It follows from Lemma 3 and Proposition 1 considering $\rho = 1$. □

4 Approximation with One Solution

In this section, we establish a necessary and sufficient condition for constructing, not necessarily in polynomial time, a constant approximation with one solution of the Pareto set for biobjective maximization problems. Moreover, if the condition is strengthened and the single-objective problem is polynomial time constant approximable, then the biobjective version is polynomial time constant approximable with one solution. Thus, using this condition, we establish a polynomial time 0.174-approximation with one solution for Biobjective Max Bisection.

In the following, we are interested in particular cases of biobjective maximization problems, Biobjective Π_2 which satisfy the following property.

Property 3. We can construct three solutions S_1, S_2, S_3 such that S_i is a ρ_i-approximation for problem Π_2 on objective i, $i = 1, 2$, and S_3 is such that $w_1(S_2) + w_1(S_3) \geq \alpha \cdot w_1(S_1)$ and $w_2(S_1) + w_2(S_3) \geq \alpha \cdot w_2(S_2)$ for some $\alpha \leq 1$.

We say that *Biobjective* Π_2 *satisfies polynomially Property 3* if S_1, S_2, S_3 can be constructed in polynomial time.

The aim of solution S_3 in Property 3 is to compensate the potential inefficiency of S_i on criterion $3 - i$, $i = 1, 2$.

Theorem 3. *Biobjective* Π_2 *is (resp. polynomial time) constant approximable with one solution if and only if it satisfies (resp. polynomially) Property 3. More precisely, if Biobjective* Π_2 *satisfies polynomially Property 3 such that* S_i *is a polynomial time* ρ_i*-approximation for problem* Π_2 *on objective* i*,* $i = 1, 2$*, then Biobjective* Π_2 *admits a polynomial time* $\alpha\frac{\min\{\rho_1,\rho_2\}}{2}$*-approximation algorithm with one solution.*

Proof. Suppose that Biobjective Π_2 is ρ-approximable with one solution. Let S_3 be this solution and S_1 and S_2 any two solutions. Then $w_1(S_3) \geq \rho \cdot opt_1 \geq$

$\rho \cdot w_1(S_1)$ and thus by setting $\alpha = \rho$ we have $w_1(S_2) + w_1(S_3) \geq \alpha \cdot w_1(S_1)$. The second inequality holds also.

Suppose now that Biobjective Π_2 satisfies Property 3. Since S_i is a ρ_i-approximation for problem Π_2 on objective i, $i = 1, 2$, we have $w_1(S_1) \geq \rho_1 \cdot opt_1$ and $w_2(S_2) \geq \rho_2 \cdot opt_2$.

Since Property 3 is satisfied, we can construct S_3 such that

$$w_1(S_2) + w_1(S_3) \geq \alpha \cdot w_1(S_1) \tag{1}$$

and

$$w_2(S_1) + w_2(S_3) \geq \alpha \cdot w_2(S_2) \tag{2}$$

Now, we study different cases:

- If $w_1(S_2) \geq \frac{\alpha}{2} w_1(S_1)$, then we deduce that S_2 is a good approximation of the Pareto set. From the hypothesis, we have $w_1(S_2) \geq \frac{\alpha}{2} w_1(S_1) \geq \alpha \cdot \frac{\min\{\rho_1, \rho_2\}}{2} opt_1$. On the other hand, we also have $w_2(S_2) \geq \rho_2 \cdot opt_2 \geq \alpha \frac{\min\{\rho_1, \rho_2\}}{2} opt_2$.
- If $w_2(S_1) \geq \frac{\alpha}{2} w_2(S_2)$, then we deduce that S_1 is a good approximation of the Pareto set. From the hypothesis, we have $w_2(S_1) \geq \frac{\alpha}{2} w_2(S_2) \geq \alpha \cdot \frac{\min\{\rho_1, \rho_2\}}{2} opt_2$. On the other hand, by the construction of S_1 we also have $w_1(S_1) \geq \rho_1 \cdot opt_1 \geq \alpha \cdot \frac{\min\{\rho_1, \rho_2\}}{2} opt_1$.
- If $w_1(S_2) \leq \frac{\alpha}{2} w_1(S_1)$ and $w_2(S_1) \leq \frac{\alpha}{2} w_2(S_2)$, then it is S_3 which is a good approximation of the Pareto set. Indeed, from inequality (1), we deduce $w_1(S_3) \geq \frac{\alpha}{2} w_1(S_1) \geq \alpha \cdot \frac{\min\{\rho_1, \rho_2\}}{2} opt_1$ and on the other hand, from inequality (2), we also get $w_2(S_3) \geq \frac{\alpha}{2} w_2(S_2) \geq \alpha \cdot \frac{\min\{\rho_1, \rho_2\}}{2} opt_2$.

In any of these three cases, we obtain a $\alpha \cdot \frac{\min\{\rho_1, \rho_2\}}{2}$-approximation with one solution.

Clearly, if S_1, S_2, S_3 are computable in polynomial time, then Biobjective Π_2 is approximable in polynomial time. □

Remark that we can extend Theorem 3 to the case where ρ_i are not constant.

The interest of Property 3 is to find a simple method in order to construct a polynomial time constant approximation for Biobjective Π_2. This method does not allow us to obtain the best polynomial time constant approximation for Biobjective Π_2 with one solution, but only to prove the fact that the problem is polynomial time constant approximable with one solution.

In Lemma 1 we prove that if a problem Π is (resp. polynomial time) constant approximable and if Biobjective Π satisfies (resp. polynomially) Property 1, then Biobjective Π is (resp. polynomial time) constant approximable with one solution, and thus Biobjective Π satisfies (resp. polynomially) Property 3 by Theorem 3. Thus all problems studied in Section 3 satisfies Property 3.

There exist problems which are polynomial time constant approximable and thus satisfy Property 3 and do not satisfy Property 1. One example is Biobjective TSP, which is polynomial time $\frac{7}{27}$-approximable with one solution [16,17] and does not satisfy Property 1.

Proposition 2. *Biobjective TSP does not satisfy Property 1.*

Proof. Consider the complete graph K_5 where a fixed K_4 is decomposable into 2 Hamiltonian paths P_1 and P_2. For every edge $e \in E(K_5)$, set $w_1(e) = 1$ and $w_2(e) = 0$ if $e \in P_1$, $w_1(e) = 0$ and $w_2(e) = 1$ if $e \in P_2$ and $w_1(e) = 0$ and $w_2(e) = 0$ if $e \notin P_1 \cup P_2$. We can check that there are four non-dominated tours T_i, $i = 1, \ldots, 4$ with $w_1(T_1) = 3$, $w_2(T_1) = 0$, $w_1(T_2) = 0$, $w_2(T_2) = 3$, $w_1(T_3) = 2$, $w_2(T_3) = 1$ and $w_1(T_4) = 1$, $w_2(T_4) = 2$. Consider $S_i = T_i$, $i = 1, 2$ and $\alpha = 1/2$. Clearly $w_2(S_1) < \alpha w_2(S_2)$ and $w_1(S_2) < \alpha w_1(S_1)$. Moreover there is no solution S_3 such that $w_1(S_3) > (1-\alpha)w_1(S_1)$ and $w_2(S_3) > (1-\alpha)w_2(S_2)$. □

We consider in the following a problem that satisfies Property 3 and for which we are not able to prove that it satisfies Property 1.

4.1 Max Bisection

Given a graph $G = (V, E)$ with non negative weights on the edges, the Max Bisection problem consists of finding a bipartition of the vertex set V into two sets of equal size such that the total weight of the cut is maximum. We establish in this part a polynomial time $\frac{\rho}{4}$-approximation algorithm for Biobjective Max Bisection where ρ is any polynomial time approximation ratio given for Max Bisection. Max Bisection is NP-hard [15] and the best approximation ratio known for Max Bisection is $\rho = 0.701$ [11].

Lemma 4. *Biobjective* Max Bisection *satisfies polynomially Property 3 with* $\alpha = 1$ *and* $\rho_1 = \rho$ *and* $\rho_2 = \frac{\rho}{2}$, *where* ρ *is any polynomial time approximation ratio given for* Max Bisection.

Corollary 11. *Biobjective* Max Bisection *admits a polynomial time* 0.174*-approximate Pareto set with one solution.*

Proof. The results follows from Theorem 3 and Lemma 4 and using the polynomial time 0.701-approximation algorithm for Max Bisection [11]. □

5 Conclusion

In this paper, we established some sufficient conditions that allow to conclude on the existence of constant approximations of the Pareto set with an explicitly given number of solutions for several biobjective maximization problems. The results we obtained establish a *polynomial time* approximation when we ask for a single solution in the approximation set. A possible future work would be to give a polynomial time approximation for any explicitly given number of solutions. A necessary and sufficient condition is given for the construction of (polynomial time) constant approximation with one solution for biobjective maximization problems. It would be interesting to generalize this result to maximization problems with more than two objectives. Another interesting future work would be

to establish lower bounds for any explicitly given number of solutions for multi-objective maximization problems.

Our approaches deal with maximization problems and they do not seem to apply to minimization problems. A possible explanation is that, in the maximization framework, adding elements to a partial solution rarely deteriorates it. Minimization problems rarely satisfy this property. Establishing constant approximation of the Pareto set with a given number of solutions or show that this is not possible for minimization problems is an interesting open question.

References

1. Alimonti, P.: Non-Oblivious Local Search for Graph and Hypergraph Coloring Problems. In: Nagl, M. (ed.) WG 1995. LNCS, vol. 1017, pp. 167–180. Springer, Heidelberg (1995)
2. Angel, E., Bampis, E., Gourvès, L.: Approximation algorithms for the bi-criteria weighted max-cut problem. Discrete Applied Mathematics 154(12), 1685–1692 (2006)
3. Angel, E., Bampis, E., Gourvès, L., Monnot, J.: (Non)-Approximability for the Multi-criteria *TSP*(1,2). In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 329–340. Springer, Heidelberg (2005)
4. Angel, E., Bampis, E., Kononov, A.: On the approximate tradeoff for bicriteria batching and parallel machine scheduling problems. Theoretical Computer Science 306(1-3), 319–338 (2003)
5. Bazgan, C., Hugot, H., Vanderpooten, D.: Implementing an efficient fptas for the 0-1 multi-objective knapsack problem. European Journal of Operational Research 198(1), 47–56 (2009)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)
7. Edmonds, J.: Paths, trees, and flowers. Canadian Journal of Mathematics 17, 449–467 (1965)
8. Ehrgott, M.: Multicriteria optimization. LNEMS. Springer, Heidelberg (2005)
9. Erlebach, T., Kellerer, H., Pferschy, U.: Approximating multiobjective knapsack problems. Management Science 48(12), 1603–1612 (2002)
10. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of ACM 42(6), 1115–1145 (1995)
11. Halperin, E., Zwick, U.: A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems. Random Structure Algorithms 20(3), 382–402 (2002)
12. Hansen, P.: Bicriteria path problems. In: Fandel, G., Gal, T. (eds.) Multiple Criteria Decision Making: Theory and Applications, pp. 109–127 (1980)
13. Hastad, J.: Some optimal inapproximability results. Journal of ACM 48(4), 798–859 (2001)
14. Kann, V., Lagergren, J., Panconesi, A.: Approximability of maximum splitting of k-sets and some other apx-complete problems. Information Processing Letters 58(3), 105–110 (1996)
15. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)

16. Manthey, B.: On Approximating Multi-Criteria TSP. In: Albers, S., Marion, J.-Y. (eds.) Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009). LIPIcs, pp. 637–648 (2009)
17. Paluch, K., Mucha, M., Mądry, A.: A 7/9 - Approximation Algorithm for the Maximum Traveling Salesman Problem. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 298–311. Springer, Heidelberg (2009)
18. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000), pp. 86–92 (2000)
19. Serafini, P.: Some considerations about computational complexity for multi objective combinatorial problems. In: Jahn, J., Krabs, W. (eds.) Recent Advances and Historical Development of Vector Optimization. Lecture Notes in Economics and Mathematical Systems, vol. 294, pp. 222–232 (1986)
20. Stein, C., Wein, J.: On the existence of schedules that are near-optimal for both makespan and total weighted completion time. Operational Research Letters 21(3), 115–122 (1997)
21. Tsaggouris, G., Zaroliagis, C.: Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-linear Objectives with Applications. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 389–398. Springer, Heidelberg (2006)
22. Warburton, A.: Approximation of pareto-optima in multiple-objective shortest path problems. Operations Research 35(1), 70–79 (1987)
23. Woeginger, G.: A polynomial time approximation scheme for maximizing the minimum machine completion time. Operations Research Letters 20(4), 149–154 (1997)
24. Zhang, J., Yea, Y., Han, Q.: Improved approximations for max set splitting and max NAE SAT. Discrete Applied Mathematics 142(1-3), 133–149 (2004)
25. Zwick, U.: Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In: Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998), pp. 201–210 (1998)

Generalized Maximum Flows over Time*

Martin Groß and Martin Skutella

Fakultät II – Mathematik und Naturwissenschaften,
Institut für Mathematik, Sekr. MA 5-2
Technische Universität Berlin, Straße des 17. Juni 136,
10623 Berlin, Germany
{gross,skutella}@math.tu-berlin.de

Abstract. Flows over time and generalized flows are two advanced network flow models of utmost importance, as they incorporate two crucial features occurring in numerous real-life networks. Flows over time feature time as a problem dimension and allow to realistically model the fact that commodities (goods, information, etc.) are routed through a network over time. Generalized flows allow for gain/loss factors on the arcs that model physical transformations of a commodity due to leakage, evaporation, breeding, theft, or interest rates. Although the latter effects are usually time-bound, generalized flow models featuring a temporal dimension have never been studied in the literature.

In this paper we introduce the problem of computing a generalized maximum flow over time in networks with both gain factors and transit times on the arcs. While generalized maximum flows and maximum flows over time can be computed efficiently, our combined problem turns out to be NP-hard and even completely non-approximable. A natural special case is given by lossy networks where the loss rate per time unit is identical on all arcs. For this case we present a (practically efficient) FPTAS.

Keywords: Flows over Time, Generalized Flows, Approximation Algorithms, Time-Expanded Networks.

1 Introduction

Two crucial characteristics of network flows occurring in real-world applications are flow variation over time and physical transformation of flow resulting in a lesser or greater amount of flow. These characteristics are not captured by standard network flow models known from the literature.

Ford and Fulkerson [11, 12] introduce the notion of flows over time (also called dynamic flows) which model flow variation over time as well as the fact that flow does not travel instantaneously through a network but requires a certain amount of time to travel through each arc. Various interesting examples and

* Supported by the DFG Research Center Matheon "Mathematics for key technologies" in Berlin.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 247–260, 2012.

applications can be found in the survey articles of Aronson [1] and Powell, Jaillet, and Odoni [27].

Generalized flows have been suggested as a tool in production planning as early as 1939 by Kantorovich [20]. They model the situation where flow is not necessarily conserved on every arc but may be physically transformed due to leakage, evaporation, breeding, theft, or interest rates. We refer to the PhD thesis of Wayne [33] for an in-depth treatment of various generalized flow problems.

Both from a practical and theoretical point of view, it seems to be natural to consider a combination of both flow models. However, to the best of our knowledge, generalized flows over time are considered for the first time in this paper. In particular, we hope that the paper will also stimulate further research in this interesting and challenging direction.

Model and Problem. Consider a directed graph G with node set $V(G)$, arc set $E(G)$, capacities $u_e \in \mathbb{R}_{\geq 0}$, transit times $\tau_e \in \mathbb{N}_0$ and gain factors $\gamma_e \in \mathbb{R}_{>0}$ on the arcs $e \in E(G)$. These arc attributes have the following meaning: the capacity of an arc limits the amount of flow that can enter the arc in any time step. For each unit of flow entering the tail of an arc $e \in E(G)$ at time θ, exactly γ_e flow units leave the arc at its head at time $\theta + \tau_e$. We assume that we are given a single source node $s \in V(G)$ without incoming arcs and a single sink node $t \in V(G)$ without outgoing arcs and $s \neq t$. Furthermore, we are given a time horizon $T \in \mathbb{N}$. Combined, we call $(G, u, \tau, \gamma, s, t, T)$ a network. For a node $v \in V(G)$, we denote the outgoing and incoming arcs by $\delta^+_G(v)$ and $\delta^-_G(v)$, respectively.

A generalized flow over time $f : E(G) \times \{0, 1, \ldots, T-1\} \to \mathbb{R}_{\geq 0}$ in such a network is a mapping that assigns flow values $f_{e,\theta} \in [0, u_e]$ to every arc $e \in E(G)$ at every point in time[1] $\theta \in \{0, 1, \ldots, T-1\}$ with respect to (generalized) flow conservation:

$$\sum_{e \in \delta^-(v)} \sum_{\xi=0}^{\theta-\tau_e} \gamma_e f_{e,\xi} \geq \sum_{e \in \delta^+(v)} \sum_{\xi=0}^{\theta} f_{e,\xi} \quad v \in V(G) \setminus \{s\}, \theta \in \{0, \ldots, T-1\}, \quad (1)$$

$$\sum_{e \in \delta^-(v)} \sum_{\xi=0}^{T-\tau_e-1} \gamma_e f_{e,\xi} = \sum_{e \in \delta^+(v)} \sum_{\xi=0}^{T-1} f_{e,\xi} \quad v \in V(G) \setminus \{s,t\}.$$

Moreover, we require that $f_{e,\theta} = 0$ for all $\theta \geq T - \tau_e$ such that no flow remains in the network at time T. The above definition of flow conservation allows storage of flow in nodes; this is referred to as *holdover*. If holdover is not desired, we require that equality holds in (1) for all $v \in V(G) \setminus \{s,t\}$. The value $|f|$ of a generalized flow over time f is the amount of flow sent to the sink within the time horizon: $|f| := \sum_{e \in \delta^-(t)} \sum_{\xi=0}^{T-\tau_e-1} \gamma_e f_{e,\xi}$. Similarly, we write $|x|$ for the value of a static generalized flow x. The *arrival pattern* of a flow over time is a mapping that

[1] In this paper, we use a discrete time model with time steps $0, 1, \ldots, T-1$ for a given time horizon $T \in \mathbb{N}$. Results in this setting often carry over to continuous time models; see, e. g., Fleischer and Tardos [7].

assigns to every time step $\theta \in \{0, 1, \ldots, T-1\}$ the total amount of flow that has arrived at the sink in the time steps $\{0, \ldots, \theta\}$. The generalized maximum flow over time problem asks for a generalized flow over time of maximum value in a given network $(G, u, \tau, \gamma, s, t, T)$.

Previous Work. There has been considerable research on the static generalized maximum flow problem (i. e., our problem without transit times and temporal dimension) and on the maximum flow over time problem (i. e., our problem without gain factors). Since generalized flow problems can be formulated as linear programs [4], they can be solved in polynomial time. The first combinatorial polynomial time algorithm for computing generalized maximum flows was proposed by Goldberg, Plotkin and Tardos [15] and has subsequently been improved by Radzik [28, 29]. Fleischer and Wayne [8], Goldfarb and Jin [16], Goldfarb, Jin and Orlin [17], Restrepo and Williamson [30] and Wayne [33, 34] described further polynomial time algorithms. Truemper [32] noted that generalized maximum flow algorithms show several analogies to minimum cost flow algorithms, if the negative logarithm of a gain factor is used as the cost. Nonetheless, these analogies are limited – it is an open problem whether a strongly polynomial time algorithm for the generalized maximum flow problem exists, contrary to the minimum cost flow problem.

Maximum flows over time have been introduced by Ford and Fulkerson [11, 12] in the 1950's. They proposed two techniques for dealing with them – creating a pseudo-polynomially large time-expanded network to reduce their computation to a static maximum flow problem, and a reduction to the static minimum cost flow problem in the given network, which allows solving this problem in strongly polynomial time. Flows over time that maximize the amount of flow sent to the sink at *any* point in time are called *earliest arrival flows* or *universally maximum flows over time*; this concept is due to Gale [14]. Minieka [23] and Wilkinson [35] showed that the successive shortest path algorithm is capable of solving this problem. Hoppe and Tardos [19] as well as Fleischer and Skutella [6] describe different types of FPTASes for this problem. Nonetheless, the complexity of the earliest arrival flow problem is mostly open; it is, for example, unclear whether this problem is NP-hard or not. This is partly due to the fact, that the arrival pattern (i.e. the function describing the amount of flow arriving at the sink over time) is a piecewise linear function with exponentially many breakpoints (which follows from the work of Zadeh [36]).

Since both generalized maximum flows and maximum flows over time can be dealt with techniques for minimum cost flows, minimum cost flow over time algorithms might seem attractive candidates for generalized maximum flow over time algorithms. However, Klinz and Woeginger [22] showed that the minimum cost flow over time problem is NP-hard.

Our Contribution. In Section 3 we examine the complexity of the generalized maximum flow over time problem with arbitrary gain factors and show that there is no polynomial approximation algorithm, even for the special case of series-parallel networks, unless $P = NP$.

For the special case of lossy networks, we show in Section 4 that the concept of condensed time-expanded networks introduced by Fleischer and Skutella [6] can be successfully generalized to the setting of generalized flows over time and yields an FPTAS. Notice, however, that this FPTAS approximates the time horizon rather than the flow value. That is, for a given time horizon T and $\varepsilon > 0$, the algorithm computes a generalized flow over time with time horizon $(1+\varepsilon)T$ and value at least as big as the value of a maximum generalized flow over time with time horizon T.

Section 5 contains the main contribution of this paper. We consider an important special case of the generalized maximum flow over time problem where gain factors are proportional to transit times. Here *proportional* means that there exists a $c \in \mathbb{R}$ such that $\gamma_e = 2^{c \cdot \tau_e}$ for every arc $e \in E(G)$. Such gain factors are motivated by the fact that in many applications effects such as leakage, evaporation or interest rates are strictly time-bound. Also many processes of growth or decay in nature can be captured by such proportional gain factors. Notice that in this setting paths with equal transit time have equal gain factors and vice versa, due to transit times being additive and gain factors being multiplicative along paths.

In Section 5.1 we show how to compute generalized maximum flows over time with a variant of the successive shortest path algorithm on the static network. This result is particularly interesting since – apart from the most basic maximum flow over time problem – hardly any flow over time problem is known to be solvable by a static flow computation on the underlying static network. It also implies that there are always optimal solutions that do not need holdover. As the successive shortest path algorithm requires an exponential number of iterations in the worst case, our algorithm is not polynomial in the input size.

Therefore we prove in Section 5.2 that an FPTAS can be obtained by terminating the successive shortest path algorithm after a polynomial number of iterations. We wish to emphasize that this FPTAS approximates the maximum flow value rather than the required time horizon (which FPTASes for flow over time problems normally do).

Finally, in Section 6 we conclude with interesting directions for future research. Due to space constraints, we omit many proofs and further details in this extended abstract and refer the reader to the full version of the paper.

2 Preliminaries

A *path* in a graph G is a sequence of arcs $P = (e_1 = (v_1, v_2), \ldots, e_k = (v_k, v_{k+1}))$ for a $k \in \mathbb{N}$, $e_1, \ldots, e_k \in E(G)$, $v_1, \ldots, v_{k+1} \in V(G)$ and $v_i \neq v_j$ unless $i = j$. A *cycle* in a graph G is a sequence of arcs $C = (e_1 = (v_1, v_2), \ldots, e_k = (v_k, v_1))$ for a $k \in \mathbb{N}$, $e_1, \ldots, e_k \in E(G)$, $v_1, \ldots, v_k \in V(G)$ and $v_i \neq v_j$ unless $i = j$. We will treat paths and cycles as sets, if the order of the arcs in a path or cycle is not relevant. With this convention, we will now extend the transit times and gain factors to paths and cycles by defining: $\tau_P := \sum_{e \in P} \tau_e$, $\tau_C := \sum_{e \in C} \tau_e$, $\gamma_P := \prod_{e \in P} \gamma_e$, and $\gamma_C := \prod_{e \in C} \gamma_e$. Cycles C with $\gamma_C = 1$ are called *unit*

gain cycles, cycles with $\gamma_C > 1$ *flow-generating cycles* and cycles with $\gamma_C < 1$ *flow-absorbing cycles*.

In the following let $\overleftarrow{E(G)} := \{ \overleftarrow{e} \mid e \in E(G)\}$ denote the set of reverse arcs of $E(G)$, i. e., each arc $e = (v, w) \in E(G)$ has a reverse arc $\overleftarrow{e} := (w, v) \in \overleftarrow{E(G)}$. Moreover, we set $\overleftarrow{\overleftarrow{e}} := e$ for all $e \in E(G)$.

Definition 1. *The residual network* $(G_x, u_x, \tau, \gamma, s, t, T)$ *of a generalized flow* x *in a network* $(G, u, \tau, \gamma, s, t, T)$ *is defined as follows:*

$$V(G_x) := V(G),$$
$$E(G_x) := \{ e \in E(G) \mid x_e < u_e\} \cup \{ \overleftarrow{e} \mid e \in E(G),\, x_e > 0\} \subseteq E(G) \cup \overleftarrow{E(G)},$$
$$(u_x)_e := \begin{cases} u_e - x_e & e \in E(G), \\ \gamma_{\overleftarrow{e}} x_{\overleftarrow{e}} & e \in \overleftarrow{E(G)}, \end{cases} \quad \text{for all } e \in E(G_x).$$

Transit times and gain factors are extended to the reverse edges as follows:

$$\tau_{\overleftarrow{e}} := -\tau_e \quad \text{and} \quad \gamma_{\overleftarrow{e}} = \frac{1}{\gamma_e}, \quad \text{for all } e \in E(G).$$

Definition 2. *The time expanded network* $(G^T, u^T, \gamma^T, s', t')$ *is constructed from a network* $(G, u, \tau, \gamma, s, t, T)$ *by "copying the network for each time step":*

$$V(G^T) := \{ v_\theta \mid v \in V(G), \theta \in \{0, 1, \ldots, T-1\}\},$$
$$E(G)^T := \{ e_\theta = (v_\theta, w_{\theta+\tau_e}) \mid e = (v, w) \in E(G), \theta \in \{0, \ldots, T - \tau_e - 1\}\},$$
$$H^T := \{ (v_\theta, v_{\theta+1}) \mid v \in V(G), \theta \in \{0, \ldots, T-2\}\},$$
$$E(G^T) := E(G)^T \cup H^T.$$

We call the arcs in H^T holdover arcs. *If holdover is forbidden at intermediate nodes, we simply let* $H^T := \{ (v_\theta, v_{\theta+1}) \mid v \in \{s, t\}, \theta \in \{0, \ldots, T-2\}\}$. *Capacities and gain factors are extended as follows:*

$$u^T_{e'} := \begin{cases} u_e & e' = e_\theta \in E(G)^T, \\ \infty & e' \in H^T, \end{cases} \qquad \gamma^T_{e'} := \begin{cases} \gamma_e & e' = e_\theta \in E(G)^T, \\ 1 & e' \in H^T, \end{cases}$$

for all $e' \in E(G^T)$. *Finally, we set* $s' := s_0$ *and* $t' := t_{T-1}$.

It is not difficult to see that flows over time correspond to static flows in the corresponding time-expanded network and vice versa. We may thus use generalized flows over time in G and generalized static flows in G^T interchangeably. More details on this can be found in full version of the paper. Moreover, we refer to [31] for an introduction to flows over time and related concepts and to Gondran and Minoux [18] for decompositions and optimality criteria for generalized flows.

3 Complexity and Hardness of Approximation

In this section we study the problem in the general case, i. e., in the setting of arbitrary gain factors on the arcs. We begin by analyzing the computational complexity of the problem.

It is easy to see that the generalized maximum flow over time problem can be solved by using the algorithms known for the static generalized maximum flow problem on a time-expanded network. This yields pseudo-polynomial time algorithms, implying that the problem is not *strongly* NP- or PSPACE-hard, unless $P = NP$. As a lower bound for the complexity of the generalized maximum flow over time problem, we will show that there is no polynomial time approximation algorithm for it, unless $P = NP$. It is still unknown whether a strongly polynomial time algorithm exists for the static generalized maximum flow problem. Proof for the theorem in this section can be found in the full version of the paper.

Theorem 1. *There is neither a polynomial algorithm nor a polynomial approximation algorithm for the generalized maximum flow over time problem, even on series-parallel graphs and proportional gains, unless $P = NP$.*

4 Lossy Networks

In this section, we consider the special case of $\gamma_e \leq 1$, for all arcs $e \in E(G)$. This means that flow is only lost, but never gained along arcs. We refer to such networks as *lossy networks*. It is well-known that any network without flow-generating cycles can be turned into a lossy network by node-dependent scaling of flow values. Thus, the results discussed in this section hold for all networks without flow-generating cycles.

Approximating the maximum flow value is hard in general, as we have seen in the last section. This result even carries over to lossy networks if the reduction given in the proof of Theorem 1 is modified accordingly[2]. Therefore, we now focus on relaxing the feasibility, i. e., given some $\alpha > 1$ and a problem instance I with a time horizon T, we ask for a feasible solution to I with time horizon $\alpha \cdot T$ whose value is at least that of an optimal solution to I with time horizon T. We can use the concept of *condensed time-expanded networks* from Fleischer and Skutella [6] to show the following theorem.

Theorem 2. *Let OPT be the value of an optimal solution to a generalized maximum flow over time problem instance $I = (G, u, \tau, \gamma, s, t, T)$ on a lossy network. For any $\epsilon > 0$, there is an algorithm with running time polynomial in the input size and $1/\epsilon$ that computes a solution of value at least OPT for the problem instance $I' = (G, u, \tau, \gamma, s, t, (1 + \epsilon) \cdot T)$.*

A discussion of this theorem can be found in the full version of the paper.

[2] Instead of rewarding the use of the positive length arcs by exponentially large gains, we punish the use of zero length arcs by exponentially small gains.

5 Proportional Losses

In this section, we consider the special case of $\gamma \equiv 2^{c \cdot \tau}$, for some constant $c < 0$. This means that in each time unit the same percentage of the remaining flow value is lost. This is motivated by problems where goods cannot be transported reliably, e. g., due to leakage or evaporation. In many applications, this loss crucially depends on the time spent in the transportation network as many processes of growth or decay in nature evolve over time according to an exponential function. Compare also the work done by Fleischer & Skutella [9] on minimum cost flows over time with proportional costs.

In Section 5.1 we show that the maximum generalized flow over time problem can be solved on the static network by a variant of the Successive Shortest Path Algorithm. This is particularly remarkable as so far only the most basic maximum flow over time problem and the closely related earliest arrival flow problem were known to be solvable to optimality by static flow computations on the static network (i. e., not requiring the use of time-expanded networks).

In Section 5.2 we show how this algorithm can be turned into an FPTAS which is considerably more efficient and uses much less space than the more general FPTAS based on condensed time-expanded networks discussed in Section 4. Furthermore, our FPTAS approximates the flow value instead of the time horizon like Fleischer and Skutella's FPTAS. That is, we approximate optimality instead of feasibility.

5.1 A Variant of the Successive Shortest Path Algorithm

Due to the work of Onaga [25, 26] it is known that augmenting flow successively along highest gain s-t-paths solves the generalized maximum s-t-flow problem. More precisely, Onaga's algorithm for lossy networks proceeds as follows. Begin with the zero-flow and the corresponding residual network. If no source-sink path exists in this residual network, terminate. Otherwise, augment flow along a source-sink path of maximum gain and continue with the resulting flow and residual network. Thus, applying Onaga's algorithm in the time expanded network solves the generalized maximum flow over time problem – at the cost of potentially requiring pseudo-polynomially many augmentations in the pseudo-polynomially large time expanded network.

We will now present an algorithm capable of solving the special case described above using only the original – not time expanded – network. The idea of this algorithm is to employ a strategy similar to Onaga's in the original network and use this as a foundation to construct a flow over time solving the special case.

We begin by introducing some notations; more precisely, we introduce a slightly non-standard way of building a time-expanded network from copies of the original network. Let $(G, u, \tau, \gamma, s, t, T)$ be a (residual) network, let $\gamma_{v \to w}$ be the maximum gain of a v-w-path in G and $\tau_{v \to w}$ the length of a shortest v-w-path

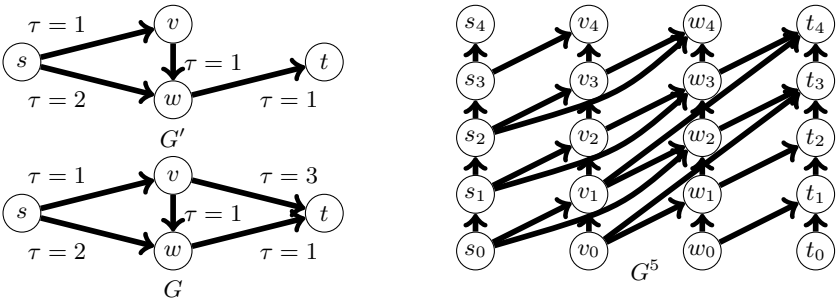


Fig. 1. A network G, its highest-gain network G' and its time-expanded network G^5 (note that gains are defined implicitly by the transit times)

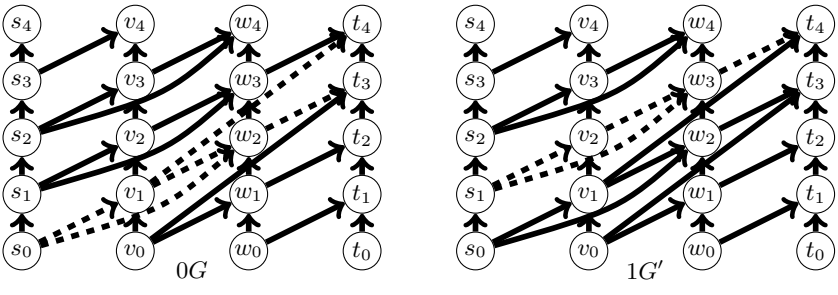


Fig. 2. $0G$ and $1G'$ (dashed) as subnetworks of G^5

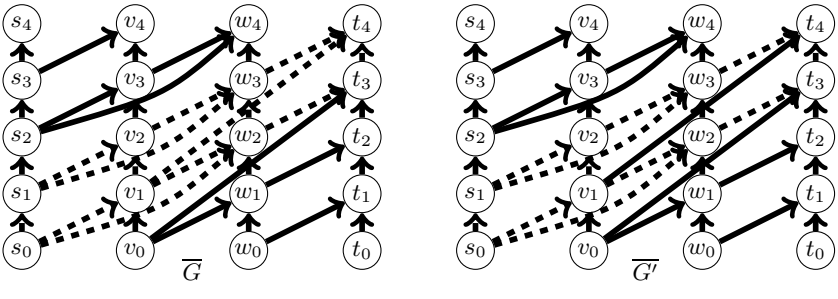


Fig. 3. $\overline{G}$ and $\overline{G'}$ (dashed) as subnetworks of G^5

(with respect to transit times) in G. Notice that $\tau_{v\to w} := \frac{1}{c}\log\gamma_{v\to w}$. Initially, we introduce our construction for the special case of unique gain networks (i. e., a network where all paths from a node $v \in V(G)$ to a node $w \in V(G)$ have the same gain) only; this special case has the advantage of allowing for a simpler and more concise definition. The *θ-copy* θG of G for $\theta \in \{0,\dots,T-\tau_{s\to t}-1\}$ is then:

$$V(\theta G) := \left\{ v_\xi \in V(G^T) \;\middle|\; v \in V(G), \xi = \theta + \tau_{s\to v} \right\}$$
$$E(\theta G) := \left\{ e_\xi \in E(G^T) \;\middle|\; e = (v,w) \in E(G), \xi = \theta + \tau_{s\to v} \right\}$$

More generally, we define *θ-copy* θG of G for some $\theta \in \{0,\dots,T-\tau_{s\to t}-1\}$ to be the following subgraph of the time expanded network G^T:

$$E(\theta G) := \left\{ e_\xi \in E(G^T) \;\middle|\; e = (v,w) \in E(G), \xi = \theta + \tau_{s\to v}, \xi + \tau_e + \tau_{w\to t} < T \right\}$$
$$V(\theta G) := \bigcup_{e=(v,w)\in E(\theta G)} \{v,w\}.$$

For the special case mentioned above these two definitions coincide. Similarly, we define the *$[\theta,\theta']$-copies* $[\theta,\theta']G$ of G, $0 \le \theta < \theta' \le T-\tau_{s\to t}-1$ as

$$V([\theta,\theta']G) := \bigcup_{\xi=\theta}^{\theta'} V(\xi G),$$
$$E([\theta,\theta']G) := \bigcup_{\xi=\theta}^{\theta'} E(\xi G) \cup \bigcup_{v\in V(G)} \bigcup_{\xi=\theta+\tau_{s\to v}}^{\theta'+\tau_{s\to v}-1} \{(v_\xi, v_{\xi+1})\}.$$

For brevity, we also define $\overline{G} := [0, T-\tau_{s\to t}-1]G$ if $\tau_{s\to t} < T-1$ and as the empty graph otherwise. Note that $\overline{G}$ is the subnetwork of G^T containing exactly the nodes and edges of G^T that can be part of s'-t'-paths (with s', t' being the source and sink of the time-expanded network, see Definition 2). For our purposes, it is clearly sufficient to work with $\overline{G}$ instead of G^T.

Furthermore, if we consider an s'-t'-flow f in a time-expanded network G^T, it can happen that flow is sent through holdover edges at source and sink. In this case, the residual network G_f^T corresponding to such a flow f can have reverse holdover edges at source and sink. These reverse holdover edges do not help to construct new s'-t'-paths in the time-expanded-network or new flow-generating cycles with a path to t' so they can be omitted as well. We write $\widetilde{G_f^T}$ for the subnetwork of G_f^T created by removing nodes and edges not on s'-t'-paths and reverse holdover edges at source and sink. Figures 1, 2, and 3 show a network, its time-expansion as well as selected θ- and $[\theta,\theta']$-copies.

Analogously, we define for a flow x in such a unique gain network G the *θ-flow* θx of x in θG for some $\theta \in \{0,\dots,T-\tau_{s\to t}-1\}$ by $(\theta x)_{e_\xi} := x_e$ for

all $e_\xi \in E(\theta G)$. Furthermore, we define the $[\theta,\theta']$-*flow* $[\theta,\theta']x$ of x in $[\theta,\theta']G$ for some $\theta,\theta' \in \{0,\dots,T-\tau_{s\to t}-1\}$ with $\theta<\theta'$ by setting for all $e \in E([\theta,\theta']G)$:

$$([\theta,\theta']x)_e := \begin{cases} 0 & e=(v_\xi,v_{\xi+1}), v\in V(G)\setminus\{s,t\}, \\ (\theta'-\theta+1)|x| & e=(s_\xi,s_{\xi+1}), \xi<\theta, \\ (\theta'-\xi)|x| & e=(s_\xi,s_{\xi+1}), \theta\le\xi<\theta', \\ 0 & e=(s_\xi,s_{\xi+1}), \theta'\le\xi, \\ 0 & e=(t_\xi,t_{\xi+1}), \xi\le\theta+\tau_{s\to t}, \\ (\xi-\theta-\tau_{s\to t}+1)|x| & e=(t_\xi,t_{\xi+1}), \theta+\tau_{s\to t}<\xi<\theta'+\tau_{s\to t}, \\ (\theta'-\theta+1)|x| & e=(t_\xi,t_{\xi+1}), \xi\ge\theta'+\tau_{s\to t}, \\ x_e & e=e_\xi. \end{cases}$$

Again, we define for brevity $\overline{x} := [0,T-\tau_{s\to t}-1]x$ for a flow x in G, if $\tau_{s\to t}<T-1$ and as the zero flow otherwise. Informally spoken, the idea of our algorithm is to start with the zero flow, compute a maximum-flow in the highest-gain / shortest-path subnetwork of the static residual network, augment this flow and repeat this process until no s-t-path exists in the static residual network. We then use the augmented maximum-flows to construct an optimal solution to our problem, by sending each flow as long as possible into the network (i. e., temporally repeated). We will use the notations introduced above to describe the construction of the flow over time in the last step of our algorithm, and show its optimality.

Algorithm 1. *Let $I=(G,u,\tau,\gamma,s,t,T)$ be an instance of the generalized maximum flow over time problem.*

1. *Begin with $i:=0$ and the static zero-flow $x_0:\equiv 0$.*
2. *If no s-t-path exists in G_{x_i} or if $\tau_{s\to t}\ge T$ in G_{x_i}, then set $k:=i-1$ and go to step 6.*
3. *Restrict the static residual network G_{x_i} to the network G'_{x_i} containing only paths of maximum gain and compute a generalized maximum flow x'_i in G'_{x_i}.*
4. *Define x_{i+1} by adding x'_i to x_i as follows: $(x_{i+1})_e := (x_i)_e+(x'_i)_e+\gamma_e^{-1}(x'_i)_{\overleftarrow{e}}$ for all $e\in E(G)$. Notice that x_{i+1} is a feasible flow in G, since x'_i is a feasible flow in a restricted residual network of x_i.*
5. *Set $i:=i+1$ and go to step 2.*
6. *Construct a generalized flow over time f defined by $f=\sum_{j=0}^{k}\overline{x'_j}$.*

We will prove the correctness of Algorithm 1 by comparing it to Onaga's algorithm applied to the time expanded network. For $i=0,\dots,k+1$, define a generalized flow over time $f_i := \sum_{j=0}^{i-1}\overline{x'_j}$. In particular, f_0 is the zero flow over time and $f=f_{k+1}$. The strategy for our proof is to show that in every iteration $\overline{x'_i}$ is a flow along highest gain paths in $G^T_{f_i}$. Then successively adding $\overline{x'_0},\dots,\overline{x'_k}$ produces the same result as Onaga's algorithm applied to the time expanded network, showing correctness of our algorithm. The following claim turns out to be helpful in proving the correctness of Algorithm 1. Proof of it and the following theorem can be found in the full version of the paper.

Claim. For each $i = 0, \ldots, k+1$, it holds that $\overline{G_{x_i}} = \widetilde{G^T_{f_i}}$, i. e., the copied static network is equal to the pruned time-expanded network after each iteration of the algorithm.

Theorem 3. *Algorithm 1 computes a generalized maximum flow over time.*

We conclude this section by examining the running time of Algorithm 1. Let $n := |V(G)|$, $m := |E(G)|$ and $U := \max_{e \in E(G)} u_e$ for a problem instance $(G, u, \tau, \gamma, s, t, T)$. In our case, a highest gain path can be found in $O(nm)$ time using Moore-Bellman-Ford's algorithm (see Bellman [2], Ford [10], Moore [24]) or in $O(m + n \log n)$ by applying Dijkstra's algorithm [5] with Fibonacci heaps (see Fredman and Tarjan [13]) and reduced costs. For both algorithms, $\tau_e = \frac{1}{c} \log \gamma_e$ is being used as a cost function. Both algorithms are capable of computing the highest-gain network as well. A generalized maximum flow in the highest-gain network can then be computed by a standard maximum flow algorithm. Since there are at most T time steps, there can be at most T iterations. The running time of an iteration is dominated by the maximum flow computation, yielding a running time of $O(maxflow \cdot T)$, where $O(maxflow)$ is the running time of the maximum flow algorithm. King, Rao, and Tarjan[21] describe a maximum flow algorithm with a running time of $O(nm \log_{m/(n \log n)} n)$, resulting in a running time of $O(nm \log_{m/(n \log n)} n \cdot T)$ for our algorithm.

For special cases, this runtime can be improved further. Beygang, Krumke, and Zeck [3] recently studied static generalized maximum flows in series-parallel networks and discovered that a greedy-strategy that chooses always the highest-gain path in the original – not residual – network is sufficient for finding an optimal solution. This can be carried over to our setting and can be used for bounding the number of paths used. Since each augmentation saturates an arc, there can be at most m iterations, yielding a polynomial time algorithm.

5.2 Turning the Algorithm into an FPTAS

In this section, we will see that Algorithm 1 can be terminated early to obtain an approximate solution. In fact, the algorithm can be turned into an FPTAS. Proofs for the theorems in this section can be found in the full version of the paper.

Theorem 4. *Let OPT be the value of an optimal solution to a generalized maximum flow over time problem instance $I = (G, u, \tau, \gamma, s, t, T)$, $\varepsilon > 0$, and $U := \max_{e \in E} u_e$. Algorithm 1 has found a solution of value at least $OPT - \varepsilon$ after all paths of length $\leq -\frac{1}{c}(\log \frac{1}{\varepsilon} + \log m + \log U + 2 \log T)$ have been processed (recall that the lengths of the paths used by the algorithm are monotonically increasing). For a constant $c < 0$ and using a maximum flow based approach as proposed in Section 5.1, this leads to a running time of $O(maxflow \cdot (\log \varepsilon^{-1} + \log U + \log T))$ for a solution of value at least $OPT - \varepsilon$.*

The above theorem allows an approximation within a constant value ε. For an FPTAS, we need to approximate OPT within a factor of $(1 - \varepsilon)$ or a value of εOPT. This can be done by a slight modification of Theorem 4.

Theorem 5. *Let OPT be the value of an optimal solution for a generalized maximum flow over time problem instance $I = (G, u, \tau, \gamma, s, t, T)$, $\varepsilon > 0$ and $U := \max_{e \in E} u_e$. Algorithm 1 has found a solution of value at least $(1-\varepsilon)OPT$ after $\lceil -\frac{1}{c}(\log \frac{1}{\varepsilon} + \log m + \log U + 2 \log T) \rceil$ iterations. For a constant $c < 0$ and using a maximum flow based approach as proposed in Section 5.1, this leads to a running time of $O(maxflow \cdot (\log \varepsilon^{-1} + \log U + \log T))$ for a solution of value at least $(1-\varepsilon)OPT$.*

6 Conclusion

We have introduced the generalized maximum flow over time problem that, for the first time, combines important features captured by flows over time and generalized flows in one network flow model. While the generalized maximum flow over time problem cannot be approximated in polynomial time, unless P=NP, we have presented an efficient FPTAS for the special case of lossy networks with proportional gain factors.

The generalized flow over time model presented in this paper raises numerous interesting questions and directions for future research. The most natural generalizations of the considered network flow problem seem to be generalized minimum cost flows over time and generalized multicommodity flows over time. An interesting approach to these flow problems is the concept of *condensed time-expanded networks* introduced by Fleischer and Skutella [6]. However, as mentioned in Section 4, the analysis of these condensed time-expanded networks crucially relies on the assumption that, in an optimum solution, flow particles travel along simple paths from the source to the sink. This assumption, however, is no longer valid for generalized flows over time in networks containing flow-generating cycles. The same holds for multi-commodity flows over time without holdover at intermediate nodes. With respect to practical applications, it is an important open problem and a big theoretical challenge to make condensed time-expanded networks usable and, in particular, analyzable for such flow over time problems.

Acknowledgements. The authors wish to thank the anonymous referees whose valuable comments helped to improve the presentation of the paper.

References

[1] Aronson, J.E.: A survey of dynamic network flows. Annals of Operations Research 20, 1–66 (1989)
[2] Bellman, R.E.: On a routing problem. Quarterly of Applied Mathematics 16, 87–90 (1958)
[3] Beygang, K., Krumke, S.O., Zeck, C.: Generalized max flow in series-parallel graphs. Report in Wirtschaftsmathematik 125, TU Kaiserslautern (2010)
[4] Dantzig, G.B.: Linear programming and extensions. Princeton University Press (1962)

[5] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
[6] Fleischer, L., Skutella, M.: Quickest flows over time. SIAM Journal on Computing 36, 1600–1630 (2007)
[7] Fleischer, L.K., Tardos, É.: Efficient continuous-time dynamic network flow algorithms. Operations Research Letters 23, 71–80 (1998)
[8] Fleischer, L.K., Wayne, K.D.: Fast and simple approximation schemes for generalized flow. Mathematical Programming 91, 215–238 (2002)
[9] Fleischer, L., Skutella, M.: Minimum cost flows over time without intermediate storage. In: Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms, Baltimore, MD, pp. 66–75 (2003)
[10] Ford, L.R.: Network flow theory. Paper P-923, The Rand Corporation (1956)
[11] Ford, L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press (1962)
[12] Ford, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. Operations Research 6, 419–433 (1987)
[13] Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization problems. Journal of the ACM 34, 596–615 (1987)
[14] Gale, D.: Transient flows in networks. Michigan Mathematical Journal 6, 59–63 (1959)
[15] Goldberg, A.V., Plotkin, S.A., Tardos, É.: Combinatorial algorithms for the generalized circulation problem. Mathematics of Operations Research 16, 351–379 (1991)
[16] Goldfarb, D., Jin, Z.: A faster combinatorial algorithm for the generalized circulation problem. Mathematics of Operations Research 21, 529–539 (1996)
[17] Goldfarb, D., Jin, Z., Orlin, J.B.: Polynomial-time highest gain augmenting path algorithms for the generalized circulation problem. Mathematics of Operations Research 22, 793–802 (1997)
[18] Gondran, M., Minoux, M.: Graphs and Algorithms. Wiley (1984)
[19] Hoppe, B., Tardos, É.: Polynomial time algorithms for some evacuation problems. In: Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 433–441 (1994)
[20] Kantorovich, L.V.: Mathematical methods of organizing and planning production. Technical report, Publication House of the Leningrad State University (1939); Translated in Management Science 6, 366–422 (1960)
[21] King, V., Rao, S., Tarjan, R.: A faster deterministic maximum flow algorithm. Journal of Algorithms 17, 447–474 (1994)
[22] Klinz, B., Woeginger, G.J.: Minimum cost dynamic flows: The series parallel case. Networks 43, 153–162 (2004)
[23] Minieka, E.: Maximal, lexicographic, and dynamic network flows. Operations Research 21, 517–527 (1973)
[24] Moore, E.F.: The shortest path through a maze. In: Proceedings of the International Symposium on Switching, Part II, pp. 285–292. Harvard University Press (1959)
[25] Onaga, K.: Dynamic programming of optimum flows in lossy communication nets. IEEE Transactions on Circuit Theory 13, 282–287 (1966)
[26] Onaga, K.: Optimal flows in general communication networks. Journal of the Franklin Institute 283, 308–327 (1967)
[27] Powell, W.B., Jaillet, P., Odoni, A.: Stochastic and dynamic networks and routing. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (eds.) Network Routing, ch. 3, vol. 8, pp. 141–295. Handbooks in Operations Research and Management Science, North–Holland, Amsterdam, The Netherlands (1995)

[28] Radzik, T.: Faster algorithms for the generalized network flow problem. Mathematics of Operations Research 23, 69–100 (1998)
[29] Radzik, T.: Improving time bounds on maximum generalised flow computations by contracting the network. Theoretical Computer Science 312, 75–94 (2004)
[30] Restrepo, M., Williamson, D.P.: A simple gap-canceling algorithm for the generalized maximum flow problem. Mathematical Programming 118, 47–74 (2009)
[31] Skutella, M.: An introduction to network flows over time. In: Research Trends in Combinatorial Optimization, pp. 451–482. Springer, Heidelberg (2009)
[32] Truemper, K.: On max flows with gains and pure min-cost flows. SIAM Journal on Applied Mathematics 32, 450–456 (1977)
[33] Wayne, K.D.: Generalized Maximum Flow Algorithms. PhD thesis, Cornell University (1999)
[34] Wayne, K.D.: A polynomial combinatorial algorithm for generalized minimum cost flow. Mathematics of Operations Research 27, 445–459 (2002)
[35] Wilkinson, W.L.: An algorithm for universal maximal dynamic flows in a network. Operations Research 19, 1602–1612 (1971)
[36] Zadeh, N.: A bad network problem for the simplex method and other minimum cost flow algorithms. Mathematical Programming 5, 255–266 (1973)

The Price of Anarchy for Minsum Related Machine Scheduling

Ruben Hoeksma and Marc Uetz

University of Twente, Dept. Applied Mathematics, P.O. Box 217, 7500AE Enschede, The Netherlands
{r.p.hoeksma,m.uetz}@utwente.nl

Abstract. We address the classical uniformly related machine scheduling problem with minsum objective. The problem is solvable in polynomial time by the algorithm of Horowitz and Sahni. In that solution, each machine sequences its jobs shortest first. However when jobs may choose the machine on which they are processed, while keeping the same sequencing rule per machine, the resulting Nash equilibria are in general not optimal. The price of anarchy measures this optimality gap. By means of a new characterization of the optimal solution, we show that the price of anarchy in this setting is bounded from above by 2. We also give a lower bound of $e/(e-1) \approx 1.58$. This complements recent results on the price of anarchy for the more general unrelated machine scheduling problem, where the price of anarchy equals 4. Interestingly, as Nash equilibria coincide with shortest processing time first (SPT) schedules, the same bounds hold for SPT schedules. Thereby, our work also fills a gap in the literature.

1 Introduction

The minsum related machine scheduling problem is one of the classical models in the area of scheduling. It has been solved already in the 1960s [5]. Given are n jobs with non-preemptive processing requirements, a set of m parallel machines with different processing speeds, the goal is to find a schedule that minimizes the sum of job completion times. In the 3-field notation of Graham et al. [8] the problem is denoted $Q||\sum C_j$. The problem is a special case of the more general unrelated machine scheduling problem $R||\sum C_j$, where the processing times of jobs on machines are represented by an arbitrary $n \times m$ matrix. The related machine problem is solved in $O(n \log nm)$ computation time by the MFT algorithm of Horowitz and Sahni [10]. The MFT algorithm is a refinement of the simple matching solution presented earlier by Conway et al. [5, pp. 78-79]. The MFT algorithm computes the optimal assignment of jobs to machines by considering them in the order longest processing time first (LPT), and the jobs eventually assigned to a given machine are then sequenced in the order shortest processing time first (SPT).

In this paper we are interested in the same problem, but in a **decentralized setting** where there is no central authority that assigns jobs to machines.

R. Solis-Oba and G. Persiano (Eds.): WAOA 2011, LNCS 7164, pp. 261–273, 2012.

Instead, jobs themselves choose the machine on which they want to be processed. Any job j seeks to minimize its own completion time C_j, and does not care about the central objective function $\sum C_j$. This results in an n-agent strategic game where the strategy space of any job-agent is the set of machines. This game is well-defined once we determine how jobs are locally sequenced on each machine. Here we only consider the local sequencing rule that is locally optimal for the global objective $\sum C_j$, that is, the jobs on each machine are processed in order of shortest processing time first (SPT). In spite of doing the optimal thing locally, Nash equilibria of the resulting game do not necessarily lead to globally optimal solutions for the objective $\sum C_j$. This optimality gap is what we are interested in. Notice that the problem that we have described so far is an example of a coordination mechanism as defined by Christodoulou et al. [3], who suggested to use local sequencing rules per machine in order to influence the dynamics of the game and thereby the quality of the corresponding equilibrium outcomes.

The **price of anarchy** is being used since about a decade to measure the deterioration of system performance caused by the lack of central coordination [13,16]. It is defined by relating the quality of the worst possible Nash equilibrium to the quality of the globally optimal solution. Here, the metric for the quality of a solution is in terms of the central objective function, in our case $\sum C_j$. In the economic literature the central objective function is rather called social choice function [15]. In our case it is utilitarian, which means that the social choice function $\sum C_j$ is simply the sum of the valuation functions of the agents C_j.

For games with utilitarian social choice function, Roughgarden [17] recently introduced the concept of *smoothness* of games and its consequences for **robust price of anarchy** bounds. He points out that many of the existing price of anarchy bounds can actually be deduced from smoothness of the underlying games, and he shows that the corresponding bounds not only hold for pure Nash equilibria, but extend to mixed Nash equilibria, correlated equilibria as defined by Aumann [1], and even beyond.

The **contribution of this paper** is an analysis of the price of anarchy for the minsum related machine scheduling game as described above. More specifically, our main result is a proof that the pice of anarchy is at most 2. This analysis also extends beyond pure Nash equilibria in the same way as in [17], even though it is not exactly a smoothness argument in the sense of Roughgarden's definition in [17]. We also give a parametric example to show that the price of anarchy cannot be less than $e/(e-1) \approx 1.58$.

An interesting aspect of our work is that also the pure Nash equilibria can easily be computed in polynomial time through **SPT schedules**. In fact, it is well known that Nash equilibria are obtained as solutions of the Ibarra and Kim algorithm [11] when machines sequence jobs in SPT order. This is even true for the more general unrelated machine scheduling problem [9,12]. When applied to the related machine scheduling problem considered here, this means scheduling the jobs in SPT order, and when a job is scheduled it is placed on the machine

that minimizes its completion time C_j[1]. This results in a pure Nash equilibrium of the game, as no job has the possibility to improve its completion time by changing to another machine. Hence, we only need to compare optimum and Nash equilibrium solutions, none of which is blemished by NP-completeness. In a first instant we therefore thought the problem was trivial. Yet we first needed a new characterization of the optimal solution to get the job done. In any case, our results also show that SPT schedules can miss the optimum by no more than a factor 2, and can be as bad as $e/(e-1)$ times the optimum.

It is also worth mentioning that the literature related to analyzing the price of anarchy for scheduling problems has almost exclusively concentrated on the egalitarian[2] **makespan objective** $C_{\max}(= \max_j C_j)$ as social choice function [2,3,7,12,13,18]. The fact that most of the literature focusses on the makspan has potentially two reasons. First, this is the model that has been originally proposed by Koutsoupias and Papadimitriou [13]. Second, makespan scheduling is akin to load balancing, with applications for example in internet routing protocols [16]. Yet it is surprising that utilitarian social choice functions have hardly received any attention from the algorithms community, given that the model is certainly not less attractive from an application perspective.

We are aware of only two references that are very closely **related to our work**, these are the recent papers by Correa and Queyranne [6] and Cole et al. [4]. Both papers address the same problem as we do, but with additional job weights w_j and in the more general context of unrelated machine scheduling, $\mathrm{R}||\sum w_j C_j$. Their objective is thus weighted utilitarian. One of the main results in both papers is the proof that the price of anarchy equals 4 when machines sequence their jobs locally optimal, that is, according to nonincreasing ratios of weight over processing time. Cole et al. [4] also give an instance which establishes a lower bound of 4 for the price of anarchy, even in the unweighted case, $\mathrm{R}||\sum C_j$. Our results nicely fit into that context.

The **organization of the paper** is as follows. In Section 2 we briefly recap the algorithm of Horowitz and Sahni [10]. We then present a new characterization of optimal solutions, which is crucial for the subsequent analysis. In Section 4 we show that the price of anarchy is not greater than 2. The basic proof idea is akin to the arguments for showing $(2,0)$-smoothness of the game, but we crucially need the characterization of optimal solutions. Hence it is at best a relaxed sort of smoothness. Section 5 describes a parametric instance, for which we show

[1] This is not the same as the "SPT schedules" as discussed by Horowitz and Sahni [10, p. 321], as they assign jobs in SPT order in a greedy list scheduling fashion, that is, to the machine that minimizes the jobs *starting* time. When doing that, the resulting SPT schedule can be arbitrarily far away from the optimum. When we refer to SPT schedules we refer to greedy list scheduling in SPT order, but jobs are placed on the machine that minimizes the completion time C_j.

[2] See Myerson [14] for a discussion of utilitarian and egalitarian social choice functions. The interpretation of $C_{\max}$ as egalitarian indeed makes sense in models where the objectives of the job-agents is the total load of the machine they are processed on, as for example in [13].

that it's price of anarchy is equal to $e/(e-1) > 1.5819$. We conclude with some further remarks in Section 6.

2 Characterization of Optimal Solutions

In this section we briefly recap the MFT algorithm of Horowitz and Sahni [10] and establish a new characterization for optimal solutions for minsum related machine scheduling. This characterization is crucial to our analysis in Section 4.

Throughout this paper we denote by J the set of n jobs and by M the set of m machines. Each job j has a length p_j and each machine i has a speed s_i. The processing requirement of job j on machine i is equal to $p_{ij} = p_j/s_i$. W.l.og. assume that $p_1 \leq p_2 \leq \cdots \leq p_n$ and $s_1 \leq s_2 \leq \cdots \leq s_m$. We assume ties on the ordering are broken consistently and that this is done based on index.

For the single machine case it is clear that the contribution of a job can be measured by its position in the schedule and its processing time. This follows from rewriting the objective function as follows. Let φ be an ordering of the jobs and let $\varphi(k)$ denote the k-th job in this ordering, then $\sum_{k=1}^{n} C_{\varphi(k)} = \sum_{k=1}^{n}\sum_{l=1}^{k} p_{\varphi(l)} = \sum_{k=1}^{n}(n-k+1)p_{\varphi(k)}$. Hence the only optimal schedules are schedules that schedule the jobs in order of nondecreasing processing time, as these match large p_j to small values $(n-k+1)$. The same idea can be extended to the case of parallel machines, even with speeds, resulting in the following *Minimum Mean Flow Time* (MFT) algorithm [10].

Algorithm 1. MFT Algorithm for problem $Q||\sum C_j$

For each machine i set $h_i = 0$
while *Not all jobs are placed* **do**
  Take from the unscheduled jobs the longest job j
  Assign job j to the machine with the smallest value of $(h_i+1)/s_i$
  For that machine update $h_i = h_i + 1$
Sort the jobs on each machine in SPT order

Similar to the single machine case, the different values $(h_i+1)/s_i$ are the values for a job's possible positions in the schedule, as in general, the x-th last job on a machine contributes to the objective value x times its processing time divided by the machine speed. The algorithm assigns the currently longest unscheduled job to the machine with the currently smallest position value.

Theorem 1 ([10]). *Any optimal schedule for* $Q||\sum C_j$ *can be computed by the MFT algorithm with the proper tie breaking rule.*

Since any optimal solution has the jobs on each machine sequenced in SPT order, we can identify a schedule by denoting for each job on which machine it is scheduled. Therefore we identify a schedule with an n-vector σ where σ_j is the machine on which job j is scheduled.

Next, let $h^\sigma(j)$ be the vector such that $h_i^\sigma(j) = |\{k > j | \sigma_k = i\}|$, indicating the number of jobs on machine i in schedule σ that have higher index than j. Now any schedule σ is optimal if and only if

$$\frac{h_{\sigma_j}^\sigma(j)+1}{s_{\sigma_j}} \leq \frac{h_i^\sigma(j)+1}{s_i} \text{ for all jobs } j \text{ and all machines } i \ . \tag{1}$$

This because, for all machines i, $(h_i^\sigma(j)+1)/s_i$ is the position value of i upon placement of job j in the MFT algorithm. This needs to be minimized for all j by any optimal schedule σ. The following lemma provides our new characterization of optimal solutions.

Lemma 1. *A schedule σ is optimal for $Q||\sum C_j$ if and only if*

$$\frac{h_i^\sigma(j)+1}{s_i} \geq \frac{h_\ell^\sigma(j)}{s_\ell} \text{ for all machines } i \text{ and } \ell \ . \tag{2}$$

Proof. We show that (2) is true if and only if (1) is true. Let σ be an optimal schedule. Note that $h_i^\sigma(j) \geq h_i^\sigma(k)$ for all machines i and all jobs $k \geq j$. We therefore get from (1) that

$$\frac{h_i^\sigma(j)+1}{s_i} \geq \frac{h_i^\sigma(k)+1}{s_i} \geq \frac{h_{\sigma_k}^\sigma(k)+1}{s_{\sigma_k}}$$

for all machines i and all jobs $k \geq j$. Since for any machine ℓ either $h_\ell^\sigma(j) = 0$, or there is a job $k > j$ such that $\sigma_k = \ell$ and $h_\ell^\sigma(j) = h_{\sigma_k}^\sigma(j) = h_{\sigma_k}^\sigma(k) + 1$, it follows that

$$\frac{h_i^\sigma(j)+1}{s_i} \geq \frac{h_\ell^\sigma(j)}{s_\ell} \text{ for all machines } i \text{ and } \ell \ .$$

Now let σ be a schedule that satisfies (2) and suppose it does not satisfy (1). Then there exist $j \in J$ and $i \in M$ such that

$$\frac{h_{\sigma_j}^\sigma(j)+1}{s_{\sigma_j}} > \frac{h_i^\sigma(j)+1}{s_i} \ ,$$

but then we get for job $j-1$ that

$$\frac{h_{\sigma_j}^\sigma(j-1)}{s_{\sigma_j}} = \frac{h_{\sigma_j}^\sigma(j)+1}{s_{\sigma_j}} > \frac{h_i^\sigma(j)+1}{s_i} = \frac{h_i^\sigma(j-1)+1}{s_i} \ ,$$

which contradicts (2). □

A intuitive interpretation for (2) is that, when applying the MFT algorithm, a job that is placed on a machine can not get a better position than the jobs already placed on a machine. While it is intuitive that this is indeed a necessary condition for the optimal solution, the intuition that it is also sufficient is not that clear. In that sense, it is indeed a nontrivial reformulation of (1).

3 Coordination Mechanism and Nash Equilibria

For the remainder of this paper we compare the optimal solution from Section 2 to outcomes of the scheduling game for $Q||\sum C_j$ where each job can individually choose on which machine it will be scheduled and machines sequence jobs in SPT order. The jobs act selfishly, each trying to minimize its own completion time. Nash equilibria are considered the natural outcomes of the resulting strategic game. The price of anarchy, defined in [13], compares the objective value of an optimal schedule to the objective value of a worst possible Nash equilibrium schedule. The resulting game for $Q||\sum C_j$ is a coordination mechanism in the sense of Christodolou et al. [3], where using SPT locally per machine proposes itself because it is locally optimal.

We denote schedules in the same way as in Section 2, but with respect to Nash equilibria, σ represents the *strategy profile* of the job-agents such that σ_j is the machine chosen by job j. Furthermore, σ_{-j} denotes the $(n-1)$-vector obtained from σ by deleting σ_j, so that $\sigma = (\sigma_j, \sigma_{-j})$. For the problem $Q||\sum C_j$ with SPT as local scheduling rule, Nash equilibria are defined as follows.

Definition 1 (Nash equilibrium). *A strategy profile $\sigma = (\sigma_j, \sigma_{-j})$ is a Nash equilibrium if and only if for all jobs j,*

$$\sum_{\substack{k \le j \\ \sigma_k = \sigma_j}} \frac{p_k}{s_{\sigma_j}} \le \sum_{\substack{k < j \\ \sigma_k = i}} \frac{p_k}{s_i} + \frac{p_j}{s_i} \quad \text{for all machines } i \ . \tag{3}$$

It is well known [9] that the *Ibarra-Kim* algorithm [11] constructs all Nash equilibria depending on the way ties are broken. For uniformly related machines the algorithm is described as follows.

Algorithm 2. Ibarra-Kim Algorithm for problem $Q||\sum C_j$

while *Not all jobs are placed* **do**
- Take from the unscheduled jobs the shortest job k
- Let machine l be the machine where job k has minimal completion time
- Schedule job k directly after the jobs already scheduled on machine l

The Ibarra-Kim algorithm was originally designed as an approximation algorithm for unrelated machine scheduling [11]. To the best of our knowledge the performance of the resulting schedules for the related machine problem $Q||\sum C_j$ has not yet been analyzed, most probably because the problem to find optimal solutions was settled long before in [5].

4 Upper Bound on the Price of Anarchy

In this Section we establish an upper bound on the price of anarchy for minsum related machine scheduling. Our proof is (in retrospect) akin to a smoothness argument for cost-minimization (=utilitarian) games, as introduced by Roughgarden [17].

Definition 2 ([17] Smooth Games). *A cost-minimization game is* (λ, μ)-smooth *if for every two outcomes* ν *and* σ*,*

$$\sum_{j=1}^{n} C_j(\sigma_j, \nu_{-j}) \leq \lambda \cdot \sum_{j=1}^{n} C_j(\sigma) + \mu \cdot \sum_{j=1}^{n} C_j(\nu) \ . \tag{4}$$

If a utilitarian game is (λ, μ)-smooth with $\lambda \geq 0$ and $\mu < 1$, it follows that for any Nash equilibrium ν and optimal solution σ

$$\sum_{j=1}^{n} C_j(\nu) \leq \sum_{j=1}^{n} C_j(\sigma_j, \nu_{-j}) \leq \lambda \cdot \sum_{j=1}^{n} C_j(\sigma) + \mu \cdot \sum_{j=1}^{n} C_j(\nu) \ . \tag{5}$$

From (5) it follows directly that $\frac{\lambda}{1-\mu}$ is an upper bound on the price of anarchy for any (λ, μ)-smooth game. Roughgarden [17] defines the *robust price of anarchy* as the least upper bound on the price of anarchy that is provable through a smoothness argument.

Definition 3 ([17] Robust PoA). *The robust price of anarchy of a cost-minimization game is*

$$\inf \left\{ \frac{\lambda}{1-\mu} \middle| \text{ the game is } (\lambda, \mu)\text{-smooth} \right\} \ .$$

Instead of proving (4) for *any* two outcomes ν and σ, we crucially need the characterization of the optimal solution from Lemma 1 and therefore will prove (4) with σ restricted to be an optimal solution. However, note that the resulting bound on the price of anarchy also extends to (mixed) Nash equilibria, correlated equilibria or no-regret sequences (see [17]) when (4) only holds for arbitrary strategy profiles ν and an optimal solution σ.

In the following, let therefore σ be an optimal schedule resulting from the MFT algorithm, and recall that for the objective value in the optimal solution σ we have

$$\sum_{j=1}^{n} C_j(\sigma) = \sum_{j=1}^{n} \left(h^{\sigma}_{\sigma_j}(j) + 1 \right) \frac{p_j}{s_{\sigma_j}} \ .$$

The next Theorem is the main result of this paper.

Theorem 2. *The price of anarchy for the minsum related machine scheduling problem* $Q||\sum C_j$ *with SPT as local sequencing rule is no greater than 2.*

Proof. We show that the game is "$(2, 0)$-smooth", by showing that

$$\sum_{j=1}^{n} C_j(\sigma_j, \nu_{-j}) \leq 2 \sum_{j=1}^{n} C_j(\sigma) \tag{6}$$

for an optimal schedule σ and any strategy profile ν.

Let $J_i(\sigma) = \{j | \sigma_j = i\}$ be the set of jobs scheduled on machine i in the optimal solution σ, likewise let $J_i(\nu) = \{j | \nu_j = i\}$ be the set of jobs scheduled on machine

i in schedule ν. For any job j in $J_i(\sigma)$, its completion time $C_j(\sigma_j, \nu_{-j})$ consists of the processing times of all jobs that are on machine i in ν and that have smaller index than j, plus its own processing time on machine i. Summing the completion times of all jobs that are on machine i in the optimal solution gives us

$$\sum_{j\in J_i(\sigma)} C_j(\sigma_j,\nu_{-j}) = \sum_{j\in J_i(\sigma)} \left(\frac{p_j}{s_i} + \sum_{\substack{k\in J_i(\nu)\\ k<j}} \frac{p_k}{s_i} \right)$$
$$= \sum_{j\in J_i(\sigma)} \frac{p_j}{s_i} + \sum_{j\in J_i(\sigma)} \sum_{\substack{k\in J_i(\nu)\\ k<j}} \frac{p_k}{s_i} \ . \tag{7}$$

Note that the number of times that a job k is counted on the right hand side of (7) equals the number of jobs with higher index than j on machine i in the optimal solution, times $\frac{1}{s_i}$. In other words, the second part of (7) can be rewritten as

$$\sum_{j\in J_i(\sigma)} \sum_{\substack{k\in J_i(\nu)\\ k<j}} \frac{p_k}{s_i} = \sum_{k\in J_i(\nu)} h_i^\sigma(k) \cdot \frac{p_k}{s_i} \ .$$

This gives us

$$\sum_{j\in J_i(\sigma)} C_j(\sigma_j,\nu_{-j}) = \sum_{j\in J_i(\sigma)} \frac{p_j}{s_i} + \sum_{k\in J_i(\nu)} h_i^\sigma(k) \cdot \frac{p_k}{s_i} \ .$$

Now, note that by definition $\sigma_j = \nu_k = i$, so

$$\sum_{j\in J_i(\sigma)} C_j(\sigma_j,\nu_{-j}) = \sum_{j\in J_i(\sigma)} \frac{p_j}{s_{\sigma_j}} + \sum_{k\in J_i(\nu)} h_{\nu_k}^\sigma(k) \cdot \frac{p_k}{s_{\nu_k}} \ .$$

Summing over all i leads to

$$\sum_{j=1}^{n} C_j(\sigma_j,\nu_{-j}) = \sum_{i=1}^{m} \sum_{j\in J_i(\sigma)} C_j(\sigma_j,\nu_{-j})$$
$$= \sum_{i=1}^{m} \sum_{j\in J_i(\sigma)} \frac{p_j}{s_{\sigma_j}} + \sum_{i=1}^{m} \sum_{k\in J_i(\nu)} h_{\nu_k}^\sigma(k) \cdot \frac{p_k}{s_{\nu_k}}$$
$$= \sum_{j=1}^{n} \frac{p_j}{s_{\sigma_j}} + \sum_{j=1}^{n} h_{\nu_j}^\sigma(j) \cdot \frac{p_j}{s_{\nu_j}} \ .$$

From Lemma 1 we know

$$\sum_{j=1}^{n} h_{\nu_j}^\sigma(j) \cdot \frac{p_j}{s_{\nu_j}} \le \sum_{j=1}^{n} \left(h_{\sigma_j}^\sigma(j) + 1 \right) \cdot \frac{p_j}{s_{\sigma_j}} = \sum_{j=1}^{n} C_j(\sigma) \ . \tag{8}$$

Also, the completion time of any job is at least its processing time on the machine it is scheduled on, so

$$\sum_{j=1}^{n} \frac{p_j}{s_{\sigma_j}} \leq \sum_{j=1}^{n} C_j(\sigma) \ . \tag{9}$$

Combining the above, we get

$$\sum_{j=1}^{n} C_j(\sigma_j, \nu_{-j}) \leq 2 \sum_{j=1}^{n} C_j(\sigma) \qquad \text{for all strategy profiles } \nu \ .$$

□

5 Lower Bound on the Price of Anarchy

In this Section we describe a parametric instance which has price of anarchy equal to $e/(e-1)$. The Nash equilibrium is the schedule with all jobs on the fastest machine (which is easily shown to be an upper bound on the quality of Nash equilibria in general, so in that sense, this is a worst case scenario).

Instance 1. *Let $\mathcal{I}$ be the parametric group of instances $I(s)$ that satisfy the following. $I(s)$ has m machines, one of which has speed $s > 1$ and all the other machines have speed 1. All speeds are integer. Furthermore, $I(s)$ has $n = m+s-1$ jobs, with length equal to*

$$p_j = \begin{cases} 1 & \text{if } 1 \leq j \leq s \\ x^{j-s} & \text{if } s+1 \leq j \leq n \end{cases} \ ,$$

where $x = s/(s-1)$.

Lemma 2. *Instances from $\mathcal{I}$ have a Nash equilibrium with all jobs on the fastest machine.*

Proof. In the schedule with all jobs in SPT order on the fastest machine, the completion time of a job $j < s$ is equal to

$$C_j = \sum_{k=1}^{j} \frac{p_k}{s} = \sum_{k=1}^{j} \frac{1}{s} = \frac{j}{s} \leq 1 \ . \tag{10}$$

For a job $j \geq s$, the completion time is equal to

$$C_j = \sum_{k=1}^{j} \frac{p_k}{s} = \frac{s-1}{s} + \sum_{k=s}^{j} \frac{\left(\frac{s}{s-1}\right)^{k-s}}{s}$$

$$= \frac{1}{s}\left(s-1+\sum_{k=0}^{j-s}\left(\frac{s}{s-1}\right)^k\right) = \frac{1}{s}\left(s-1+\frac{\left(\frac{s}{s-1}\right)^{j-s+1}-1}{\left(\frac{s}{s-1}\right)-1}\right)$$
$$= \frac{1}{s}\left(s-1+(s-1)\left(\frac{s}{s-1}\right)^{j-s+1}-(s-1)\right)$$
$$= \left(\frac{s}{s-1}\right)^{j-s} = p_j \ . \tag{11}$$

So the Nash equilibrium condition (3) holds, as all other machines have speed 1.

We use this to compute a lower bound on the price of anarchy.

Theorem 3. *The price of anarchy for the minsum related machine scheduling problem* $Q||\sum C_j$ *with SPT local scheduling rule is no less than* $e/(e-1) \approx 1.58$.

Proof. Consider instances $I(s)$ from $\mathcal{I}$ as defined above. In the optimal solution the s longest jobs are on the fastest machine. All other jobs are on a slow machine. So the objective value in the optimal solution is equal to

$$\begin{aligned}
\mathrm{OPT}(I(s)) &= \sum_{j=1}^{s-1} p_j + \sum_{j=s}^{n-s} p_j + \sum_{j=n-s+1}^{n}\sum_{k=n-s+1}^{j}\frac{p_k}{s} \\
&= \sum_{j=1}^{s-1} p_j + \sum_{j=s}^{n-s} x^{j-s} + \sum_{j=n-s+1}^{n}\sum_{k=n-s+1}^{j}\frac{x^{k-s}}{s} \\
&= \sum_{j=1}^{s-1} 1 + \sum_{j=0}^{n-2s} x^j + \sum_{j=n-s+1}^{n}\frac{1}{s}\left(\sum_{k=0}^{j-s}x^k - \sum_{k=0}^{n-2s}x^k\right) \\
&= s-1+(s-1)x^{n-2s+1}-(s-1)+\sum_{j=n-s+1}^{n}\left(x^{j-s}-x^{n-2s}\right) \\
&= (s-1)x^{n-2s+1} + \sum_{j=n-2s+1}^{n-s} x^j - \sum_{j=n-s+1}^{n} x^{n-2s} \\
&= (s-1)x^{n-2s+1} + (s-1)x^{n-s+1} - (s-1)x^{n-2s+1} - sx^{n-2s} \\
&= (s-1)x^{n-s+1} - (s-1)x^{n-2s+1} \ .
\end{aligned} \tag{12}$$

From Lemma 2 we know that the schedule with all jobs on the fastest machine is a Nash equilibrium. From (10) and (11) we know that the completion time of the jobs in this schedule is equal to

$$C_j = \begin{cases} \frac{j}{s} & \text{if } j \le s-1 \\ \left(\frac{s}{s-1}\right)^{j-s} & \text{otherwise} \end{cases} \ .$$

From this we compute the objective value in the Nash equilibrium

$$\begin{aligned}\mathrm{NE}(I(s)) &= \sum_{j=1}^{s-1} \frac{j}{s} + \sum_{j=s}^{n} x^{j-s} \\ &= \frac{s(s-1)}{2s} + \sum_{j=0}^{n-s} x^{j} \\ &= \frac{(s-1)}{2} + (s-1)x^{n-s+1} - (s-1) \\ &= (s-1)x^{n-s+1} - \frac{(s-1)}{2} \ . \end{aligned} \tag{13}$$

Combining (12) and (13) gives us the price of anarchy:

$$\begin{aligned}\mathrm{PoA}(I(s)) &= \frac{(s-1)x^{n-s+1} - \frac{(s-1)}{2}}{(s-1)x^{n-s+1} - (s-1)x^{n-2s+1}} \\ &= \frac{x^{n-s+1} - \frac{1}{2}}{x^{n-s+1} - x^{n-2s+1}} \\ &= \frac{x^{s} - \frac{1}{2}x^{-(n-2s+1)}}{x^{s} - 1} \\ &= \frac{\left(\frac{s}{s-1}\right)^{s} - \frac{1}{2}\left(\frac{s}{s-1}\right)^{-(n-2s+1)}}{\left(\frac{s}{s-1}\right)^{s} - 1} \ . \end{aligned} \tag{14}$$

Now, if we let n go to infinity, (14) becomes:

$$\lim_{n\to\infty} \mathrm{PoA}(I(s)) = \frac{\left(\frac{s}{s-1}\right)^{s}}{\left(\frac{s}{s-1}\right)^{s} - 1} \ , \tag{15}$$

and letting also s go to infinity, (15) goes to $e/(e-1) \approx 1.58$. □

6 Concluding Remarks

Of course, the question remains what the truth is concerning the price of anarchy for the considered problem, which we could bound in the interval $[1.58, 2]$. This gap may be due to the fact that the upper bound holds for more general equilibria than only pure Nash equilibria. While for the parametric instances from Theorem 3, scheduling all jobs on the fastest machine is even a dominant strategy equilibrium.

Note that it is indeed possible for mixed Nash equilibria to induce (significantly) worse price of anarchy than pure Nash equilibria. This can be seen by

the simple example of two identical machines with two identical jobs. For such an instance pure Nash equilibria are optimal solutions. However, the randomized schedule where each job choses each machine with equal probability of $1/2$ is a mixed Nash equilibrium, and yields an expected objective value $5/4$ times the optimal value.

All this leaves open the possibility that indeed 2 would be the true value of the robust price of anarchy, while the true value for the (pure) price of anarchy is $e/(e-1)$. We believe however that an improvement on the upper bound of 2 is possible, because either of the two terms that appears in our analysis in (8) and (9) can be equal to the optimum value, but we have not been able to construct instances where both inequalities are tight. Neither have we been able (so far) to offset the two terms against each other, which might be a feasible approach for improving our analysis for the upper bound.

References

1. Aumann, R.J.: Subjectivity and correlation in randomized strategies. J. Math. Econom. 1(1), 67–96 (1974)
2. Azar, Y., Jain, K., Mirrokni, V.: (Almost) optimal coordination mechanisms for unrelated machine scheduling. In: Proceedings 19th SODA, pp. 323–332. ACM/SIAM (2008)
3. Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination mechanisms. Theoret. Comput. Sci. 410(36), 3327–3336 (2009)
4. Cole, R., Correa, J.R., Gkatzelis, V., Mirrokni, V., Olver, N.: Inner Product Spaces for MinSum Coordination Mechanisms. In: Proceedings 43rd STOC, pp. 539–548. ACM (2011)
5. Conway, R.W., Maxwell, W.L., Miller, L.W.: Theory of Scheduling. Addison-Wesley Publishing Co., Reading (1967)
6. Correa, J., Queyranne, M.: Efficiency of Equilibria in Restricted Uniform Machine Scheduling with MINSUM Social Cost (manuscript) (2010)
7. Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. ACM Trans. Algorithms 3(1), Art. 4, 17 (2007)
8. Graham, R., Lawler, E., Lenstra, J., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5(2), 287–326 (1979)
9. Heydenreich, B., Müller, R., Uetz, M.: Games and mechanism design in machine scheduling - An introduction. Production and Operations Management 16(4), 437–454 (2007)
10. Horowitz, E., Sahni, S.: Exact and approximate algorithms for scheduling nonidentical processors. Journal of the ACM 23(2), 317–327 (1976)
11. Ibarra, O., Kim, C.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM 24(2), 280–289 (1977)
12. Immorlica, N., Li, L., Mirrokni, V.S., Schulz, A.S.: Coordination mechanisms for selfish scheduling. Theoret. Comput. Sci. 410(17), 1589–1598 (2009)
13. Koutsoupias, E., Papadimitriou, C.: Worst-Case Equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)

14. Myerson, R.B.: Utilitarianism, egalitarianism, and the timing effect in social choice problems. Econometrica 49(4), 883–897 (1981)
15. Myerson, R.B.: Game theory - Analysis of conflict. Harvard University Press, Cambridge (1991)
16. Papadimitriou, C.: Algorithms, games, and the internet. In: Proceedings 33rd STOC, pp. 749–753. ACM (2001)
17. Roughgarden, T.: Intrinsic robustness of the price of anarchy. In: Proceedings 41st STOC, pp. 513–522. ACM (2009)
18. Yu, L., She, K., Gong, H., Yu, C.: Price of anarchy in parallel processing. Inform. Process. Lett. 110(8-9), 288–293 (2010)

Author Index