Proceedings of the
12th Workshop on Models and Algorithms
for Planning and Scheduling Problems

MAPSP 2015

La Roche-en-Ardenne, Belgium

8-12 June 2015

Editors: Alberto Marchetti-Spaccamela, Yves Crama, Dries Goossens, Roel Leus, Michaël Schyns, Frits Spieksma

ISBN: 9789081409971

Preface

This volume contains abstracts of talks presented at the 12th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2015), held from June 8 to June 12, 2015, in La Roche-en-Ardenne, Belgium.

MAPSP is a biennial workshop dedicated to all theoretical and practical aspects of scheduling, planning, and timetabling. Previous MAPSP meetings have been held in Menaggio, Italy (1993), Wernigerode, Germany (1995), Cambridge, UK (1997), Renesse, Netherlands (1999), Aussois, France (2001), Aussois, France (2003), Siena, Italy (2005), Istanbul, Turkey (2007), Kerkrade, Netherlands (2009), Nymburk, Czech Republic (2011) and Pont à Mousson, France (2013).

The abstracts in this volume include 5 invited talks by Onno Boxma, Michel Goemans, Willem-Jan van Hoeve, Rolf Niedermeier, Stephan Westphal plus 88 contributed talks selected out of 95 submissions. Each submission was reviewed by at least three program committee members.

Submitted papers are presented in three parallel tracks. It is well known that parallel sessions suffer from the phenomenon that a participant's favorite talks are scheduled simultaneously in different sessions. In setting up this year's MAPSP schedule, the organizers, after surveying the participants' preferences, constructed a schedule that maximizes total attendance and satisfaction.

We thank all sponsors of MAPSP 2015 for their generous support. The conference is supported by KU Leuven, the University of Liège, Ghent University, the Belgian Science Policy Office (IAP project COMEX), and the Belgian Society for Operations Research (ORBEL). Further, we are very grateful to the companies N-SIDE, OM-PARTNERS, and ORTEC for sponsoring MAPSP 2015.

We thank the referees, and the members of the organizing committee for making MAPSP 2015 possible.

On behalf of the Program Committee of MAPSP2015,

Alberto Marchetti-Spaccamela (Chair).

Contents

Invited contributions

Submitted contributions

High multiplicity scheduling problems with a constant number of job types are polynomially solvable

Prof. Michel Goemans
Massachusetts Institute of Technology

Abstract

Many scheduling problems can also be considered in a high-multiplicity setting in which there are different types of jobs and all the jobs of the same type share the same parameters (processing times, release dates, deadlines, etc.). The difficulty arises from the fact that the number of jobs of a certain type is given in binary encoding. We develop an approach to solve optimally and in polynomial time a class of high multiplicity scheduling problems when the number of job types is constant. This applies for example to the Multiprocessor Scheduling Problem (even with release dates and deadlines, with or without preemption) for which the previous best result due to McCormick et al. could only deal with two job types (and no other constraints), instead of an arbitrary constant number. Our approach is geometric, and the algorithm in fact solves the following problem for constant d in polynomial time: given two d-dimensional polytopes P and Q, find the smallest number of integer points in P whose sum lies in Q.

This is joint work with Thomas Rothvoss (U. Washington).

Decision diagrams for optimization and scheduling

Prof. Willem-Jan van Hoeve
Tepper School of Business, Carnegie Mellon University

Abstract

In this talk, we will present an overview of the recent successful application of decision diagrams to combinatorial optimization. In particular, we will discuss how decision diagrams of limited size can be used as discrete relaxation in the context of integer optimization and constraint-based scheduling. See http://www.andrew.cmu.edu/user/vanhoeve/mdd/ for more information.

A parameterized complexity view on scheduling and planning problems

Prof. Rolf Niedermeier
TU Berlin

Abstract

The talk discusses results concerning the parameterized computational complexity of NP-hard problems in the context of scheduling and planning. In particular, we will present algorithms for problems such as Job Interval Selection (modeled as Colorful Independent Set problem in graphs) and Arc Routing (modeled as Eulerian extension problems in graphs).

Asymmetric sports schedules for the German Basketball League

Prof. Stephan Westphal
TU Clausthal

Abstract

In this talk we discuss the problem of finding an optimal schedule for the German Basketball League (BBL). The main problem is that most of the games take place in multi-purpose arenas which are also used for a wide variety of other events as well and are thus not always available. Furthermore, the total number of successive home or away-games has to be minimized, the most interesting games have to be assigned to TV broadcasting slots, the total distance driven on a newly established derby day has to be minimized and several wishes for home or away-games of teams or specific encounters have to be taken care of.

We present several algorithmic approaches and show how good the different models fit the needs of the BBL. In this process, we prove that the classic models which have been applied by the BBL and are still applied by lots of other leagues as well are too limited to meet the requirements. We show that there are no canonical schedules which have the desired properties. We could also prove that there are no mirrored plans which do much better and could thus convince the BBL to apply non-mirrored schedules to their league for the first time in their history.

As all of the requirements mentioned above are very typical for sports leagues in general, the presented approaches are not limited to the considered example of the BBL but can be applied to a whole variety of other sports leagues as well. Especially, since lots of other leagues still use mirrored schedules, it is quite thinkable that many of their problems can be solved in a better way by making use of our concepts as well. Our method has been implemented in a scheduling software which we developed for the German Basketball League and which was first applied to generate the plan for the season 2011/2012.

Analysis, optimization and scheduling of polling systems

Prof. Onno Boxma
Eindhoven University of Technology

Abstract

A polling model is a queueing model consisting of several queues, which are cyclically visited by a server. The server visits the queues according to some discipline, like 1-limited (serve at most one customer in a visit) or exhaustive (serve a queue until it has become empty). Polling models find many applications in computer-communications, and also in other areas like maintenance, production-inventory systems, and signalized traffic intersections.

The first part of the talk contains a global introduction to polling systems, and a review of some of their key properties. In the second part of the talk I'd like to touch upon some studies with Urtzi Ayesta, Sem Borst, Josine Bruin, Jan-Pieter Dorsman, Brian Fralix, Maaike Verloop, Maria Vlasiou, Adam Wierman and Erik Winands on various optimization an scheduling problems in polling systems.

Minimizing maximum flow-time on related machines

Nikhil Bansal * Bouke Cloostermans (Speaker) †

1 Introduction

We consider the problem $Q \,|\, \textit{on-line}; r_j \,|\, F_{\max}$; i.e., minimizing the maximum flow time on related machines. There are m machines where machine i has speed s_i. Jobs arrive online, where job j has release date r_j and size p_j. The scheduler knows p_j upon arrival of j, and processing j on machine i takes p_j/s_i time. The goal is to find a non-migratory schedule that minimizes $\max_j(C_j - r_j)$, where C_j is the completion time of j.

For the easier identical machines setting, the *Greedy* algorithm that schedules a job on the least loaded machine, is 3-competitive [2]. On the other hand, for the more general unrelated machines setting, a lower bound of $\Omega(m)$ is known [1]. Interestingly the status of the related machine setting is still unclear. No $O(1)$ competitive algorithm is known (even in the offline case), and no non-constant lower bound on the competitive ratio is known either.

One difficulty is that the two most natural algorithms, *Slow-fit* and *Greedy*, are not $O(1)$-competitive. Recall that *Slow-fit* always schedules a job on the slowest possible machine (provided the load stays under some threhold). We sketch these lower bounds in Section 2. Roughly speaking, slow-fit is too conservative and may waste the capacity on fast machines, while *Greedy* may schedule too many small jobs on fast machines, which is problematic when large jobs arrive later. Our main result is the following.

Theorem 1. *There is an 18-competitive algorithm for the problem* $Q \,|\, \textit{on-line}; r_j \,|\, F_{\max}$.

This also gives the first $O(1)$-approximation for the offline version of the problem. Our algorithm is a hybrid between *Slow-fit* and *Greedy* and carefully combines the features of both. More precisely, it does slow-fit using two thresholds, which ensures that fast machines remain busy but do not get unnecessarily loaded with small jobs. The analysis is based on defining a careful invariant on the total load over certain subsets of machines, and showing inductively that this invariant is maintained over time.

2 Two natural algorithms

Slow-fit: Given a threshold F_{opt} (the current guess on optimum), Slow-fit schedules jobs on the slowest possible machine while keeping the load below F_{opt}. If the jobs cannot be feasibly scheduled the threshold in doubled.

Lemma 2. Slow-fit *has a competitive ratio of* $\Omega(m)$.

*n.bansal@tue.nl. Eindhoven University of Technology
†b.cloostermans@tue.nl. Eindhoven University of Technology

Proof. We describe an instance where the threshold F_{opt} keeps doubling until it reaches m even though the optimum solution has value 2. There are m identical machines (but we arbitrarily order them from slow to fast). Next, we assume that $F_{\text{opt}} \geq 2$, which can be achieved by giving $2m$ unit-size jobs initially at $t = 0$.

At each time step $t \geq 2$, m unit-length jobs arrive. As Slow-fit will not use all machines initially, there will be a time t_0 at which the machines $1, \ldots, m-1$ have F_{opt} load and machine m is empty. At time $t_0 + 1$, when the $m - 1$ machines have $F_{\text{opt}} - 1$ pending jobs, we release $2m$ unit-size jobs. As there is only $m - 1 + F_{\text{opt}}$ total capacity available, these jobs cannot be scheduled feasibly if $F_{\text{opt}} \leq m$. On the other hand, the optimum spread the jobs over all machines at each time and achieve value 2. □

Greedy: When a job arrives, the *Greedy* algorithm schedules it on the machine such that its flow time is minimized. Ties are broken arbitrarily.

Lemma 3. Greedy *has a competitive ratio of* $\Omega(\log m)$.

Proof. Consider an instance where we have k groups of machines where group G_i contains 2^{2k-2i} machines of speed 2^i. Note that the total processing power in group G_i is equal to $S_i = 2^{2k-i}$. The processing power of groups $i, \ldots, k$ combined is thus equal to $P_i = \sum_{i'=i}^{k} 2^{2k-i'} \leq 2S_i$

We receive k sets of jobs, all at time 0, but in order. For all $i = 1, \ldots, k$, set J_i contains 2^{2k-2i} jobs of size 2^i. Again, note that the total size of jobs in set J_i is equal to 2^{2k-i}. *Greedy* will spread jobs from set i over groups $i, \ldots, k$. Group k (containing only a single machine of speed 2^k) will receive a $S_k/P_i \geq \frac{1}{2} S_k/S_i = 2^{-k+i-1}$ fraction of these jobs. This means group k receives $\sum_{i=1}^{k} 2^{2k-i} 2^{-k+i-1} = k2^{k-1}$ work. Since group k has a single machine of speed 2^k, finishing these jobs takes $\Omega(k)$ time.

However, optimum can schedule the i-th batch of jobs on group i machines, incurring a maximum load of 1 (i.e., it does Slow-fit with threshold 1).

□

3 A two threshold algorithm

Using the doubling trick, assume our algorithm knows the optimal solution F_{opt}. Jobs with release dates in the interval $(3(k-1)F_{\text{opt}}, 3kF_{\text{opt}}]$ are scheduled at time $3kF_{\text{opt}}$. Every time a batch of job is scheduled, this batch is partitioned into sets $J_1, \ldots, J_m$. Each job j is assigned to the first set J_i such that $p_j \leq s_i \cdot F_{\text{opt}}$ (the slowest machine on which OPT could schedule j). Then sets J_i are scheduled in decreasing order of i starting from $i = m$. When scheduling J_i, jobs are slow-fit onto machines $i, \ldots, m$. First with threshold $4F_{\text{opt}}$ and if no room remains under $4F_{\text{opt}}$, then under $6F_{\text{opt}}$.

The analysis we show here is simplified a lot. We compare our schedule to the schedule of a restricted optimum, which also schedules jobs in batches. To analyze the performance of our algorithm we introduce $W_i(t)$, the total work on machines $i, \ldots, m$ at time t, *before* the jobs in the current batch are scheduled. We also use $W_i^{OPT}(t)$, a similar quantity for a restricted optimal schedule. Our goal will be to prove the following invariant holds for all i and scheduling times t (so $t = 0, 3F_{\text{opt}}, 6F_{\text{opt}}, \ldots$):

$$W_i(t) \leq W_i^{OPT}(t) + \sum_{i'=i}^{m} s_{i'} \cdot F_{\text{opt}}. \tag{1}$$

To this end, we introduce *separated* machines.

Definition 4. *Machines i and i' $(i < i')$ are* separated *at time t if no jobs from $J_1, \ldots, J_i$ were scheduled onto machines $i', \ldots, m$ at time t.*

The proof uses two key lemmas which exploit our schedule's properties.

Lemma 5. *Let $L_i(t)$ be the total size of jobs on machine i at time t* after *jobs have been scheduled. For any machine i, at any time t, either $L_i(t) \leq 4F_{opt} \cdot s_i$, or $L_{i'}(t) \geq 3F_{opt} \cdot s_i$ for all $i' > i$.*

Lemma 6. *If machines $i-1$ and i are separated at time t, then (1) implies*

$$W_i(t + 3F_{opt}) \leq W_i^{OPT}(t + 3F_{opt}) + \sum_{i'=i}^{m} s_{i'} \cdot F_{opt}. \tag{2}$$

Sketch of proof of Theorem 1. We use induction over i and t. When proving (1) holds for some pair (i, t) we assume (1) holds for all (i, t') with $t' < t$ and for all (i', t) with $i' > i$. Lemma 5 ensures we process jobs quickly enough. For the scheduling part, we consider three cases.

No jobs from $J_1, \ldots, J_{i-1}$ were scheduled onto machines $i, \ldots, m$: Machines $i-1$ and i are separated, thus (2) follows immediately from Lemma 6.

Jobs from $J_1, \ldots, J_{i-1}$ were only scheduled onto machines $i, \ldots, m$ under $4F_{\mathbf{opt}}$: Consider $i_{max} \geq i$, the smallest index such that machines i and i_{max} are separated. For machines i' slower than i_{max} we can show $L_{i'}(t) \leq 4F_{\text{opt}} \cdot s_i$. Combining this observation with Lemma 6 gives the desired result.

Some job j from $J_1, \ldots, J_{i-1}$ was scheduled onto machines $i, \ldots, m$ under $6F_{\mathbf{opt}}$: Consider $i_{min} \leq i$, the largest index such that machines $i_{min}, \ldots, i-1$ have load at least $5F_{\text{opt}}$ and i_{min} has load at most $5F_{\text{opt}}$. We can show machines $i_{min} - 1$ and i_{min} are separated. In the restricted optimal schedule, no machines are ever loaded above $4F_{\text{opt}}$. In particular, this holds for machines $i_{min}, \ldots, i-1$. Combining this observation with Lemma 6 gives the desired result.

Plugging in $i = m$ into (1) implies our algorithm never schedules above $5F_{\text{opt}}$ on machine m and therefore always succeeds. In total each job spends at most $3F_{\text{opt}}$ time waiting to be assigned, and then another $6F_{\text{opt}}$ time in the queue on its designated machine. Another factor 2 in the competitive ratio is incurred because our guess of F_{opt} can be off by a factor 2. □

References

[1] S Anand, Karl Bringmann, Tobias Friedrich, Naveen Garg, and Amit Kumar. Minimizing maximum (weighted) flow-time on related and unrelated machines. In *Automata, Languages, and Programming*, pages 13–24. Springer, 2013.

[2] Michael A Bender, Soumen Chakrabarti, and Sambavi Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *SODA*, volume 98, pages 270–279, 1998.

On the Online Machine Minimization Problem*

Lin Chen (Speaker) Nicole Megow Kevin Schewior†

1 Introduction

We consider the fundamental problem of minimizing the number of machines that is necessary for feasibly scheduling jobs with release dates and hard deadlines. We consider the online variant of this problem in which every job becomes known to the online algorithm only at its release date. We denote this problem as *online machine minimization problem*. We also consider a *semi-online* variant in which it is known in advance that the instance released by the adversary could be scheduled offline on m machines.

Known results. The preemptive semi-online machine minimization problem, in which the optimal number of machines is known in advance, has been investigated extensively by Phillips et al. [4], and there have hardly been any improvements since then. Phillips et al. show a general lower bound of $\frac{5}{4}$ and leave a huge gap to the upper bound $\mathcal{O}(\log \frac{p_{\max}}{p_{\min}})$ on the competitive ratio for the so-called *Least Laxity First* (LLF) Algorithm. Not so surprisingly, they also rule out that the *Earliest Deadline First* (EDF) Algorithm could improve on the performance of LLF; indeed they show a lower bound of $\Omega(\frac{p_{\max}}{p_{\min}})$. It is a wide open question if preemptive semi-online machine minimization admits a constant-competitive algorithm, or admits an algorithm which has a competitive ratio independent of the job set, i.e., an $f(m)$-competitive algorithm for some function f. It is not even known whether a constant-competitive algorithm exists for $m = 2$.

The non-preemptive problem is considerably harder than the preemptive problem. If the set of jobs arrive online over time, then no algorithm can achieve a constant or a competitive ratio sublinear in the number of jobs [5]. However, relevant special cases admit online algorithms with small constant worst-case guarantees. The problem with unit processing times was studied in a series of papers [3, 5, 2] and implicitly in the context of energy-minimization in [1]. It has been shown that an optimal online algorithm has the exact competitive ratio $e \approx 2.72$ [1, 2]. For non-preemptive scheduling of jobs with equal deadlines, an upper bound of 16 is given in [2].

Our results Our main contribution is a new preemptive online algorithm with a competitive ratio $f(m)$ which is independent of the number of jobs.

*Technische Universität Berlin, Institut für Mathematik, Berlin, Germany. Email: {nmegow, schewior}@math.tu-berlin.de. Supported by the German Science Foundation (DFG) under contract ME 3825/1.

†This author was supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

Theorem 1 *There exists an $\mathcal{O}(m^2 \log m)$-competitive algorithm for the preemptive (semi-)online machine minimization problem.*

Our algorithm is the first improvement upon the $\mathcal{O}(\log \frac{p_{\max}}{p_{\min}})$-competitive algorithm by Phillips et al. [4]. Specifically, if m is a constant, our algorithm is a constant-competitive algorithm.

We achieve this algorithm by studying two complementary special cases of the problem, namely, laminar instances and agreeable instances. Let $J_j = [r_j, d_j)$ denote the processing interval of job j, where r_j and d_j are its release date and deadline. In a laminar instance, for any two jobs i and j it holds that either $J_i \subseteq J_j$, or $J_j \subseteq J_i$, or $J_i \cap J_j = \emptyset$. In an agreeable instance, however, for any two jobs i and j, $r_i < r_j$ implies that $d_i \leq d_j$. We provide an $\mathcal{O}(\log m)$-competitive algorithm for laminar instances, and an $\mathcal{O}(1)$-competitive algorithm for agreeable instances. Extending both techniques, we derive an $\mathcal{O}(m^2 \log m)$-competitive algorithm for the general problem.

Interestingly, our $\mathcal{O}(1)$-competitive algorithm for agreeable instances actually produces a non-preemptive schedule. Thus we have the following theorem.

Theorem 2 *There exists an $\mathcal{O}(1)$-competitive algorithm for the non-preemptive online machine minimization problem when jobs have agreeable deadlines. The non-preemptive solution is $\mathcal{O}(1)$-competitive even within the preemptive problem setting.*

This result is of its own interest as the lower bound of n for the general non-preemptive (semi)-online machine minimization problem [5] even holds for laminar instances.

We also obtain algorithms with small constant competitive ratios for more restricted special cases such as equal deadlines or equal processing times.

2 Overview of the main algorithm

We first observe that, despite the fact that EDF has a competitive ratio of $\Omega(p_{max}/p_{min})$ for the general problem, it is, however, an $\mathcal{O}(1)$-competitive algorithm if every job is *loose* in the sense that $p_j \leq \alpha(d_j - r_j)$ holds for some constant $\alpha < 1$. Hence, we may restrict our attention to tight jobs, i.e., $p_j > \alpha(d_j - r_j)$. We consider two complementary subclasses, namely laminar instances and agreeable instances, and then we show how to extend the techniques to the general problem.

An $\mathcal{O}(\log m)$-competitive algorithm for laminar instances. The basic idea underlying the algorithm is a "diagonal packing" procedure. Notice that the laminar structure of the input allows us to derive a partial order of jobs based on inclusion, i.e., $j \prec i$ means $J_j \subseteq J_i$, and we denote job i as the dominating job of j. Once our algorithm decides to preempt job i in favor of job j, it fully preempts i during the interval $[r_j, d_j)$. Thus, later on, when another job $j' \prec j$ is released, we only need to check job j to see if j should be preempted in favor of j', and job i is no longer considered during $[r_j, d_j)$.

Let Γ be the number of machines our algorithm uses. Once job j is released, either there is a machine empty during $[r_j, d_j)$ and we assign j to it, or on each machine there exists some j' such that $j \prec j'$, and we need to preempt it to make room for job j. According to our previous description, among all the dominating jobs of j on every machine, we only need to consider the smallest (in terms of the partial order) one. Hence, in total, we need to consider Γ jobs, one on each machine. Due to the

partial order we may list these Γ jobs as $j_1 \prec j_2 \prec \cdots \prec j_\Gamma$. We have to select one of them to preempt during $[r_j, d_j)$. Here, a natural idea would be to view the laxity ℓ_i of job i (i.e., $\ell_i = d_i - r_i - p_i$) as a knapsack of volume ℓ_i. Once job i is preempted in favor of some other job, this knapsack is filled up to some extent, rendering a smaller remaining volume. Unlike LLF, which preempts those jobs with largest such remaining volumes, our algorithm equally partitions the knapsack into Γ sub-knapsacks, each of volume ℓ_i/Γ. To decide which job, among j_1 to j_Γ, should be preempted in favor of j, the algorithm checks the first sub-knapsack of j_1, the 2nd sub-knapsack of j_2, ..., the Γ-th sub-knapsack of j_Γ. If it decides to preempt j_h, then only the h-th sub-knapsack of j_h is filled, and its remaining volume is decreased by $|J_j| = d_j - r_j$. We prove that, for $\Gamma = \mathcal{O}(m \log m)$, we can always find some h among these Γ sub-knapsacks such that the h-th sub-knapsack of j_h has enough remaining volume.

An $\mathcal{O}(1)$-competitive algorithm for agreeable instances. For agreeable instances, we observe that the following simple algorithm is already $\mathcal{O}(1)$-competitive for α-tight jobs: We always process job j during $[r_j + 1/2 \cdot \ell_j, d_j - 1/2 \cdot \ell_j)$. We also prove that a non-preemptive version of EDF achieves a competitive ratio of $\mathcal{O}(1)$ for α-loose jobs if they have agreeable deadlines. Hence, we derive a non-preemptive schedule within an factor $\mathcal{O}(1)$ of the preemptive offline optimum.

An $\mathcal{O}(m^2 \log m)$-competitive algorithm for the general problem. To handle the general problem, we combine both algorithms above in a subtle way. Our algorithm has two steps. First, it partitions jobs in an online way into Γ groups such that, in each group h, if we let $U(i,h)$ denote the jobs that are dominated by i, then $|\cup_{j \in U(i,h)} [r_j, d_j)| \leq \theta \ell_j$ for some constant $\theta < 1$. Here we make use of the "diagonal packing" procedure in a more sophisticated way as jobs no longer form a laminar structure. Then, again online, the algorithm schedules jobs of each group on $\mathcal{O}(m)$ machines. Here a job i of group h is preempted during $[r_i, r_i + (1-\theta)\ell_i/2) \cup [d_i - (1-\theta)\ell_i/2, d_i)$, and also during $\cup_{j \in U(i,h)} [r_j, d_j)$. Hence, it uses $\mathcal{O}(m\Gamma)$ machines in total. We prove that the partition procedure is successful for $\Gamma = \mathcal{O}(m^2 \log m)$, which implies the claim.

References

[1] N. Bansal, T. Kimbrel, and K. Pruhs, *Speed scaling to manage energy and temperature*, J. ACM, 54 (2007).

[2] N. R. Devanur, K. Makarychev, D. Panigrahi, and G. Yaroslavtsev, *Online algorithms for machine minimization*, CoRR, abs/1403.0486 (2014).

[3] M.-J. Kao, J.-J. Chen, I. Rutter, and D. Wagner, *Competitive design and analysis for machine-minimizing job scheduling problem*, in Proc. of the 23rd International Symposium (ISAAC 2012), vol. 7676 of LNCS, 2012, pp. 75–84.

[4] C. A. Phillips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, Algorithmica, 32 (2002), pp. 163–200.

[5] B. Saha, *Renting a cloud*, in Proc. of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013), vol. 24 of LIPIcs, 2013, pp. 437–448.

Improved Online Algorithms for the Machine Covering Problem with Bounded Migration

Waldo Gálvez (Speaker) * José A. Soto * José Verschae †

1 Introduction

We consider an online version of the *machine covering* problem on identical machines. An instance of our problem is determined by a set of m identical parallel machines M, and a set of jobs J that is revealed incrementally one by one. If p_j denotes the processing time of job $j \in J$, the *load* of a machine is the sum of the processing time of jobs assigned to it. The scheduler must choose a machine to process each new job, with the goal of maintaining a solution that balances the load of the machines. More precisely, we seek to maximize the load of the least loaded machine.

Unlike classic online problems in which decisions are irrevocable, we focus on a dynamic model introduced by Sanders et al. [3] where at the arrival of a job j we are allowed to migrate some previously assigned jobs. However, we should not migrate jobs indiscriminately: The total processing time of the migrated jobs must be bounded by $\beta \cdot p_j$. The parameter β is called the *migration factor* of the algorithm and measures the robustness of the constructed solutions. This problem is motivated by the following scenario: consider a system consisting of m identical machines that is *alive* only when all the machines are alive. In order to keep a machine alive, it requires resources. Resources of various sizes arrive one after another. Each one must be assigned immediately upon arrival to one of the machines. It is not acceptable to completely reconfigure the system when a new resource arrive, but rather a "proportionate response" is expected. That is, if a resource of size x arrives, it is adequate to move a total amount of resources of the same order of magnitude, but no more. The goal is to keep the system alive as long as possible. See [3] for applications of this model to Storage Area Networks.

The main motivation for our work is to understand the tradeoff between competitive ratio and migration. It is known that no algorithm with constant migration can achieve a competitive ratio of $\left(\frac{20}{19} - \varepsilon\right)$ [4], while the best known algorithm with constant migration factor is 2-competitive [3]. This work contributes to diminish this gap by providing a $(\frac{4}{3} + \varepsilon)$-competitive algorithm with constant migration factor.

*{wgalvez,jsoto}@dim.uchile.cl. Departamento de Ingeniería Matemática, Universidad de Chile, Santiago, Chile.

†jverschae@uc.cl. Departamento de Matemáticas y Departamento de Ingeniería Industrial y de Sistemas, Pontificia Universidad Católica de Chile, Santiago, Chile.

2 Previous Work

It is easy to see that the offline version is NP-complete in the strong sense by a simple reduction from 3-Partition. Woeginger [5] designs a PTAS for this offline problem and shows that the greedy algorithm is m-competitive, while no deterministic algorithm can achieve a smaller competitive ratio. For the offline version of the problem, one of the classic algorithms found in the literature is the *Longest Processing Time first* (LPT) rule, where the jobs are sorted from bigger to smaller and assigned iteratively to the least loaded machine. This algorithm is a $\frac{4}{3}$-approximation, with this ratio being asymptotically tight [1, 2]. For the makespan objective, Sanders et al. [3] develop a *robust PTAS*, i.e., a $(1+\varepsilon)$-approximation algorithm using a migration factor of $2^{poly(\frac{1}{\varepsilon})}$. For this result the processing time of the jobs are rounded in order to add symmetry to the solutions, which is crucial for the algorithm. It is open whether the migration factor can be made polynomial in $\frac{1}{\varepsilon}$. For the machine covering version, Skutella and Verschae [3] show that there exists a robust PTAS in a setting where the migration is measured in an amortized manner.

3 Our Results

We present two algorithms that provide different tradeoffs between competitiveness and robustness. Both algorithms are based on adapting the LPT rule to the online setting.

Let $J = \{1, \ldots, n\}$ be the current set of jobs, $L_{\min}$ the load of the least loaded machine in the solution given by the algorithm, and OPT the optimal objective value for J. We also define the load of the machine $i \in M$ in the schedule S as $\text{load}_S(i) := \sum_{j \in J_i} p_j$, where J_i is the set of jobs assigned to machine i.

For simplicity, suppose that J is sorted such that $p_1 \geq p_2 \geq \cdots \geq p_n$. Let $j^* \in J$ be the first job that LPT assigns to a machine with two jobs. Our algorithm will maintain the following invariants: The set $J^* := \{j \in J : j < j^*, p_j \in [\varepsilon \text{ OPT}, \text{OPT}]\}$ is assigned using LPT, and the load of the machines with a job $p_j < \frac{1}{3}\text{OPT}$ is at most $L_{min} + \frac{1}{3}\text{OPT}$. We show that any schedule satisfying these properties is a $\frac{3}{2}$-approximation.

This invariant can be maintained at each iteration in an approximated way with migration factor $O\left(\frac{1}{\varepsilon}\right)$ in the following way: we round down the size of each job to powers of $(1+\varepsilon)$, and approximate OPT using the PTAS from [5]. If the new job is not in J^*, then we add it greedily and the property holds. If the new job is in J^*, then the LPT structure can be reconstructed with low migration since there are at most $\log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)$ different types of jobs in J^*, and we need to move at most one job per class, obtaining a total migrated load of $O\left(\frac{1}{\varepsilon}\right)$. These ideas yield the following result.

Theorem 1 *For any $\varepsilon > 0$, there exists a $\left(\frac{3}{2}+\varepsilon\right)$-competitive algorithm with a migration factor of $O\left(\frac{1}{\varepsilon}\right)$.*

Our main result is a more refined approximate online version of LPT, with the same approximation ratio up to an additive epsilon term. For simplicity, let us assume that all jobs sizes are in $[\varepsilon\text{OPT}, \text{OPT}]$: larger jobs can be rounded down to OPT, smaller jobs in J can be grouped into jobs of size $O(\varepsilon\text{OPT})$, and if the new arriving job is smaller we can simply assign it greedily. The algorithm first round down the processing time of each job j to the nearest multiple of $\varepsilon^2\text{OPT}$. Thus, the load of each machine is also

a multiple of ε^2OPT, and therefore there are only $O(\frac{1}{\varepsilon^2})$ possible different load values. Roughly speaking, this increases the number of equivalent LPT schedules: in each step of LPT, we will find more least loaded machines where to assign a job. This symmetry is helpful for designing online algorithms because we will have a larger pool of equivalent available solutions to choose from, one of which will be close to our current solution.

To exploit this property we define the *load profile* $L(S)$ of a schedule S as the vector $(\text{load}_S(1), \ldots, \text{load}_S(|M|))$ where the entries are sorted in non-decreasing order. Then the following useful property follows: If S and S' are LPT schedules for J and $J \cup \{n+1\}$, respectively, then $L(S) \leq L(S')$ (i.e., less or equal coordinate-wise). Combining this property and the rounding procedure, it is possible to show that the load profile of S and S' differ in at most $\frac{1}{\varepsilon^2}$ coordinates. This follows mainly because if two machines have different loads, this difference is not arbitrarily small (is bounded below by ε^2OPT). Based on that we are able to construct an LPT schedule for $J \cup \{n+1\}$ in several phases, where in phase $k = \frac{1}{\varepsilon^2}, \frac{1}{\varepsilon^2} - 1, \ldots, \frac{1}{\varepsilon}$ we assign the set of jobs with processing time $k \cdot \varepsilon^2$OPT. At the end of each phase, we make sure that the partial new schedule has the same machine loads as the original one, except for at most $\frac{1}{\varepsilon^2}$ machines. This can be done migrating $O(\frac{1}{\varepsilon^2}\text{OPT})$ load at each phase, and since there are $O(\frac{1}{\varepsilon^2})$ phases and the new job has processing time at least εOPT, the migration factor used is $O(\frac{1}{\varepsilon^5})$.

In order to implement these ideas we need to deal with many technical problems. In particular, the value of OPT may change at each iteration, which will change the (rounded) processing times. This might break the desired structure. This can be solved by a more careful rounding procedure where the rounded processing times do not change when a new job arrives.

Theorem 2 *For any $\varepsilon > 0$, there exists a $\left(\frac{4}{3} + \varepsilon\right)$-competitive algorithm with a migration factor of $O\left(\frac{1}{\varepsilon^5}\right)$.*

Finally, a related question consists of analyzing local search algorithms. A natural local search algorithm migrates a job to the least loaded machine if this move does not decrease the objective function. We show the following result.

Theorem 3 *The approximation ratio of the local search algorithm lies in the interval* $[1.691, 1.75]$.

References

[1] J. Csirik, H. Kellerer, and G. Woeginger. The exact lpt-bound for maximizing the minimum completion time. *Oper. Res. Lett.*, 11:281–287, 1992.

[2] B. Deuermeyer, D. Friesen, and M. Langston. Scheduling to maximize the minimum processor finish time in a multiprocessor system. *SIJADM*, 3:190–196, 1982.

[3] P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34:481–498, 2009.

[4] M. Skutella and J. Verschae. A robust ptas for machine covering and packing. In *ESA (1)*, volume 6346 of *LNCS*, pages 36–47, 2010.

[5] G. J. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Oper. Res. Lett.*, 20:149–154, 1997.

Delay prediction models for robust airline resource scheduling

Lucian Ionescu (Speaker) * Natalia Kliewer †

February 16th, 2015

1 Introduction

Cost-optimized airline resource schedules often imply a lack of delay tolerance in case of unforeseen disruptions, e.g. late check-ins, technical defects or airport and airspace congestion. In the context of regular operations, we distinguish between primary and secondary delay. Delay that occurs due to exogenous disruptions is called primary delay. By contrast, secondary delay emerges from propagation effects in resource networks, e.g. for aircraft or crews. It depends on scheduling decisions and can be avoided by robust scheduling. On the one hand stability of resource schedules can be increased by incorporating buffer times between tasks in order to absorb delay. On the other hand, the flexibility can be increased by swap opportunities for resources that can be used by the operations control in case of delay.

Nevertheless, there is a trade-off between the goals of planned cost minimization and schedule robustness. Scheduling approaches are efficient, if the robustness can be increased at low increase of the planned costs. A wide range of sophisticated robust scheduling approaches has been developed in recent years, see for example [1], [2] or [3].

For the evaluation of scheduling strategies, the robustness of generated resource schedules has to be measured. The most relevant figure is the on-time performance, i.e. the percentage of flights departing without delay. However, exogenous primary delay cannot be influenced by scheduling. Therefore, the relevant figure to consider is the secondary delay propagated due to insufficient buffer times between flights connected by the same resource. In consequence, a schedule A is more robust than schedule B if the amount of propagated secondary delay is less in schedule A than in B. While secondary delay can be determined for example by simulating delay propagation, primary delay has to be generated in advance by statistical delay prediction models.

2 Required Work

In this study we evaluate the benefit of refined primary delay prediction models with increasing information content in robust crew and aircraft scheduling. It is based on

*lucian.ionescu@fu-berlin.de. Information Systems Department, Freie Universität Berlin, Garystraße 21, 14195 Berlin, Germany.

†natalia.kliewer@fu-berlin.de. Information Systems Department, Freie Universität Berlin, Garystraße 21, 14195 Berlin, Germany.

a previous study, examining the potential of data-driven delay prediction model generation, see [4]. It turned out that for primary delay the signal-to-noise ratio is rather low. During the statistical analysis, the variance of delays around these time trends became apparent. One can assume that in general, primary delays are inherently hard to predict in the long-term for robust scheduling. In this context, one always has to take into account that delay recording underlies constraints that lead to underestimation, e.g. predictable delay may already be prevented by scheduling decisions of an airline. Additionally, delay can only be recorded if it results in delayed flight departure. Delays absorbed by buffer times are not recognizable.

The simplest delay prediction model may assume the same primary delay risk for all flights, i.e. the same distribution for primary delay is used, independently from spatio-temporal attributes of the particular flight. This initial model then can be further refined by considering different delay risk for attributes, e.g. for different daytimes and seasonal components (season, month, week of the year), but as well for local (departure and arrival airport) and network attributes (hub-to-spoke and spoke-to-hub connection). In addition, we also evaluate theoretical models generated without using delay data.

For the study, we use real-world flight schedules and delay data of a major European airline for the time span from March 2003 to February 2007. It consists of 2.5 million delay records for domestic and continental flights in a distinct hub-and-spoke network with two major hubs. Moreover, 24.45% of all flights are spoke-to-spoke connections. It might be reasonably assumed that more complex models lead to more realistic assumptions made on primary delay and following to more efficient robust scheduling. However, scheduling decisions may be already limited by the airline flight network as some connections between flights are already predetermined, e.g. due to the out-and-back principle in hub-and-spoke networks. Following, most scheduling decisions refer to the hub airports.

Therefore, delay prediction models with a different degree of complexity are used in an existing robust scheduling framework. For a set of given flight schedules, crew and aircraft schedules are generated and then compared regarding their planned cost and on-time performance. Note that resource schedules can only be compared if they are generated for the same underlying flight schedule. In order to evaluate the degree of freedom for resource scheduling, we analogously analyze the similarity of the crew and aircraft schedules generated for the same flight schedule with different underlying prediction models. Finally, we discuss in how far an increased delay prediction accuracy leads to a better trade-off between robustness and cost-efficiency in robust resource scheduling.

Acknowledgements

This research was supported by a grant from the German Research Foundation (DFG, Grant No. KL2152/3-1).

References

[1] J.W. Yen, J.R. Birge (2006). *A stochastic programming approach to the airline crew scheduling problem.* Transportation Science, 40(1), 3-14.

[2] B. Tam (2011). *Optimisation approaches for robust airline crew scheduling*. PhD Thesis, School of Engineering, University of Auckland.

[3] V. Dück, L. Ionescu, N. Kliewer, and L. Suhl (2012). *Increasing stability of crew and aircraft schedules*. Transportation Research Part C: Emerging Technologies 20(1), 47-61.

[4] L. Ionescu, C. Gwiggner, N. Kliewer (2015). *Data Analysis of Delays in Airline Resource Networks*. To appear in: Business and Information Systems Engineering.

Valid inequalities for a time-indexed formulation

Lotte Berghman (Speaker) * Frits C.R. Spieksma †

1 Introduction

Consider the following problem. We have a single machine, n jobs (the jobset J), and a discrete timespan (the set of periods T). Each job $j \in J$ has a known processing time that depends on the period $t \in T$ at which job j is started: $p_{j,t}$; there is also a known cost $c_{j,t}$ associated to starting job j at period t. Notice that the period $|T|$ refers to the latest possible period that any job $j \in J$ can start. We assume all data to be integral, and all processing times to be positive. The problem is to schedule all jobs, nonpreemptively, such that the machine needs to work on at most one job in a period, while minimizing total costs.

Here is a formulation, involving binary variables $x_{j,t}$ indicating whether job $j \in J$ starts in period $t \in T$. We use, in this formulation, a set of periods called $A_{j,t}$, which is defined for each $j \in J, t \in T$, as follows:

$$A_{j,t} \equiv \{s \leq t : \ s + p_{j,s} - 1 \geq t\}.$$

Thus, $A_{j,t}$ represents, for a given job j and a given period t, the set of periods s such that if job j starts at period s, job j is still being processed at period t; notice that this set of periods need not be consecutive.

$$\min \quad \sum_{j \in J} \sum_{t \in T} c_{j,t} x_{j,t} \tag{1}$$

$$\text{such that} \quad \sum_{t \in T} x_{j,t} = 1 \qquad \forall j \in J, \tag{2}$$

$$\sum_{j \in J} \sum_{s \in A_{j,t}} x_{j,s} \leq 1 \quad \forall t \in T, \tag{3}$$

$$x_{j,t} \in \{0,1\} \qquad \forall j \in J, \forall t \in T. \tag{4}$$

This formulation is called a time-indexed formulation. One can verify easily that the above formulation is a correct one. We use P to denote the convex hull of feasible solutions to (2)-(4).

*l.berghman@tbs-education.fr. Université de Toulouse - Toulouse Business School, 20 BD Lascrosses BP 7010, 31068 Toulouse Cedex 7, France.

†frits.spieksma@kuleuven.be. Operations Research Group, KU Leuven, Naamsestraat 69, B-3000 Leuven, Belgium.

2 Motivation and Literature

There are two main reasons to investigate this formulation. First, model (1) - (4) is quite general. Depending on the choice of cost-coefficients $c_{j,t}$ many well-known scheduling problems arise. To give some examples, for minimizing the total completion time (the time at which the execution of a job ends) $c_{j,t} = t+p_{jt}$, for minimizing the total weighted completion time $c_{j,t} = w_j(t + p_{jt})$, for minimizing the total lateness (the difference between the completion time and the due date) $c_{j,t} = t + p_{jt} - d_j$, for minimizing the total tardiness (the violation of the due date) $c_{j,t} = \max\{0, t + p_{jt} - d_j\}$ and for minimizing the total flow time (the time elapsed from the release date of a job to its completion) $c_{j,t} = t + p_{jt} - r_j$. (Remark that release dates and deadline can be modeled easily using very large cost coefficients for the periods before the release date and after the deadline.)

Moreover, although stated as a single machine scheduling problem, in fact, by allowing time-dependent processing times $p_{j,t}$, scheduling problems with multiple machines are contained in this formulation.

Second, there are many cases sketched in literature where the processing time depends on the starting time. Apart from specific applications, the occurence of more general phenomena as learning effects, and deterioration effects have been well-documented in literature, and give rise to time-dependent processing times.

3 The Generalized Sousa-Wolsey inequalities

We can deduce the following set of valid inequalities. For each $i \in J$, for each $t_2 \in T\setminus\{1\}$, we define the set:

$$R_{i,t_2} = \{t : \ t \leq t_2 - 1 \text{ and } \exists j \in J \setminus \{i\} \text{ such that } t + p_{j,t} - 1 \geq t_2\}.$$

Thus, R_{i,t_2} represents the set of periods $t < t_2$ such that there exists a job $j \neq i$ that, when started at period t, is active at period t_2. Now we introduce, for each $i \in J$, $t_2 \in T \setminus \{1\}$, and for each $t_1 \in R_{i,t_2}$, the following inequality:

$$\sum_{j\in J\setminus\{i\}} \sum_{s\in A_{j,t_2} s\leq t_1} x_{j,s} + \sum_{s\in A_{i,t_1}} x_{i,s} + \sum_{s=t_1+1}^{t_2} x_{i,s} \leq 1. \tag{5}$$

We observe that, in case $p_{j,t} = p_j$ for all j, t, these inequalities are precisely those introduced by Sousa and Wolsey [3]. Therefore, we refer to these inequalities as the Generalized Sousa-Wolsey (or GSW) inequalities.

Theorem 1 *The GSW inequalities (5) are valid inequalities for P.*

We will now investigate the polyhedral structure of P. These results can be seen as generalizations of results given in Sousa and Wolsey [3], Crama and Spieksma [1], and Van Den Akker et al. [4]. We use techniques similar to the ones used in these papers to derive our results; we point out, however, that not all properties valid for the problem with $p_{j,t} = p_j$ remain valid for our more general case of time-dependent processing times. We refer to Nemhauser and Wolsey [2] for a thorough introduction into polyhedral theory.

Theorem 2 *The dimension of the polytope P, $dim(P) = n(|T| - 1)$.*

For the GSW inequalities, we need a definition to be able to identify those GSW inequalities that define facets of P:

$$R^{=}_{i,t_2} = \{t : \ t \leq t_2 - 1 \text{ and } \exists j \in J \setminus \{i\} \text{ such that } t + p_{j,t} - 1 = t_2\}.$$

Theorem 3 *For each $i \in J$, $t_2 \in T \setminus \{1\}$, $t_1 \in R^{=}_{i,t_2}$: a GSW inequality (5) defines a facet of P.*

A natural question to consider is whether other families of facet-defining inequalities with coefficients in $\{0, 1\}$, and with right-hand side 1 exist. Let us call an inequality where the coefficient of each variable is in $\{0, 1\}$ a *set-packing* inequality. In fact, it turns out that no other facet-defining set-packing inequalities exist for P.

Theorem 4 *All facet-defining inequalities of P that are of the set-packing type and have right-hand side 1, are constraints (3) and the GSW-inequalities.*

References

[1] Y. Crama and F.C.R. Spieksma (1996). Scheduling jobs of equal length: complexity, facets and computational results. *Mathematical Programming 54*, 353–367.

[2] G. Nemhauser and L.A. Wolsey (1988). Integer and Combinatorial Optimization. *Wiley*, New York.

[3] J.P. Sousa and L.A. Wolsey (1992). A time-indexed formulation of non-preemptive single-machine scheduling problems. *Mathematical Programming 72*, 207–227.

[4] J.M. Van Den Akker, C.A.J. Hurkens, and M.W.P. Savelsbergh (1992). Time-Indexed Formulations for Machine Scheduling Problems: Column Generation. *INFORMS Journal on Computing 12*, 111–124.

Locks and emissions

Dirk Briskorn (Speaker) [*] Frits C.R. Spieksma [†] Ward Passchyn [‡]

1 Problem descriptions

On many inland waterways, locks are required to ensure a suitable water level for navigation. Typically, and notably when the waterway traffic density is high, locks act as bottlenecks, introducing waiting time for ships that pass through these canals and waterways. We consider here locks arranged in a sequence along a canal. Scheduling of ships in waterways has attracted quite some attention in the scientific community recently, see e. g. Hermans (2014), Lübbecke et al. (2014), and Verstichel (2013).

In our problem settings, ships arrive at one end of the waterway at a predetermined point of time and travel to the opposite end of the waterway. Each ship must pass all locks in the corresponding order. Each lock acts as a single server which handles both the upstream and the downstream traffic. Lock operations must alternate between upwards (downstream to upstream) and downwards (upstream to downstream) movements. The lock operations take a positive amount of time.

We consider two different problem settings:

- We consider a single lock and ships moving with constant speed. The question is when to start a lock operation such that total waiting time of ships is minimized. We refer to this problem as MinWait in the following.
- We consider multiple locks and ships with variable speeds, speed-dependent emissions, and deadlines for reaching the opposite end of the waterway. Here, the question is when to start lock operations and how to set ships' speeds in order to minimize total emission without any ship missing its deadline.

2 Results

2.1 Minimization of Total Waiting Time

The main result regarding MinWait is given by the following theorem, see Coene et al. (2013).

[*] briskorn@uni-wuppertal.de. Schumpeter School of Business and Economics, Department of Production and Logistics, University of Wuppertal, Rainer-Gruenter-Str. 21, D-42119 Wuppertal, Germany.

[†] frits.spieksma@kuleuven.be. Research group Operations Research and Business Statistics (ORSTAT), KU Leuven

[‡] ward.passchyn@kuleuven.be. Research group Operations Research and Business Statistics (ORSTAT), KU Leuven

Theorem 1 *MinWait can be solved in $O(n^2)$ if the lock's chamber is uncapacitated with n being the number of ships.*

First, we reduce MinWait to the shortest path problem on an acyclic graph. The basic idea is to consider subsequences of lock operations without any idela time in between and to represent a solution by a set of such subsequences. It is not hard to see that such a subsequence starts only at the time and the position of a ship arrival. This gives us a polynomial number of subsequences to be represented. We obtain a graph with $O(n^2)$ nodes and $O(n^3)$ arcs. Finally, we show that we can handle a subset of more efficiently than with straight-forward dynamic programming. This yields our result.

Building on this result we can show that several extensions of MinWait can be solved in polynomial time. These extensions cover (alternatively)

- arbitrary regular objective functions,
- non-uniform lockage times,
- capacitated chamber (no of ships),
- capacitated (size of ships, non-overtaking), and
- bounded number of movements

However, when the chamber is capacitated, ships have individual sizes, and we allow overtaking MinWait becomes NP-hard. Also, if ship-dependent handling times when entering the chamber are considered the resulting problem version is NP-hard. Both results can be proven by reduction from 3-PARTITION which is known to be strongly NP-complete, see Garey and Johnson (1979).

2.2 Minimization of Total Emission

Our main results are two MIP models, see Passchyn et al. (2015). The first one is based on a discretization of the planning horizon and constitutes a time-indexed formulation. The number of binary variables employed depends on the length of planning horizon and the number of periods resulting from the discretization. The second model considers a continuous planning horizon and employs a number of binary variables which is independent from the length of the planning horizon.

Our computational study shows that both models have advantages as well as disadvantages with regards to computation time. Naturally, the run time of the first model heavily depends on the size (and, consequently, the number) of time slots. While it performs better for relatively large (and, therefore, few) time slots, it is outperformed by the second one for relatively small (and, therefore, many) time slots.

References

Coene, S., Passchyn, W., Spieksma, F., Vanden Berghe, G., Briskorn, D., and Hurink, J. (2015). The lockmaster's problem. *under review.*

Passchyn, W., Briskorn, D., and Spieksma, F. (2015). Mathematical Programming Models for Lock Scheduling with an Emission Objective. *under review.*

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco.

Hermans, J. (2014). Optimization of inland shipping - a polynomial time algorithm for the single ship single lock optimization problem. *Journal of Scheduling*, 17:305–319.

Lübbecke, E., Lübbecke, M. E., and Möhring, R. H. (2014). Ship traffic optimization for the kiel canal. Technical report, TU Berlin, Inst. Math. and RWTH Aachen, Operations Research.

Verstichel, J. (2013). *The Lock Scheduling Problem.* PhD thesis, KU Leuven.

Scheduling Messages to Detect Patterns Continuously on a Grid Sensor Network

Bala Kalyanasundaram (Speaker) * Mahe Velauthapillai †

1 Introduction

A wireless sensor network (WSN) is a network of devices called sensor nodes that communicate wirelessly. This network is used in many areas including environmet monitoring, traffic management, wild life monitoring [1, 2, 7]. Depending on the application, a WSN can consist of a few nodes to millions of nodes. A sensor node may have one or more sensor modules for measuring some information, for example intensity of light, temperature, humidity, pressure, sound, etc. Additionally, each sensor node includes components to handle storage, processing, communication, and power. When activated by a broadcast message from an external source, the network monitors the environment continuously to detect and/or react to certain predefined events or patterns.

The problem we consider in this paper is not restricted to wireless communication between sensors. For instance, CMOS technology for active pixel image sensors (see [8]) are chosen over charge-coupled devices (i.e., CCD's) due to the possibility of on-chip signal processing for rapid image analysis in applications such as particle tracking in Physics (see [5]). In this case, the sensor network is a simple grid network with millions of image pixels. Inability to offload images fast enough to process them on a faster and higher-end machines make CMOS technology with active pattern detection more suitable for such applications. A pattern in this context is an image specified as a $(2k+1)$ by $(2k+1)$ (pixel) matrix. For instance, it could be an image of a person. The system could continually monitor and alert when the image of the person appears.

At each time step, a sensor detects information at its location and exchanges messages with its immediate neighbors so that they can collectively detect when and where pattern/event appears. If a $2k+1$ by $2k+1$ pattern appears at time t then the sensor at the center of the pattern must detect it no later than time $t+O(k)$. This is achieved by devising a protocol to exchange messages between neighboring nodes. The detection process is not a one-time but ongoing process. Every occurrence of the pattern (time- and space-wise) must be detected by its center node. When a pattern size is big enough to cover the entire network, we say that the pattern is a global pattern. This paper introduces continuous communication complexity for a grid and analyzes the communication complexity of the detection process. Upper bounds (i.e., protocols) are also presented for

*kalyan@cs.georgetown.edu. Department of Computer Science, Georgetown University, Washington DC, 20057, USA.

†mahe@cs.georgetown.edu. Department of Computer Science, Georgetown University, Washington DC, 20057, USA.

patterns such as monochromatic, parity, majority, counting and any arbitrary singleton pattern. Some bounds are shown to be tight.

Given a $(2k+1)$ by $(2k+1)$ pattern, what information should each sensor exchange with its neighbor so that it can detect the event continuously? How does it scale when the size of the pattern increases? What happens when the pattern becomes global where every node co-operatively tries to detect one global pattern? What if sensors are uniform and employ same protocol? What if we allow protocols to differ in some uniform-way.

We define communication complexity of a protocol to be the maximum number of bits sent by a node to its neighbor at any time. The communication complexity of a pattern is the communication complexity of the best protocol. This definition is intuitive and captures the sampling rate of the detection process.

This line of work has some resemblance to the volume of work done in systolic computations. Many different parallel algorithms and architectures were considered by many researchers. We refer readers to books by Leighton [6] and Reif [9]. Despite some similarities, there are some major differences that make our results unique. Continuously changing input and simultaneous tracking of patterns on each location of the network make our problem different. The main objective function of our model is to minimize communication while dealing with rapidly changing input. Tracking each occurrence of the pattern and changing input poses some challenges akin to the direct sum conjecture in two-party communication model of Yao [10].

It is also important to recognize the volume of work done on data aggregation in sensor network and systolic computations [3, 4]. Typically, the entire network co-operates to calculate data aggregation such as sum, average and other possible statistical measures on detected sensor values. Gao et. al [4] have considered aggregation of data from a small subset of nodes instead of an entire network. This paper can be viewed as a continuous aggregation/detection of multiple subsets of data all over the network. We will show that global data aggregation/detection can be exponentially easier than the problem of local data aggregation/detection. There is a legitimate and simple reason for such a seemingly non-intuitive gap.

For simplicity, we assume that each sensor node detects a bit at each time step. A pattern of size k is a set of $2k+1$ by $2k+1$ boolean matrices. We say that a pattern is singleton if the size of the set is 1. We say that a protocol is *uniform* if every node employs the same protocol.

Theorem 1 *The communication complexity of an arbitrary singleton pattern of size k on a grid is $O(\log k)$. On the other hand, the communication complexity of any irreducible pattern of size k is $\Omega(\log k)$ for a straight-line.*

Theorem 2 *There is a uniform protocol to detect monochromatic pattern (all 0) on a grid with communication complexity $O(\log k)$. No asymptotically better uniform protocol exists for a grid.*

Theorem 3 *The communication complexity of parity pattern of size k is $\Theta(k)$.*

Recall that pattern P of size k is a set of $2k+1$ by $2k+1$ boolean matrix and the size of the set P is between 1 and $2^{(2k+1)(2k+1)}$.

Theorem 4 *Consider a pattern of size k where P is the corresponding set. The communication complexity of the pattern is $O(\log(max(k, |P|))$.*

Observe that if $|P|$ is polynomial in k, then the communication complexity is $O(\log k)$.

Open Problem 5 *Is there pattern of size k with communication complexity $\omega(k)$?*

These ideas can be extend to other networks such as tree, bounded degree graph or any arbitrary graph. For a bounded degree tree, detecting whether there is a "1" within a distance k from any node is $O(\log k)$. However the protocol is non-uniform.

2 Acknowledgement

The authors would like to acknowledge the support of Craves Family Professorship and McBride Professorship.

References

[1] Daniel J. Abadi, Samuel Madden, and Wolfgang Lindner. Reed: Robust, efficient filtering and event detection in sensor networks. In *IN VLDB*, pages 769–780, 2005.

[2] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. In *IEEE Personal Communication Magazine*, pages 10–15, 2000.

[3] Sorabh Gandhi, John Hershberger, and Subhash Suri. Approximate isocontours and spatial summaries for sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 400–409, New York, NY, USA, 2007. ACM.

[4] Jie Gao, Leonidas Guibas, Nikola Milosavljevic, and John Hershberger. Sparse data aggregation in sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 430–439, New York, NY, USA, 2007. ACM.

[5] G. Gaycken, A. Besson, A. Gay, Y. Gornushkin, D. Grandjean, F. Guilloux, S. Heini, A. Himmi, Ch. Hu, K. Jaaskelainen, M. Pellicioli, I. Valin, M. Winter, G. Claus, C. Colledani, G. Deptuch, W. Dulinski, and M. Szelezniak. Monolithic active pixel sensors for fast and high resolution vertex detectors. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 560(1):44 – 48, 2006. Proceedings of the 13th International Workshop on Vertex Detectors - VERTEX 2004.

[6] F. Thomson Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[7] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, and David Culler. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97, 2002.

[8] Sunetra K. Mendis, Sabrina E. Kemeny, Russell C. Gee, Bedabrata Pain, Craig O. Staller, Quiesup Kim, Eric R. Fossum, and Senior Member. Cmos active pixel image sensors for highly integrated imaging systems. *IEEE Journal of Solid-State Circuits*, 32:187–197, 1997.

[9] John H. Reif. *Synthesis of Parallel Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[10] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *STOC*, pages 209–213, 1979.

Some positive news on the proportionate open shop problem

Sergey Sevastyanov (Speaker) *

1 Introduction

We consider the m-machine *open shop* problem with the minimum makespan objective, shortly denoted as $Om||C_{\max}$. In this problem we are given a set of jobs $\{J_1, \ldots, J_n\}$ that should be processed on a given set of machines $\{M_1, \ldots, M_m\}$. Each job consists of m parts, called *operations*. Different operations of a job should be processed on different machines without overlapping in time, and the processing of each operation requires a given portion of time. The processing of different operations on the same machine should not overlap in time, neither. Among the set of feasible schedules (which meat the above requirements), we wish to find a schedule that minimizes the maximum job completion time (denoted as $C_{\max}$).

This classical scheduling problem was posed by Gonzalez and Sahni in 1976 [1], where they proved that already the three-machine problem $\langle O3 \,||\, C_{\max}\rangle$ is ordinary NP-hard. The question on its strong NP-hardness remains open till today, for nearly forty years.

We investigate a special case of the open shop problem, which is known as a "proportionate open shop problem" and is denoted by $\langle Om \,|\, prpt \,|\, C_{\max}\rangle$. In this problem, each job J_j has m operations of the same length p_j that should be performed on different machines (thus, the operation processing times are only job-dependent). This is also a difficult problem, since its three-machine case $\langle O3 \,|\, prpt \,|\, C_{\max}\rangle$ still remains ordinary NP-hard. We will not concern here the topic of numerous practical applications of this problem, — it can be found in scheduling literature. So, we will only concern some theoretical issues of this problem.

First we note that, despite the existence of a quite efficient PTAS for the general open shop problem [7], such traditional research directions as construction of efficient heuristics and searching for efficiently solvable sub-cases still remain urgent for the proportionate open shop problem, which can be confirmed by recent publications on this subject (see e.g. [3] and [4]). In our paper we present a few results of both types. This, in turn, will enable us to compute the exact values of the *power of preemption* in the proportionate open shop problem for some fixed values of the number of machines.

We remind that the "power of preemption" for a given instance I (denoted as $PoP(I)$) is defined as a ratio of two optima: the one for the problem version without preemption to the one for the version with preemption allowed.

*seva@math.nsc.ru. Sobolev Institute of mathematics, Novosibirsk State University, Acad. Koptyug avenue, 4, Novosibirsk, Russia. Supported by the Russian Foundation for Humanities (grant No. 13-22-10002)

2 Results

1) **Polynomially solvable cases** (based on sufficient conditions imposed on input data). In each (known to us) case of polynomial solvability of the open shop problem for an infinite class of instances $\mathcal{I}'$ this event appears to be synchronized with the event that the optimum $OPT(I)$ of any instance $I \in \mathcal{I}'$ coincides with the standard lower bound $LB(I)$ defined as the maximum of two amounts: the maximum job length $JL(I)$ and the maximum machine load $ML(I)$. As was convincingly demonstrated in [2], the property $OPT(I) = LB(I)$ is quite widespread in open shops. For that reason, any instance I with this property was called in [2] a *normal instance*. A class of instances $\mathcal{I}'$ with the property $OPT(I) = LB(I),\ \forall\ I \in \mathcal{I}'$, was called "normal", as well.

In our paper we found some new normal classes of instances for the proportionate open shop problem, which also appeared to be polynomially solvable.

(a) The first such class defined by the sufficient condition

$$ML(I) \geq 2(m-1)p_{\max} \tag{1}$$

is an immediate corollary from our old result from [5]. In [5] it was proved for the general open shop problem that the condition $ML(I) \geq mR + 2(m-1)p_{\max}$ is sufficient for a problem instance I to be normal, where R is the value of the radius of vector summation in the *compact vector summation problem* related to the $Om||C_{\max}$ problem. Since in the case of the proportionate open shop, all those vectors (defined in the corresponding compact vector summation problem) are zero-vectors, they can be summed within a ball of radius $R = 0$, and the desired condition (1) immediately follows. The length of the optimal schedule in this case coincides with $ML(I)$. We prove that condition (1) is tight, which means that for any $\varepsilon > 0$ there exists an "abnormal" instance I of the m-machine proportionate open shop problem with the relation $ML(I) = (2m - 2 - \varepsilon)p_{\max}$.

As for the running time is concerned, while the above mentioned solvable case of the $Om||C_{\max}$ problem requires $O(n^2m^2)$ time (mainly needed for finding the vector summation), in the special case of the proportionate open shop problem we can do with only $O(n)$ time.

(b) Another normal (and polynomially solvable) class is defined by the sufficient condition

$$ML(I) \leq (m-1)p_{\max}. \tag{2}$$

The optimal schedule of length $ML(I)$ can be found in $O(n + m\log m)$ time.

(c) Condition (2) can be slightly improved for small values of m. In particular, we can show that for $m = 3$ and $m = 4$ machines it can be replaced by more powerful conditions $ML(I) \leq 2.5p_{\max}$ and $ML(I) \leq 3.5p_{\max}$ respectively. Both conditions are tight, and the optimal schedules can be easily obtained in linear time.

2) Optima localization results and efficient heuristics.

In [6] it was proved that for any instance I of the $\langle O3\,||\,C_{\max}\rangle$ problem its optimum belongs to the interval $[LB(I), \frac{4}{3}LB(I)]$, and the bounds of the interval (for the general 3-machine open shop problem) are tight. As a byproduct of the proof of that theoretical result, a linear time heuristic H was designed which guaranteed finding a schedule $\sigma_H(I)$ with length in the same interval:

$$C_{\max}(\sigma_H(I)) \leq \frac{4}{3}LB(I).$$

This implied that the ratio performance guarantee of the heuristic was equal to $\frac{4}{3}$.

In our paper we tighten the above bounds with respect to that special case of the open shop problem we are interested in. Namely, it is proved that for any instance I of the $\langle O3 \,|\, prpt \,|\, C_{\max}\rangle$ problem its optimum belongs to the interval $[LB(I), \frac{10}{9}LB(I)]$, and the bounds of the interval are tight. And similarly, we design a simple linear-time heuristic able to find a schedule with the length in this interval, thus providing the ratio performance guarantee of the heuristic equal to $\frac{10}{9}$. This also improves on the result by Koulamas and Kyparisis [3] who suggested for the same problem a heuristic with a weaker performance guarantee (of $\frac{7}{6}$) and a longer running time (of $O(n \log n)$).

Similar results we obtained for further small values of m. It was shown that the maximum value of the ratio $OPT(I)/LB(I)$ is not greater than $\frac{9}{8}$ for $m = 4$, not greater than $\frac{17}{15}$ for $m = 5$, and not greater than $\frac{8}{7}$ for $m = 6$. In a similar way, we design linear-time heuristic for these special cases of the proportionate open shop problem with performance guarantees $\frac{9}{8}$, $\frac{17}{15}$, and $\frac{8}{7}$ respectively.

Besides that, we invented another simple heuristic H' (with running time $O(n + m \log m)$) which for any number of machines m and any m-machine instance I guarantees finding a schedule $\sigma_{H'}(I)$ with length

$$C_{\max}(\sigma_{H'}(I)) \leq \left(1 + \frac{1}{m}\right) LB(I).$$

Thus, the relative error of the heuristic is not greater than $1/m$ and tends to zero as m tends to infinity. Our result surpasses the characteristics of a recent result published by Naderi a.o. [4] who suggested a heuristic with the performance guarantee of $\left(2 - \frac{1}{m}\right)$, at a worse bound on running time.

3) The exact values of the Power of Preemption for the Proportionate Open Shop problem.

From the tight bounds on the optima (mentioned above) and from the fact that the optimum of any instance I of the **preemptive open shop** problem coincides with the lower bound $LB(I)$, we can derive the exact values of the power of preemption for small values of m. Namely, for $m = 3, 4, 5, 6, 7$ it is equal to $10/9, 9/8, 17/15, 8/7, 8/7$, respectively. At that, for any m it is not greater than $(m + 1)/m$. Thus, the maximum value of the power of preemption (over all values of m) is attained at $m = 6$ and $m = 7$ and is equal to $8/7$.

Список литературы

[1] Gonzalez T., Sahni S. (1976), Open shop scheduling to minimize finish time, *J. Assoc. Comput. Mach.*, 23(4). P. 665–679.

[2] A. Kononov, S. Sevastianov, and I. Tchernykh (1999), When difference in machine loads leads to efficient scheduling in open shops, *Annals of Operations Research*, 92. P. 211–239.

[3] Christos Koulamas. and George J. Kyparisis (2015), The Three-Machine Proportionate Open Shop and Mixed Shop Minimum Makespan Problems, *European Journal of Operational Research*, to appear.

[4] B. Naderi, M. Zandieh, and M. Yazdani (2014), Polynomial time approximation algorithms for proportionate open-shop scheduling, *International Transactions in Operational Research*, 21(6). P. 1031–1044.

[5] S.V. Sevast'janov (1995), Vector summation in Banach space and polynomial algorithms for flow shops and open shops, *Mathematics of Operations Research*, 20(1). P. 90–103.

[6] S.V. Sevastianov and I.D. Tchernykh (1998), Computer-Aided Way to Prove Theorems in Scheduling, in: Bilardi, a.o. (Eds.), Algorithms - ESA'98, 6th Annual European Symposium, Venice, Italy, August 1998. Proceedings, *Lecture Notes in Computer Science*, 1461, Springer-Verlag. P. 502–513. ISBN: 3-540-64848-8

[7] S.V. Sevastianov and G.J. Woeginger (2001), Linear time approximation scheme for the multiprocessor open shop problem, *Discrete Applide Mathematics*, 114. P. 273–288.

Linearization of directed acyclic graphs on a failure-prone processor*

Guillaume Aupy (Speaker) † Anne Benoit ‡ Henri Casanova §
Yves Robert ¶

1 Introduction

Motivated by the execution of tightly-coupled applications on large-scale platforms [2, 3], we study the scheduling of computational workflows on compute resources that experience exponentially distributed failures. A tightly-coupled application executed on p processor is such that if one processor fails, the whole application fails. Hence one can see the p processors each experiencing exponentially distributed failures with parameter λ_{ind} as a single "super"-processor experiencing exponentially distributed failures with parameter $\lambda = p\lambda_{\text{ind}}$. In order to cope with failures, we have the possibility to checkpoint (save the work done) at the end of the execution of the different tasks of the workflow. When a failure occurs, rollback and recovery is used to resume the execution from the last checkpointed state. The scheduling problem is to minimize the expected execution time by deciding in which order to execute the tasks in the workflow and whether to checkpoint or not checkpoint a task after it completes.

Platform We consider a single processor subject to failures exponentially distributed with mean time between failures (MTBF) $\mu = \frac{1}{\lambda}$. There are two storage locations associated to the platform: *memory* and *disks*. When a failure occurs, all that was written on memory is lost. Then the processor experiences a *downtime* of D units of time before it can be used again.

Applications We want to execute an application that is structured as a DAG $\mathcal{G} = (V, E)$ where V is a set of vertices and E a set of edges. Each vertex represents a task, and the edges represent the data dependencies between the tasks. In particular, for all tasks $T \in V$, a failure-free execution of task T takes w units of time (the task's computational *weight*). This execution produces an output, that is stored on memory. This output that can be checkpointed (written to disks) in c units of time. Finally, this checkpoint can be recovered (read from disks) from a failure in r units of time.

*The extended version of this abstract can be found under the name "Scheduling computational workflows on failure-prone platforms" by the same authors

†guillaume.aupy@ens-lyon.fr. LIP, ENS Lyon, France
‡anne.benoit@ens-lyon.fr. LIP, ENS Lyon, France
§henric@hawaii.edu. University of Hawai'i, USA
¶yves.robert@inria.fr. LIP, ENS Lyon, France & University of Knoxville, USA

An execution of T requires that the input data to T be available in memory. In particular, if a failure happens during the execution of T, then T must be re-executed. In order to do so, for each reverse path in the DAG from T back to an entry task, one must find the most recently executed checkpointed task. One must then recover from that checkpoint, and re-execute all the tasks that were executed after that checkpointed task, i.e., all tasks whose output was lost and that are ancestors of T along the reverse path. It may be that on such a path from T to an entry task, no checkpointed task is found, in which case one must begin by re-executing the entry task.

Optimization problem We define a schedule as a linearization of the DAG in which, for each task, it is specified whether the task's output should be checkpointed. The objective is to find the schedule that has the minimum expected makespan ($\mathbb{E}(C_{\max})$). We call this problem: DAG-ChkptSched.

2 Theoretical results

The full proofs for this section can be found in the companion research report [1].

The main result is the following:

Theorem 1. *Given a DAG, and a schedule for this DAG, it is possible to compute the expected execution time in polynomial time (i.e.,* DAG-ChkptSched *is in NP).*

This theorem, as uninformative as it seems, is quite tricky to prove. Consider a DAG and a schedule for this DAG. For simplicity we renumber the tasks so that task T_i is the i^{th} task executed in the linearization of the graph. The idea is to let X_i be the random variable that corresponds to the execution time between the end of the first successful execution of task T_{i-1} and the end of the first successful execution of task T_i. The expected execution time of the DAG is $\mathbb{E}[\sum_{i=1}^{n} X_i]$. We then want to compute for all i $\mathbb{E}[X_i]$ (using the linearization of the expectation). In order to do so, we partition the set of events depending when the last fault before the first execution of T_i occured. This decomposition eventually allows us to compute the expected makespan of the schedule.

We then show the intractability of DAG-ChkptSched, as well as some polynomial algorithms for specific graphs.

Fork DAGs are simple DAGs made of a source task with independent children. Formally $G = (V, E)$ where $V = \{T_0, T_1, \ldots, T_n\}$ and $E = \{(T_0, T_i), 1 \leq i \leq n\}$. Join DAGs are fork DAGs where edges are reversed, hence they are made of independent source tasks and a common sink task.

Theorem 2. DAG-ChkptSched *(i) can be solved in linear time for fork DAGs (ii) is NP-complete for join DAGs.*

The proof for fork DAGs is simple, because we can show that the ordering of the children does not matter. The only decision to make is whether to checkpoint the source task. One can compute the expected makespan of both cases and simply pick the case that achieves the lowest expected makespan.

The proof for join DAGs is much more elaborate and involves technical derivations. To prove the result for join DAGs, we start by denoting by I_{Ckpt}, resp. I_{NCkpt}, the subset

of $\{T_1, \ldots, T_n\}$ composed of the tasks that are checkpointed, resp. not checkpointed. We then prove that in the optimal solution, the tasks from I_{CKPT} should be executed before the tasks from I_{NCKPT}. Furthermore, while the execution order of the tasks from I_{NCKPT} does not matter the tasks from I_{CKPT} must be executed in a specific order (namely in non-increasing values of $g(i)$, where $g(i) = e^{-\lambda(w_i+c_i+r_i)} + e^{-\lambda r_i} - e^{-\lambda(w_i+c_i)}$). Given the two sets $(I_{\text{CKPT}}, I_{\text{NCKPT}})$, we can construct the optimal solution in polynomial time, hence the problem belongs to NP. Finally, we reduce the problem of finding the sets $(I_{\text{CKPT}}, I_{\text{NCKPT}})$ for a given upper bound on the expected makespan to finding a solution to the SUBSET-SUM problem [4].

While this problem is hard in general for join DAGs, there are instances in which it can be solved in polynomial time.

Proposition 3. DAG-ChkptSched *for a join DAG where $c_i = c$ and $r_i = r$ for all i can be solved in quadratic time.*

To prove this result, we simply need to show that in the optimal solution, if a task is checkpointed, then all tasks of weight greater than that last task are also checkpointed. Then the order of the tasks in I_{CKPT} is still given by function g. Finally, we compute the optimal solution amongst the solutions that checkpoint the i largest tasks for $i = 0$ to n.

The main open problem remaining is `what is the complexity of DAG-ChkptSched when` $c_i = c$ `and` $r_i = r$ `for all` i`?` Other problem include proving approximation results for special schemes. In particular, experimental evaluation show that Depth First Search schemes for linearization coupled with checkpointing largest tasks perform very well.

References

[1] Guillaume Aupy, Anne Benoit, Henri Casanova, and Yves Robert. Scheduling computational workflow on failure-prone platforms. Research report RR-8609, INRIA, Oct. 2014. Available at `http://graal.ens-lyon.fr/~abenoit`.

[2] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science (WORKS 2008)*, pages 1–10. IEEE, 2008.

[3] Jack Dongarra et al. The International Exascale Software Project. *Int. J. High Performance Computing App.*, 23(4):309–322, 2009.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

The optimal absolute ratio for online bin packing

János Balogh* József Békési† György Dósa† Jiří Sgall‡
Rob van Stee§

1 Introduction

In the online bin packing problem, a sequence of *items* with sizes in the interval $(0, 1]$ arrive one by one and need to be packed into *bins*, so that each bin contains items of total size at most 1. Each item must be irrevocably assigned to a bin before the next item becomes available. The algorithm has no knowledge about future items. There is an unlimited supply of bins available, and the goal is to minimize the total number of used bins (bins that receive at least one item).

Bin packing is a classical and well-studied problem in combinatorial optimization. Extensive research has gone into developing approximation algorithms for this problem. The focus of the research into approximation algorithms is on the question of how much performance degrades if an algorithm is constrained to work in polynomial time. In practical packing problems, however, it happens frequently that the input is not known completely before the algorithm starts working. It is therefore very natural to consider the *online* version of this problem. In online problems, we ask how much performance degrades as a result of not knowing the future. In general, there is no restriction on the amount of computation time used by an online algorithm. However, most online algorithms, including all the ones we consider in this paper, are very efficient.

For an input L, let $ALG(L)$ be the number of bins used by algorithm ALG to pack this input. Let $\textsc{opt}(L)$ denote the number of bins in an optimal

*Department of Applied Informatics, Gyula Juhász Faculty of Education, University of Szeged, H-6701 Szeged, POB 396, Hungary. {balogh,bekesi}@jgypk.u-szeged.hu

†Department of Mathematics, University of Pannonia, H-8200 Veszprém, Hungary. dosagy@almos.vein.hu

‡Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Praha, Czech Republic. sgall@ iuuk.mff.cuni.cz

§Department of Computer Science, University of Leicester, Leicester, UK. rob.vanstee@leicester.ac.uk

solution. If we want to have a performance guarantee relative to the optimal solution for every possible input, we need to consider the absolute competitive ratio, which is defined as $R_{ABS}(A) := \sup_L \left\{ \frac{A(L)}{\mathrm{OPT}(L)} \right\}$. Before our work, 1.7 was the best known absolute competitive ratio of any algorithm [1, 2]. There is a simple lower bound, consisting of only 18 items, which shows that no algorithm can be better than 5/3-competitive.

2 Algorithm and analysis

The main idea of our algorithm, which we call Five-Thirds (FT), is to use First Fit (FF) whenever possible, but try to avoid long sequences of 2-bins which occur in the lower bound construction for FF. It can be shown easily that 3^+-bins are generally fuller than 2-bins, and this compensates for 1-bins that are only half full at the end. However, 2-bins are problematic, since they may be only about 2/3 full on average. Therefore, whenever FT is about to put an item into any bin that has one item so far, it will from time to time put such an item into a new, empty bin instead, creating a *special bin* which is *specifically reserved for a large item*; no other item will be packed into it.

Definition 1 *A regular* 2^+*-bin is called* good *if it has level at least 5/6, it contains a large item, or the total size of its first two items is at least 3/4. A regular* 2*-bin is called* critical *if it is not a good bin. A regular* 2+*-bin is called* interesting *if it does not contain a large item.*

Whenever FT creates a special bin, it will immediately be *matched* to an existing critical bin, which is not yet matched to any special bin. Let s denote the number of special bins. Algorithm FT is shown in Figure 1.

The analysis of FT splits into three main cases. A major difficulty in all cases is that there may exist a single non-special 1-bin that has an item of size less than 1/2 (e.g., if this item arrives near the end of the input and is followed only by large items that do not fit with it). This complicates both the size-based and the weight-based analysis methods that we describe below.

If no special bin is ever created, FT behaves as FF throughout, and (due to our conditions for creating special bins), FF is 5/3-competitive in this situation. If there exists a special bin that does not contain a large item at the end of the input, it means that all large items in 1-bins are relatively large, since FT always puts large items in existing special bins and special items in existing 1-bins with large items if they fit. For this case, we use a size-based analysis.

Finally, if all special bins have large items, it means that these bins are relatively full. For this case, we use a weight-based analysis. Each item is assigned a weight which is a measure for how much space this item needs in

For each item a of the input, do the following:

1. If a is large, pack a by FF into the collection of all bins (or into a new bin if it does not fit anywhere).

2. Otherwise, let B be the bin that FF would pack a into if the collection of existing bins were restricted to the regular bins. (B is possibly a new bin.) If after packing a into B, there are at most $4s+1$ *interesting* bins, or B is not critical, or no unmatched critical bin exists, pack the item into B.

3. Otherwise, if before packing a there exists a regular 1-bin with a large item where a fits, pack a into the first such bin B'; declare B' to be a special bin where a is the special item. Match the special bin to the last unmatched critical bin.

4. Otherwise pack a into a new bin B''. Let b be the only item previously packed in B. If $a \leq b$, let B'' be a special bin while B remains regular. Else, change B to a special bin and let B'' be regular. The single item packed into the special bin is a special item. Match the special bin to the last unmatched critical bin.

Figure 1: Algorithm FT

any packing. The idea is that each optimal bin has total weight packed into it of at most 5/3, whereas FT packs an average weight of at least 1 per bin, immediately implying the desired result.

References

[1] G. Dósa and J. Sgall, *First Fit bin packing: A tight analysis*, in Proceedings of the 30th Symposium on the Theoretical Aspects of Computer Science (STACS 2013), Kiel, Germany, pp. 538–549, 2013.

[2] G. Dósa and J. Sgall, *Optimal analysis of Best Fit bin packing*, in Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014), to appear.

[3] M. R. Garey, R. L. Graham, and J. D. Ullman, *Worst-case analysis of memory allocation algorithms*, in Proceedings of the 4th Symposium on the Theory of Computing (STOC), ACM, pp. 143–150, 1973.

Lagrangian Duality based Algorithms in Online Scheduling

Nguyen Kim Thang *

In the paper [9], we consider Lagrangian duality based approaches to design and analyze algorithms for online energy-efficient scheduling. First, we present a primal-dual framework. Our approach makes use of the Lagrangian weak duality and convexity to derive dual programs for problems which could be formulated as convex assignment problems. The duals have intuitive structures as the ones in linear programming. The constraints of the duals explicitly indicate the online decisions and naturally lead to competitive algorithms. Second, we use a dual-fitting approach, which also based on the weak duality, to study problems which are unlikely to admit convex relaxations. Through the analysis, we show an interesting feature in which primal-dual gives idea for designing algorithms while the analysis is done by dual-fitting.

We illustrate the advantages and the flexibility of the approaches through problems in different setting: from single machine to unrelated machine environments, from typical competitive analysis to the one with resource augmentation, from convex relaxations to non-convex relaxations.

Applications of the primal-dual approach. In the setting, there are a set of unrelated machines. Each job j is released at time r_j, has deadline d_j, a value a_j and a processing volume p_{ij} if job j is executed in machine i. Jobs could be executed preemptively but migration is not allowed, i.e., no job could be executed in more than one machine. At any time t, the scheduler has to choose an assignment of jobs to machines and the speed of each machine in order to process such jobs. The energy cost of machine i is $\int_0^\infty P(s_i(t))dt$ where P is a given convex energy power and $s_i(t)$ is the speed of machine i at time t. Typically, $P(z) = z^\alpha$ for some constant $\alpha \geq 1$. In the setting, we look for competitive and energy-efficient algorithms. The following objectives are natural ones representing the tradeoff between value and energy. The first objective is to minimize energy cost plus the *lost value* — which is the total value of uncompleted jobs. The second objective is to maximize the total value of completed jobs minus the energy cost.

1. For the objective of minimizing energy plus the lost value we derive a primal-dual algorithm for the single machine setting. The competitive ratio is characterized by a system of differential equations. For a specific case where $P(z) = z^\alpha$, the competitive ratio turns out to be α^α (and recognize the result in [7]). With the primal-dual framework, the result is more general and the analysis is simpler.

2. For the objective of maximizing the total value of completed jobs minus the energy cost, it has been shown that without resource augmentation no algorithm has

*thang@ibisc.fr, IBISC, University Evry Val d'Essonne, France.

bounded competitive ratio even for a single machine [8]. Moreover, Pruhs and Stein [8] gave a $(1+\epsilon)$-speed and $O(1/\epsilon^3)$-competitive algorithm for a single machine. The analysis is done by a complex charging scheme. The authors raised interesting open direction to study the problem in more general context of unrelated machines.

We study the problem for unrelated machines in the resource augmentation model. We give a primal-dual algorithm which is $(1+\epsilon)$-speed and $1/\epsilon$-competitive for every $\epsilon \geq \epsilon(P) > 0$ where $\epsilon(P)$ depends on function P. For typical function $P(z) = z^\alpha$, $\epsilon(P) = 1 - \alpha^{-1/\alpha}$ which is closed to 0 for large α.

Application of the dual-fitting approach. We consider the general energy model: speed scaling with power down. There is a machine which can be set either in the sleep state or in the active state. Each transition of the machine from the sleep state to the active one has cost A, which represents the *wake-up* cost. In the sleep state, the energy consumption of the machine is 0. The machine, in its active state, can choose a speed $s(t)$ to execute jobs. The power energy consumption of the machine at time t in its active state is $P(s(t)) = s(t)^\alpha + g$ where $\alpha \geq 1$ and $g \geq 0$. Hence, the consumed energy (without wake-up cost) of the machine is $\int_0^\infty P(s(t))dt$ where the integral is taken during the machine's active periods. We decompose the latter into *dynamic energy* $\int_0^\infty s^\alpha(t)dt$ and *static energy* $\int_0^\infty g dt$ (where again the integrals are taken during active periods). Jobs arrive over time, a job j is released at time r_j, has weight w_j and requires p_j units of processing volume if it is processed on machine i. A job could be processed preemptively. At any time, the scheduler has to determine the state and the speed of every machine (it it is active) and also a policy to execute jobs. We consider two problems in the setting.

In the first problem, each job j has additionally a deadline d_j by which the job has to be completed. The objective is to minimize the total consumed energy.

In the second problem, jobs do not have deadline. Let C_j be the completion time of the job j. The *flow-time* of a job j is defined as $C_j - r_j$, which represented the waiting time of j on the server. The objective is to minimize the total weighted flow-time of all jobs plus the total energy.

As posed in [1], an important direction in energy-efficient scheduling is to design competitive algorithms for online problems in the general model of speed scaling with power down. Attempting efficient algorithms in the general energy model, one encounters the limits of current tools which have been successfully applied in previous energy models. That results in a few work on the model in contrast to the widely-studied models of speed scaling only or power down only. The potential function method, as mentioned earlier, yield little insight on the construction of new algorithms in this general setting. Besides, different proposed approaches based on the duality of mathematical programming [2, 5, 4] require that the problems admit linear of convex relaxations. However, it is unlikely to formulate problems in the general energy model as convex programs.

Our results in the general energy model are the following.

1. For the problem of minimizing the total consumed energy, we formulate a natural *non-convex* relaxation using the Dirac delta function. We prove that SOA [6] is indeed $\max\{4, \alpha^\alpha\}$-competitive by the dual-fitting technique. Although the improvement is slight, the analysis is *tight*[1] and it suggests that the duality-based

[1]The algorithm has competitive ratio exactly α^α even without wake-up cost [3].

approach is seemingly a right tool for online scheduling. Through the problem, we illustrate an interesting feature in the construction of algorithms for non-convex relaxations. The primal-dual framework gives ideas for the design of an algorithm while the analysis is done using dual-fitting technique.

2. For the problem of minimizing energy plus weighted flow-time, we derive a $O(\alpha/\log\alpha)$-competitive algorithm using the dual fitting framework; that matches the best known competitive ratio (up to a constant) for the same problem in the restricted speed scaling model (where the wake-up cost and the static energy cost are 0). Informally, the dual solutions are constructed as the combination of a solution for the convex part of the problem and a term that represents the lost due to the non-convex part. Building upon the salient ideas of the previous analysis, we manage to show the competitiveness of the algorithm.

References

[1] Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.

[2] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proc. 23rd ACM-SIAM Symposium on Discrete Algorithms*, pages 1228–1241, 2012.

[3] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.

[4] Nikhil R. Devanur and Zhiyi Huang. Primal dual gives almost optimal energy efficient online algorithms. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms*, 2014.

[5] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *Proc. 10th Workshop on Approximation and Online Algorithms*, pages 173–186, 2012.

[6] Xin Han, Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Deadline scheduling and power management for speed bounded processors. *Theor. Comput. Sci.*, 411(40-42):3587–3600, 2010.

[7] Peter Kling and Peter Pietrzyk. Profitable scheduling on multiple speed-scalable processors. In *Proc. 25th Symposium on Parallelism in Algorithms and Architectures*, 2013.

[8] Kirk Pruhs and Clifford Stein. How to schedule when you have to buy your energy. In *APPROX-RANDOM*, pages 352–365, 2010.

[9] Nguyen Kim Thang. Lagrangian duality based algorithms in online scheduling. *CoRR*, abs/1408.0965, 2014.

Primal-dual and dual-fitting analysis of online scheduling algorithms for generalized flow-time problems

Spyros Angelopoulos* Giorgio Lucarelli† Nguyen Kim Thang‡

1 Introduction

We consider online scheduling problems in which a set of jobs $\mathcal{J}$ arrive over time, and the jobs must be executed on a single processor. In particular, each job $j \in \mathcal{J}$ is released at time r_j and it is characterized by a *processing time* $p_j > 0$ and a *weight* $w_j > 0$, which become known after its release. The *density* of job j is $\delta_j = w_j/p_j$. Given a scheduling strategy, we denote by C_j the *completion time* of job j. The *flow time* of j is then defined as $F_j = C_j - r_j$. A natural optimization objective is to design schedules that minimize the *total weighted flow time*, i.e., $\sum_{j\in\mathcal{J}} w_j F_j$. We assume that preemptions are allowed.

Total weighted flow-time has been extensively studied. In the unweighted setting, it is well-known that the online algorithm Shortest Remaining Processing Time is optimal. In contrast, Bansal and Chan [2] showed that no algorithm is constant-competitive for minimizing total weighted flow-time on a single processor. This rather pessimistic lower bound motivated the study of the effect of *resource augmentation*, originally introduced by Kalyanasundaram and Pruhs [6]. Given some optimization objective (e.g. total flow time), an algorithm is said to be α-speed β-competitive if it is β-competitive with respect to an offline optimal scheduling algorithm of speed $\frac{1}{\alpha}$ (here $\alpha \leq 1$). In this context, Becchetti et al. [3] showed that the natural algorithm Highest-Density-First (HDF) is $(1+\epsilon)$-speed $\frac{1+\epsilon}{\epsilon}$-competitive for total weighted flow time.

Im et al. [5] introduced a generalization of the total weighted flow-time problem, in which jobs may incur non-linear contributions to the objective. More formally, they defined the *Generalized Flow-Time Problem* (GFP) in which the objective is to minimize $\sum_{j\in\mathcal{J}} w_j g(F_j)$, where $g : \mathbb{R}^+ \to \mathbb{R}^+$ is a given non-decreasing cost function with $g(0) = 0$. This extension captures many natural variants of flow-time with real-life applications; moreover, it is an appropriate formulation of the setting of optimizing simultaneously several objectives. Im et al. [5] showed that HDF is $(2+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm for general non decreasing functions g. On the negative side, they showed that no *oblivious* algorithm is $O(1)$-competitive with speed augmentation $2-\epsilon$, for any $\epsilon > 0$; the term oblivious refers to algorithms that do not know the function g. If g is a twice-differentiable, concave function, then there is an $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm, while for unit size jobs and general cost functions, FIFO is $(1+\epsilon)$-speed $\frac{4}{\epsilon^2}$-competitive [5]. Convex cost functions have been studied by Fox et al. [4].

*CNRS and LIP6, University Pierre and Marie Curie, Paris, France.
†LIG, University of Grenoble-Alpes, INP, France.
‡IBISC, University of Evry, Val d'Essonne, France.

Most of the above results rely to techniques based on amortized analysis, with or without an explicit potential function. More recently, techniques based on tools from linear programming have emerged for online scheduling problems. One of the main benefits in the application of primal-dual techniques is the fact that it offers intuition on both aspects of algorithm design and analysis. The objective of this work is to present a unified framework for a class of generalized flow-time problems that is based on primal-dual and dual-fitting techniques. More precisely, we first give a primal-dual analysis of HDF for the total weighted flow time problem which, albeit significantly more complicated than the known combinatorial one, yields insights about more complex problems. We then abstract the salient ideas of this proof into a framework which is applicable to more complex objectives and uses similar intuitive geometric interpretations of the primal/dual objectives. This framework allows us to either reprove in a simpler way known results or obtain improvements as well as new results. A full version of this abstract can be found in [1].

2 Results

A common approach in obtaining a competitive scheduling algorithm is by first deriving an algorithm for the *fractional* objective. Let $q_j(t)$ be the remaining processing time of job j at time t. The fractional remaining weight $w_j(t)$ of j at time t is $w_j \frac{q_j(t)}{p_j}$. The fractional objective of the GFP problem is $\sum_j w_j(t) g'(t - r_j)$. In [4] is proved that *if an algorithm is s-speed c-competitive for online fractional GFP, then there exists an* $(1+\epsilon)s$*-speed* $\frac{1+\epsilon}{\epsilon}c$*-competitive algorithm for the integral objective, for* $0 < \epsilon \leq 1$.

Let $x_j(t) \in [0, 1]$ be a variable that indicates the execution rate of $j \in \mathcal{J}$ at time t. The following is a valid linear-programming formulation for fractional GFP and its dual.

$$\min \sum_{j \in \mathcal{J}} \delta_j \int_{r_j}^{\infty} g(t - r_j) x_j(t) dt \qquad \text{(P)}$$

$$\int_{r_j}^{\infty} x_j(t) dt \geq p_j \quad \forall j \in \mathcal{J} \qquad (1)$$

$$\sum_{j \in \mathcal{J}} x_j(t) \leq 1 \quad \forall t \geq 0 \qquad (2)$$

$$x_j(t) \geq 0 \quad \forall j \in \mathcal{J}, t \geq 0$$

$$\max \sum_{j \in \mathcal{J}} \lambda_j p_j - \int_0^{\infty} \gamma(t) dt \qquad \text{(D)}$$

$$\lambda_j - \gamma(t) \leq \delta_j g(t - r_j) \qquad \forall j \in \mathcal{J}, t \geq r_j \qquad (3)$$

$$\lambda_j, \gamma(t) \geq 0 \qquad \forall j \in \mathcal{J}, \forall t \geq 0$$

The primal complementary slackness (CS) condition states that for a given job j and time t, if $x_j(t) > 0$, i.e., if the algorithm executes job j at time t, then it should be that $\gamma(t) = \lambda_j - \delta_j g(t - r_j)$. We would like then the dual variable $\gamma(t)$ to be such that we obtain some information about which job to schedule at time t. To this end, for any job $j \in \mathcal{J}$, we define the curve $\gamma_j(t) = \lambda_j - \delta_j g(t - r_j)$, with domain $[r_j, \infty)$, and slope $-\delta_j$. In order to ensure feasibility, our algorithm for every $t \geq 0$ choose $\gamma(t) = \max\{0, \max_{j \in \mathcal{J}: r_j \leq t}\{\gamma_j(t)\}\}$. We say that at time t the line γ_j is *dominant* if $\gamma_j(t) = \gamma(t)$. We can thus restate the primal CS condition as a *dominance* condition: if a job j is executed at time t, then γ_j must be dominant at t. Based on this, we abstract the essential properties in order to obtain optimal online algorithms for fractional objectives. We consider that the primal solution is generated by an online algorithm A. The crux is in maintaining dual variables λ_j, upon release of a new job z at time τ, such that the following properties are satisfied: ($\mathcal{P}1$) *Future dominance*: if the algorithm A executes job j at time $t \geq \tau$, then γ_j is dominant at t; ($\mathcal{P}2$) *Past dominance*: if the algorithm

A executes job j at time $t < \tau$, then γ_j remains dominant at t. In addition, the primal solution for $t < \tau$ does not change due to the release of z; and ($\mathcal{P}3$) *Completion*: $\gamma(t) = 0$ for all $t > C_{\max}$, where $C_{\max}$ is the completion time of the last job.

Theorem 1. *Any algorithm that satisfies the properties* ($\mathcal{P}1$)*,* ($\mathcal{P}2$) *and* ($\mathcal{P}3$) *with respect to a feasible dual solution is an optimal online algorithm for the fractional GFP problem with general cost functions* g*.*

Corollary 2.
(i) HDF is optimal for the fractional online GFP problem with linear cost function.
(ii) HDF is optimal for the fractional online problem of minimizing $\sum_{j\in\mathcal{J}} w_j g(C_j)$*.*
(iii) Adapted HDF is $\max_j \frac{\max_i b_{ij}}{\min_i b_{ij}}$*-speed 1-competitive for the online Packing Scheduling Problem problem of minimizing* $\sum_{j\in\mathcal{J}} \delta_j \int_{r_j}^{\infty}(t-r_j)x_j(t)dt$ *subject to packing constraints* $\{B\mathbf{x} \leq 1, \mathbf{x} \geq 0\}$ *which must be upheld at all times, where* $B = \{b_{ij} > 0\}$*.*
(iv) FIFO (resp. LIFO) is optimal for the fractional online GFP problem with convex (resp. concave) costs functions and jobs of equal density.

In order to allow for competitive algorithms, we relax certain properties: ($\mathcal{Q}1$) if the algorithm A schedules job j at time $t \geq \tau$ then $\gamma_j(t) \geq 0$ and $\lambda_j \geq \gamma_{j'}(t)$ for every other pending job j' at time t; ($\mathcal{Q}2$) if the algorithm A schedules job j at time $t < \tau$, then $\gamma_j(t) \geq 0$ and $\lambda_j \geq \gamma_{j'}(t)$ for every other pending job j' at time t. In addition, the primal solution for $t < \tau$ is not affected by the release of z; and ($\mathcal{Q}3$) $\gamma(t) = 0$ for all $t > C_{\max}$.

Theorem 3. *Any algorithm that satisfies the properties* ($\mathcal{Q}1$)*,* ($\mathcal{Q}2$) *and* ($\mathcal{Q}3$) *with respect to a feasible dual solution is a* $\frac{1}{1-\epsilon}$*-speed* $\frac{1}{\epsilon}$*-competitive algorithm for the fractional GFP problem with general cost functions* g*.*

Corollary 4. *FIFO (resp. HDF) is* $\frac{1}{1-\epsilon}$*-speed* $\frac{1}{\epsilon}$*-competitive for the fractional online GFP problem with general cost functions and equal-density jobs (resp. for the fractional online GFP problem with concave cost functions).*

References

[1] S. Angelopoulos, G. Lucarelli, and N. K. Thang. Primal-dual and dual-fitting analysis of online scheduling algorithms for generalized flow-time problems. *CoRR*, arXiv:1502.03946, 2015.

[2] N. Bansal and H.-L. Chan. Weighted flow time does not admit $o(1)$-competitive algorithms. In *SODA*, pages 1238–1244, 2009.

[3] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs. Online weighted flow time and deadline scheduling. *J. Discrete Algorithms*, 4(3):339–352, 2006.

[4] K. Fox, S. Im, J. Kulkarni, and B. Moseley. Online non-clairvoyant scheduling to simultaneously minimize all convex functions. In *APPROX-RANDOM*, pages 142–157, 2013.

[5] S. Im, B. Moseley, and K. Pruhs. Online scheduling with general cost functions. *SIAM Journal on Computing*, 43(1):126–143, 2014.

[6] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

Decomposition algorithm for the single machine scheduling polytope

Ruben Hoeksma (Speaker) * Bodo Manthey* Marc Uetz*

1 Introduction

Given any point x in a d-dimensional polytope P, Carathéodory's Theorem implies that x is a convex combination of at most $d+1$ vertices of P. We are interested in an algorithmic version of Carathéodory's theorem for the *single machine scheduling polytope*, Q. More specifically, we are given a vector of positive processing times $p \in \mathbb{R}^n_+$ and some point x in the corresponding single machine scheduling polytope. Our goal is to compute an explicit representation of x by at most n vertices v^i of Q, such that $x = \sum_i \lambda_i v^i$ for $\lambda_i \geq 0$ for all i and $\sum_i \lambda_i = 1$. We refer to this problem as a *decomposition problem*.

These decomposition problems arise in the design of efficient mechanisms for optimization problems in private information settings. In such settings, some of the data, like job processing times, are private to the jobs. One approach for computing Bayes-Nash optimal mechanisms, that recently has received attention, is to use linear programming relaxations for the reduced form of the mechanism [1, 6]. This results in so-called interim allocations. To implement the optimal mechanism, the interim allocations need to be translated into a lottery over actual allocations. At this point one is confronted with a decomposition problem, where the fractional point can even lie in the interior of the polytope. We consider exactly this problem for the single machine scheduling polytope.

The single machine scheduling polytope itself is well understood [7]. In particular, it is known to be a polymatroid, and the separation problem for C can be solved in $\mathrm{O}(n \log n)$ time. Therefore, the existence of a polynomial time decomposition algorithm follows from the ellipsoid method [3]. A generic approach to compute a decomposition has been described by Grötschel, Lovász, and Schrijver [4]. We call this the *GLS method* in the following. Furthermore, an $\mathrm{O}(n^9)$ decomposition algorithm follows directly from work by Fonlupt and Skoda [2] on the intersection of a line with a (general) polymatroid using the GLS method. However, a closer look reveals that an $\mathrm{O}(n^3 \log n)$ implementation is also possible [6]. Still, this result is unsatisfactory in the following sense. For the permutahedron, Yasutake et al. suggested an $\mathrm{O}(n^2)$ decomposition algorithm [9]. The permutahedron is precisely the single machine scheduling polytope for the special case where all processing times are 1. Hence, a natural question is if their $\mathrm{O}(n^2)$ algorithm can be generalized to the scheduling polytope.[1]

Theorem 1 (Main result [5]) *There exists an* $\mathrm{O}(n^2)$ *time decomposition algorithm for the single machine scheduling polytope.*

*{r.p.hoeksma,b.manthey,m.uetz}@utwente.nl. Department of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.

[1] The results from this abstract where previously presented at ISCO 2014 and published as [5].

Note that $O(n^2)$ is linear in the output of the algorithm, since, in general, it outputs n vertices, each with n components.

2 The single machine scheduling polytope

In this paper, we represent a schedule by h, the vector of half times, instead of by a vector of completion times, which is more commonly used. The *half time* of a job is the time at which the job has finished half of its processing. We have

$$h_j = s_j + \frac{1}{2}p_j = c_j - \frac{1}{2}p_j \ ,$$

where s_j, h_j, c_j and p_j are Job j's start time, half time, completion time and processing time, respectively.

According to the well known formulation from Queyranne [7], the single machine scheduling polytope for half times can be described by a set of inequalities, containing $n-1$ inequalities for each permutation of the jobs. Let Q denote this polytope.

3 Zonotopes and barycentric subdivision

The algorithm that we present in this work makes heavy use of the fact that the single machine scheduling polytope is a, so-called, *zonotope*. There are several equivalent definitions for zonotopes of which we give the following.

Definition 2 (Centrally symmetric polytope, zonotope) *Let $P \subseteq \mathbb{R}^n$ be a polytope. P is* centrally symmetric *if it has a center $c \in P$, such that $c + x \in P$ if and only if $c - x \in P$. If all faces of P are centrally symmetric, then P is called a* zonotope.

Note that any center c of a face of a zonotope P can be expressed as $c = \frac{1}{2}(v_1 + v_2)$, where v_1 and v_2 are vertices of P.

Lemma 3 (Queyranne & Schulz [8, Thm. 4.1]) *The single machine scheduling polytope is a zonotope.*

Next, consider for each half time vector an ordering of the jobs according to their half time. We divide Q into subpolytopes, such that each subpolytopes contains all vectors for which this ordering is the same. Note that some vectors are contained in multiple subpolytopes. Each subpolytope of this division contains exactly one vertex of Q and each such vertex represents a permutation schedule. That is, for each permutation of the jobs, the schedule that processes the jobs non-preemptively in that order results exactly in a half time vector which is a vertex of Q. We refer to the subpolytope that contains v as Q_v. The vertices of Q_v are exactly centers of Q.

The definition of Q_v allows us to describe it with a set of linear ordering inequalities. With these inequalities it is easy to compute the intersection of any line with Q_v.

4 The algorithm

Algorithm 1 describes a $O(n^2)$ time algorithm that decomposes a point $h^0 \in Q$ into a convex combination of vertices of Q. Figure 1 shows a visualization of this algorithm.

Algorithm 1 Decomposition algorithm

Given $h^0 \in Q$, determine vertex v with the same order as h^0 and the subpolytope Q_v. $h^0 \in Q_v$.
Apply the GLS method on Q_v and h^0:
while $h^i \neq$ the center of Q **do**
Choose vertex v^i of Q_v on a shared face of Q_v and Q.
The line from v^i through the incumbent point h^i intersects Q_v exactly in a point h^{i+1} such that there are $k, \ell \in \{1, \ldots, n\}$ with $h^{i+1}_k = h^{i+1}_\ell$ and $h^i_k = h^i_\ell$ and $h^i = \lambda^i v^i + (1 - \lambda^i) h^{i+1}$.
end while
Each vertex of Q_v can be written as $v^i = \frac{1}{2}(v + v^{i*})$, where v^{i*} is a vertex of Q.

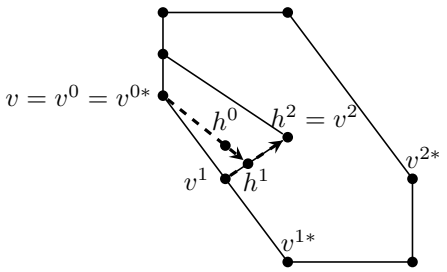


Figure 1: The decomposition algorithm on a scheduling polytope for three jobs.

References

[1] Y. Cai, C. Daskalakis and S. M. Weinberg (2012). *Optimal multi-dimensional mechanism design: Reducing revenue to welfare maximization.* In Proc. 53rd FOCS, pp. 130–139, IEEE, 2012.

[2] J. Fonlupt and A. Skoda (2009). *Strongly polynomial algorithm for the intersection of a line with a polymatroid.* In Research Trends in Combinatorial Optimization, pp. 69–85, Springer, 2009.

[3] M. Grötschel, L. Lovász and A. Schrijver (1981). *The ellipsoid method and its consequences in combinatorial optimization.* Combinatorica, pp. 1:169–197, 1981.

[4] M. Grötschel, L. Lovász and A. Schrijver (1988). *Geometric Algorithms and Combinatorial Optimization, vol. 2 of Algorithms and Combinatorics.* Springer, 1988.

[5] R. Hoeksma, B. Manthey and M. Uetz (2014). *Decomposition algorithm for the single machine scheduling polytope.* In: 3rd ISCO, pp. 280–291, Springer, 2014.

[6] R. Hoeksma and M. Uetz (2013). *Two Dimensional Optimal Mechanism Design for a Sequencing Problem.* In: Proc. IPCO, pp. 242–253, Springer, 2013.

[7] M. Queyranne (1993). *Structure of a simple scheduling polyhedron.* Math. Prog. 58, 1993, pp. 263–285.

[8] M. Queyranne and A. S. Schulz (1994). *Polyhedral approaches to machine scheduling.* Preprint 408-1994, TU Berlin, 1994.

[9] S. Yasutake, K. Hatano, S. Kijima, E. Takimoto and M. Takeda (2011). *Online linear optimization over permutations.* In: Algorithms and Computation, pp. 534–543, Springer, 2011.

Scheduling with state-dependent machine speeds

Veerle Timmermans (Speaker) * Tjark Vredeveld

1 Introduction

In queueing theory, many studies have been made about queues with state-dependent service speeds, see e.g. Bekker and Boxma [5] or Bekker, Borst, Boxma and Kella [4] and the references therein. This model is a.o. motivated by Bertrand and Van Ooijen [6] through human servers who may be slow when there is much work to do, due to stress, or when there is little work to do due to laziness. For state-dependent server speeds in packet-switched communication systems, we refer to [7, 8, 10, 12].

Although these models have been extensively studied in queueing theory, not much is known about algorithms that solve these models to optimality nor the computational complexity of this type of problem. During the 2013 Scheduling workshop in Dagstuhl, Urtzi Ayesta [3] posed it as an open question how optimal policies look like and what the computational complexity is. In this paper, we settle this open problem for one variant of state-dependent machine speeds, namely when the speed of the machine varies with the number of jobs in the system. This number of jobs is a good measure for the total workload in the system, when the service requirements of the jobs are i.i.d. Moreover, in this setting the speed of the server only changes at discrete times, see e.g. [5].

Related work. Models with workload dependent server speeds originate from Queueing Theory. Bertrand and Van Ooijen [6] assumes in his paper that the workload level affects the effective processing times in a job shop. This assumption is based on the results of empirical research on the relationship between workload and shop performance. Bekker, Borst, Boxma and Kella [4] considered two types of queues with workload dependent arrival rate and service speed. Bekker and Boxma [5] considered a queueing system where feedback information about the level of congestion is given right after arrival instants. When the amount of work right after arrival is at most some threshold, then the server works at a low speed until the next arrival instant.

Related work in deterministic scheduling with varying machine speeds includes the following. Research into speed scaling algorithms started with the work of Yao, Demers and Shenker [13], where each job is to be executed between its arrival time and deadline by a single processor with variable speed, which the scheduler also needs to decide on. A review paper on a.o. speed scaling algorithm is written by Albers [1].

Megow and Verschae [11] also study scheduling problems on a machine of varying speed, but they assume a speed function that depends on the time which is known a priori. They developed a PTAS for minimizing the total weighted completion time.

*{v.timmermans, t.vredeveld}@maastrichtuniversity.nl. Department of Quantitative Economics, Maastricht University, P.O.Box 616, 6200MD, Maastricht, The Netherlands.

The machine speed model we consider, was previously investigated by Gawiejnowicz [9], but he considered the makespan objective whereas we study the goal to minimize the sum of (weighted) completion times.

Model definition. In the model under consideration, n jobs need to be scheduled on a single machine. A job j is associated with a processing requirement denoted by p_j and depending on the variant that we consider also with a weight w_j. All jobs as well as the machine are available from the beginning. The machine is allowed to preempt a job, i.e., the processing of a job may be interrupted and resumed later on the machine. By allowing infinitesimally small processing on a job before preempting it, the preemption model can be viewed as one in which during each time interval the processing capacity of the machine is divided over one or more jobs. The goal is to minimize the total (weighted) completion time. In our scheduling model, the speed at which the machine processes its jobs varies with the number of unfinished jobs in the system. Hereto, we are given a speed function s_i $(i = 1, \ldots, n)$, where s_i denotes the speed of the machine when $i - 1$ jobs have been completed. Note that, as all jobs are available from the beginning, when the machine is running on speed s_i there are $n + 1 - i$ jobs in the system.

2 Our results

Assuming that we know the order in which the jobs complete, we can formulate the problem as an LP. Hereto, assume that the jobs complete in order of their index. We define variables $\Delta_j = C_j - C_{j-1}$ $(j = 1, \ldots, n)$, where we define $C_0 = 0$. Then, the completion time of job j is the sum of all Δ_k from $k = 1, \ldots, j$. The LP has two type of constraints. The first type of constraints ensure that the amount of processing done during an interval is not more than the capacity of the machine in that interval. The second type of constraints are non-negativity constraints on Δ_j ensuring that the jobs complete in the order that is assumed. Any solution to our scheduling problem can be represented as a solution to the LP and vice versa. Using this fact, we can prove the following key lemma.

Lemma 1 *There exists an optimal solution such that at every completion time C_j, any job k is already completed by time C_j or it has not received any processing yet by this time.*

Intuitively this lemma means that there exists an optimal solution where the jobs are partitioned in groups and all jobs in a group are processed at the same time. Given the order of job completions, we still need to decide on how to partition the jobs into groups. Although this follows from the optimal LP solution, we also developed a greedy algorithm. At stage i it calculates the objective value for two possibilities: putting all jobs that are not processed yet in a new group or putting them all in the old group. It turns out this will construct an optimal schedule for the given order on the job completions.

For the case of the total completion time objective, it is easy to show that jobs will complete in shortest processing time (SPT) order, whereas the case of unit processing times the optimal order will be sorting according to non-increasing weight. When both weight and processing times can be arbitrary, the problem is NP-hard and we make a reduction from 3-partition.

Acknowledgements

We thank Urtzi Ayesta for helpful discussion after posing this open question during the Dagstuhl Seminar 13111 "Scheduling" in 2013. Furthermore, we thank the organizers of this seminar and Schloss Dagstuhl for providing the right atmosphere to facilitate research.

References

[1] S. Albers. Energy efficient algorithms. *Communications of the ACM*, 53:86 – 96, 2010.

[2] S. Albers, O.J. Boxma, and K. Pruhs. Scheduling Dagstuhl Seminar 13111. *Dagstuhl Reports*, 3:24 – 50, 2013.

[3] U. Ayesta. Scheduling with time-varying capacities. Section 3.3 in [2], 2013.

[4] R. Bekker, S.C. Borst, O.J. Boxma, and O. Kelly. Queues with workload-dependent arrival and service rates. *Queueing SYstems*, 46:537 – 556, 2004.

[5] R. Bekker and O.J. Boxma. An M/G/1 queue with adaptable service speed. *Stochastic Models*, 23:373 – 396, 2007.

[6] J.W.M. Bertrand and H.P.G. Van Ooijen. Workload based order release and productivity: a missing link. *Production Planning and Control : The Management of Operations*, 13:665 – 678, 2002.

[7] A. Ewalid and D. Mintra. Analysis and design of rate-based congestion control of high-speed networks I: stochastic fluid models, acces regulation. *Queueing Systems*, 9:29 – 64, 1991.

[8] A. Ewalid and D. Mintra. Statistical multiplexing with loss priorities in rate-baed congestion control of high-speed networks. *IEEE Transactions on Communications*, 42:2989 – 3002, 1994.

[9] S. Gawiejnowicz. A note on scheduling on a single processor with speed dependent on a number of executed jobs. *Information Processing Letters*, 57:297 – 300, 1996.

[10] M. Mandjes and D. Mintra. A simple model of network access: feedback adaptation of rates and admision control. In *Proceedings of Infocom*, pages 3 – 12, 2002.

[11] N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In *Automata, Languages and Programming (ICALP 2013)*, volume 7965 of *LNCS*, pages 745 – 756. Springer, 2013.

[12] K.A. Ramanan and A. Weiss. Sharing bandwidth in atm. In *Proceedings of the Allerton Conference*, pages 732 – 740, 1997.

[13] F.F. Yao, A.J. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 374 – 382, 1995.

High Multiplicity Scheduling with Sequencing Costs

Michaël Gabay * Alexander Grigoriev † Vincent J.C. Kreuzen (Speaker) †
Tim Oosterwijk †

1 Introduction

In the current competitive economy, companies need to be aware of multiple objectives such as decreasing costs and enhancing customer service. Among the core activities of many companies and supply chains are mechanisms to match supply with demand, to prevent stock-outs and to cut back unnecessary overhead costs. Production companies are required to conduct extensive research into cost reduction to remain competitive within the market.

One of these well-studied problems in operations research is the *capacitated lot-sizing problem*, where one machine needs to produce a set of products to minimize average holding and setup costs. In this problem, the ongoing production of a product can be represented as the repeated scheduling of a single job on the machine, enabling a highly compact encoding of the input of the problem known as *high multiplicity scheduling*. Jobs in the high multiplicity setting are represented by a single job description with a multiplicity, representing the number of individual jobs. Obviously, the input length of the traditional setting can be exponentially larger than the length of the high multiplicity input, allowing for exponentially slower algorithms and raising doubts as to the applicability of conventional encoding for practical high multiplicity problems. For many companies, the high multiplicity encoding is a natural way to provide input from real-world data, especially if thousands of jobs are identical. Our research was inspired by one of such practical applications, a multinational textile company posed the problem of finding the optimal cycle length for their production, of only three types of lycra in extremely large quantities on a single machine.

In this paper we study an extended version of the aforementioned real-life problem, the capacitated lot-sizing problem with *sequence-dependent setup costs*. In this problem we have a single machine that is capable of producing a single product at any given time and a set of products that need to be produced. Each product is associated with a demand rate, a maximum production rate and inventory holding costs per unit. The objective is to find a cyclic schedule such that the demand of every product is met, minimizing the average costs. For any schedule, sequence-dependent setup costs referred to as *sequencing costs* are incurred each time the machine switches production between two different products. Moreover, input is provided under high multiplicity encoding.

*michael.gabay@artelys.com. Laboratoire G-SCOP 46, avenue Félix Viallet - 38031 Grenoble Cedex 1, France

†{a.grigoriev,v.kreuzen,t.oosterwijk}@maastrichtuniversity.nl. School of Business and Economics, Maastricht University, Operations Research Group, School of Business and Economics, PO Box 616, 6200 MD, Maastricht, The Netherlands

Related Work Some problems related to the capacitated lot-sizing problem with sequencing costs have been investigated in a number of variants, see e.g. [7, 4, 2].

Most papers on high multiplicity scheduling consider discrete variants, in which time and/or quantities are discretized into units. There has also been some work considering a continuous setting, in which production can start and stop at any time, e.g. with fluids. Bertsimas et al. [1] consider the high multiplicity job-shop problem without sequencing costs, and use this continuous setting as a relaxation for the original discrete job-shop problem. They round an optimal solution for the fluid problem to an asymptotically optimal solution for the discrete problem, and provide some computational experiments. In another work on the continuous setting, Haase [5] discusses a problem very closely related to ours, where production rates are fixed. He proposes a local-search heuristic and evaluates it by comparing it to optimal solutions for small instances. Haase and Kimms [6] consider the same problem and, by making additional assumptions on the input instances, solve the problem to optimality. They present a Mixed Integer Programming formulation for their model and a fast enumeration scheme, which they evaluate by a computational study.

Brauner et al. [3] provide a detailed framework for the complexity analysis of high multiplicity scheduling problems. We refer the reader to this paper for an excellent survey of related work in this field.

2 Results

The Model We study several variants of the single machine capacitated lot sizing problem with sequence-dependent setup costs and product-dependent inventory costs. Here we are given a single machine and k types of products that need to be scheduled, where each product i is associated with a constant demand rate d_i, production rate p_i and inventory costs per unit h_i. When the machine switches from producing product i to product j, sequencing costs $s_{i,j}$ are incurred. The goal is to find a schedule such that demand is met at all times and the average per-time-unit costs are minimized. This can be seen as lifting a conventional scheduling problem to its more general high multiplicity counterpart where there are only a few job types, but each with a high multiplicity (large number of occurrences of the same type). This severely increases the complexity of the problem.

We distinguish three cases. In the continuous case the machine can switch products at any time and it can produce at most p_i units of product i. In the discrete case the machine can only switch products at the end of every unit of time (e.g. a day) and it can produce at most p_i units of product i. In the fixed case the machine can only switch products at the end of every unit of time and if it produces product i during that unit of time, it has to produce exactly p_i units.

Structural Properties As our main contribution we present structural properties for the three variants. We prove strongly NP-hardness by reducing the problem to the Traveling Salesman Problem, proving that even without considering the high multiplicity encoding of the input, the problem is already NP-hard. Furthermore we characterize feasible instances and present a number of lemmata which largely characterize optimal solutions. We provide an optimal simple schedule for identical products, a common real-world input, and give a lower bound on the cost of the optimal schedule for the

continuous case, by replacing products in a sequence with the cheapest product in terms of holdings costs. We then characterize optimal solutions for small number of products, where the fixed case for $k = 1$, a single product, is already non-trivial.

Due to the hardness found in different levels of the problem, computing even a feasible solution can be tough, and finding an optimal solution might be impossible. Restricting the problem to instances with a fixed time horizon enables the use of different techniques applicable only when working with a bounded cycle length.

References

[1] Dimitris Bertsimas, David Gamarnik, and Jay Sethuraman. From fluid relaxations to practical algorithms for high-multiplicity job-shop scheduling: The holding cost objective. *Operations Research*, 51(5):798–813, 2003.

[2] Fayez Fouad Boctor. The two-product, single-machine, static demand, infinite horizon lot scheduling problem. *Management Science*, 28(7):798–807, 1982.

[3] Nadia Brauner, Yves Crama, Alexander Grigoriev, and Joris Van De Klundert. A framework for the complexity of high-multiplicity scheduling problems. *Journal of combinatorial optimization*, 9(3):313–323, 2005.

[4] SK Goyal. Scheduling a multi-product single machine system. *Journal of the Operational Research Society*, 24(2):261–269, 1973.

[5] Knut Haase. Capacitated lot-sizing with sequence dependent setup costs. *Operations-Research-Spektrum*, 18(1):51–59, 1996.

[6] Knut Haase and Alf Kimms. Lot sizing and scheduling with sequence-dependent setup costs and times and efficient rescheduling opportunities. *International Journal of Production Economics*, 66(2):159 – 169, 2000.

[7] JG Madigan. Scheduling a multi-product single machine system for an infinite planning period. *Management Science*, 14(11):713–719, 1968.

Group-dependent Models for Single Machine Scheduling with Changing Processing Times and Rate-modifying Activities

Kabir Rustogi (Speaker) * Vitaly A Strusevich †

1 Introduction

In classical scheduling models, it is normally assumed that the processing times of jobs are fixed. However, one of the current trends in deterministic scheduling research is to investigate scheduling problems with variable processing times. Some of the common rationales provided for considering such models are as follows: the machine conditions may deteriorate as more jobs are processed, resulting in higher than normal processing times, or conversely, the machine's operator may gain more experience as more jobs are processed, so she/he can process the jobs faster. However, in real-life situations, it is often observed that the machines/operators are subject to periodic maintenance activities or replacements, which modify the rate of change of processing times by disrupting the learning/deterioration process.

Active research in this area has lead to a vast body of publications on scheduling with changing processing times and rate-modifying activities. In this talk, we highlight some of our recent work which aims to create a unified modelling framework that enables us to generalise previously known results and systematically study a wide range of scheduling models, while ensuring computational efficiency of the algorithms used.

2 Formulation and Scope of Problem

Jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on a single machine without preemption. It is assumed that the machine is in a perfect processing state at time zero and the processing conditions change according to a certain deterministic rule. Each job $j \in N$ is associated with a positive number p_j, which can be interpreted as its processing time under *normal* machine conditions. However, the processing times may change as a function of the job's location in the schedule or as a function of the time a machine has spent processing jobs, or a combination of both.

Additionally, we also allow a schedule to be affected by certain rate-modifying periods (RMPs), e.g., machine maintenance or change of worker. We consider a general situation, in which the decision-maker is presented with a total of $K \geq 0$ possible RMPs, which

*K.Rustogi@gre.ac.uk. Department of Mathematical Sciences (FACH), University of Greenwich, Old Royal Naval College, Park Row, London SE10 9LS, United Kingdom.

†V.Strusevich@gre.ac.uk. Department of Mathematical Sciences (FACH), University of Greenwich, Old Royal Naval College, Park Row, London SE10 9LS, United Kingdom.

can be either distinct or alike. If, out of the available K RMPs, $k-1$ of them are selected and included in a schedule, then the jobs in that schedule will be divided into k, $1 \leq k \leq K+1$ groups, one to be scheduled before the first RMP and one after each of the $k-1$ RMPs. Each RMP can have a different effect on the machine conditions, so that they do not necessarily restore the machine to its default "as good as new". It follows that jobs sequenced in different groups may be subject to different effects, which makes the duration of a job not only dependent on the location of the job within a group, but also on the location and type of a group. We refer to such an effect as a *group-dependent* effect.

To allow such a general situation, we introduce a range of group-dependent scheduling models, classified as follows:

1. **Position-dependent effects:** The actual processing time of job $j \in N$, that is sequenced in the position r of the x-th group is be given by

$$p_j^{[x]}(r) = p_j g_j^{[x]}(r), \ j \in N, \ 1 \leq r \leq n, \ 1 \leq x \leq k, \ 1 \leq k \leq K+1,$$

where the function $g_j^{[x]}(r)$ may be defined appropriately to become either job-dependent, group-dependent, or to represent a learning or deterioration effect. This model is the most general representation for positional effects found in current scheduling literature.

2. **Time-dependent effects:** The actual processing time of job $j \in N$, starting at time τ of the x-th group is be given by

$$p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, \ j \in N, \ \tau \geq 0, 1 \leq x \leq k, \ 1 \leq k \leq K+1,$$

where the rate $a^{[x]}$ may be defined appropriately to become either group-dependent or to represent a learning or deterioration effect.

3. **Combined effects:** Such a model combines the first two models and provides a very general formulation to represent almost any job-independent effect, including instances in which learning and deterioration effects are applied simultaneously. Suppose there exists a permutation $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$ of jobs, where $\pi^{[x]}$ represents a permutation of jobs in group x, $1 \leq x \leq k$. Let each group contain a total of $n^{[x]}$, $1 \leq x \leq k$, jobs, so that $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$. The actual processing time of a job $j = \pi^{[x]}(r)$ sequenced in position r of group x, is defined by

$$\begin{aligned} p_j^{[x]}(r) = \ & \left(p_{\pi^{[x]}(r)} + a_1^{[x]}F_1 + a_2^{[x]}F_2 + \ldots + a_{x-1}^{[x]}F_{x-1} + a_x^{[x]}F_{(x,r-1)}\right) g^{[x]}(r), \\ & 1 \leq r \leq n^{[x]}, 1 \leq x \leq k, \ 1 \leq k \leq K+1, \end{aligned}$$

where F_x, $1 \leq x \leq k$, represents the time it takes to complete all jobs in a group x, $1 \leq x \leq k$, and $F_{(x,r)}$ represents the time taken to complete the first r jobs in the $x-$th group, such that $F_0 = F_{(x,0)} = 0$. The factor $g^{[x]}(r)$ is a positive, possibly non-monotone, group-dependent job-independent positional factor and the factors $a_1^{[x]}, a_2^{[x]}, \ldots, a_x^{[x]}$ are real numbers and represent group-dependent rates that determine how the length of previous groups affects a job's processing time.

Furthermore, these effects are combined with various types of RMPs. For each type of effect, we consider several versions of the model, distinguishing them based on three criteria: **(i)** RMPs are able to fully restore machine conditions or not; **(ii)** RMPs are identical or distinct; **(iii)** Duration of the RMPs are constant or start-time dependent given by $D_x = \alpha\tau + \beta$,where τ is the start-time of the RMP, measured from the time the previous RMP was completed, and α and β are parameters that define the RMP.

For each of these situations, we focus on two classical objective functions: the makespan and the sum of completion times, and develope general principles of designing the algorithms for handling various types of the generalised problems, for these two objective functions. The optimal solution provides us with two pieces of information: **(i)** an optimal permutation of $k-1$ RMPs, from K available RMPs; **(ii)** an optimal permutation of n jobs.

The problems outlined above encompass almost every job-independent scheduling model and a majority of job-dependent models as well. Traditionally, these models were studied independently of each other and were restricted to very special cases only. Our work has attempted to bring together seemingly unrelated families of models under one umbrella and solve all types of problems using a unified algorithmic framework. Developing such a framework enables us to tackle many practical scheduling problems which were previously ignored.

3 Results

The main results obtained from this study involves the development of polynomial-time algorithms that solve a general class of scheduling problems as defined above. The papers [1, 2, 3, 4] present our results on 4 such classes of problems. In each of these papers, it was noticed that all the problems considered reduce to some form of the assignment problem: square, rectangular or with a product matrix. The challenge was to identify the structure of each problem and appropriately classify them. Once the problems were classified it was possible to generalise previously known results and extend the boundaries in terms of efficiency in algorithm design. In fact, the range of developed algorithms not only match the running times of existing algorithms known for simpler problems, but demonstrate improved performance.

References

[1] K. Rustogi and V.A. Strusevich (2012). Single machine scheduling with general positional deterioration and rate-modifying maintenance, *Omega*, 40, 791–804.

[2] K. Rustogi and V.A. Strusevich (2012). Simple matching vs linear assignment in scheduling models with positional effects: A critical review, *European Journal of Operational Research* 222 (3), 393–407.

[3] K. Rustogi and V.A. Strusevich (2013). Combining time and position dependent effects on a single machine subject to rate modifying activities, *Omega*, 42, 166–178.

[4] K. Rustogi and V.A. Strusevich (2015). Single machine scheduling with time-dependent deterioration and rate-modifying maintenance. *Journal of the Operational Research Society*, 66, 500–515.

Energy-efficient Task Execution to Cope with Timing Errors

Aurélien Cavelan (Speaker) * Yves Robert † Hongyang Sun ‡
Frédéric Vivien §

1 Introduction

Energy considerations are unavoidable nowadays, for both economical and environmental reasons. In this paper, we investigate whether one can aggressively use *voltage overscaling* [1, 3, 2], in a purely algorithmic/software-based approach [4], to reduce the energy cost of executing a chain of tasks. Voltage and frequency cannot be set independently and at any value. For any frequency value, there is a minimal *threshold* or *nominal* voltage V_{TH}, at which the core can safely be used. If the core is used at a voltage below that limit, *timing errors* could happen, that is, the results of some logic gates could be used before their output signals reach their final values. Since timing errors are essentially silent data corruptions (SDC), they do not manifest themselves until the corrupted data is activated and/or leads to an unusual application behavior, wasting the entire computation done so far. Therefore, a verification mechanism is necessary to ensure timely detection of these errors. In this work, we assume such a verification mechanism exist and can be used to verify the integrity of a task.

Unlike other soft errors caused by, e.g., electro-magnetic radiation or cosmic rays, which create single event upset to the system in a somewhat random manner, timing errors are more *deterministic* in nature: if the very same computation is performed in the very same context (temperature, voltage, operands, content of registers, history of instructions, etc.), the very same faulty result will be produced. Because timing errors are reproducible[1], we have no choice but to re-execute faulty computations in a different context. The rough idea would be to execute a task at a very low voltage and to check its correctness with a verification. If the result were incorrect, we would then recompute it at a higher voltage, or in the worst case at the nominal voltage. We assume the energy cost of executing a task at each available voltage is known, so is the probability of encountering SDC at each voltage. The scheduling problem is then to derive an optimal sequence of voltages, that is, knowing these costs and probabilities, at which voltage to start executing a task and at which voltage to re-execute them in case of failure (should we go directly for the nominal voltage or should we risk once again an execution at a voltage below threshold?).

*aurelien.cavelan@ens-lyon.fr. Ecole Normale Superieure de Lyon, France & INRIA, France.
†yves.robert@inria.fr. Ecole Normale Superieure de Lyon, France & University of Knoxville, USA.
‡hongyang.sun@ens-lyon.fr. Ecole Normale Superieure de Lyon, France.
§frederic.vivien@inria.fr. INRIA, France.

[1]Note that, although timing errors are deterministic, they cannot be forecasted in practice. Indeed, this would require considering any potential context of any processing: voltage, temperature, operands, content of registers, etc.

Platform We set the frequency to be the lowest possible one in the system. The platform can choose an operating voltage among a set $\mathcal{V} = \{V_1, V_2, \cdots, V_k\}$ of k discrete values, where $V_1 < V_2 < \cdots < V_k$. Each voltage V_ℓ has a *failure probability* p_ℓ and we assume that the highest voltage V_k is the *nominal* voltage $\mathrm{V_{TH}}$ with failure probability $p_k = 0$. We model timing errors based on the following assumptions: (i) given a computation and an input I, there exists a *threshold voltage* $\mathrm{V_{TH}}(I)$: using any voltage V below the threshold ($V < \mathrm{V_{TH}}(I)$) will always lead to an incorrect result, while using any voltage above that threshold ($V \geq \mathrm{V_{TH}}(I)$) will always lead to a successful execution; and (ii) there is a probability p_V that the computation fails under a given voltage V, i.e., produces at least one error on a random input. To switch the operating voltage also incurs an energy cost. Let $o_{\ell,h}$ denote the energy consumed to switch the system operating voltage from V_ℓ to V_h.

Applications We model the computation workflow as a task graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$ that contains a set $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$ of n tasks, and each task represents a block of computation. The precedence constraint forms a linear chain of tasks, i.e., $\mathcal{E} = \cup_{i=1}^{n-1}\{T_i \prec T_{i+1}\}$. In this paper, we assume that all the tasks have the same amount of computation to be done, including the work to verify the correctness of the result at the end. Hence, they also share the same execution time and energy consumption under a fixed voltage and frequency setting.

Optimization Problem We define a schedule as a sequence of voltages. The execution of a task starts with the first voltage of the sequence: if the execution fails, the task is re-executed with the second voltage, etc. Upon successful execution of a task, we simply move to the following one. The objective is to find, for each task, the schedule that minimizes the total expected energy consumption needed to execute the chain.

Full proofs and simulations for the sections below can be found in the full version of the paper at `http://perso.ens-lyon.fr/aurelien.cavelan/publications/RR-8682.pdf`

2 Conditional Probabilities

We show from the assumptions on timing errors in Section 1 that, for a given input, the probability of having a timing error only depends on the last voltage used:

Lemma 1. *Consider a sequence of m voltages $\langle V_1, V_2, \cdots, V_m\rangle$, where $V_1 < V_2 < \cdots < V_m$ under which a given task is going to be executed.*

(i) For any $1 \leq \ell \leq m$, given that the execution of the task has already failed under voltages $V_0, V_1, \cdots, V_{\ell-1}$, the probability that the task execution will fail under voltage V_ℓ on the same input is

$$\mathbb{P}(V_\ell\text{-fail} \mid V_0 V_1 \cdots V_{\ell-1}\text{-fail}) = \frac{p_\ell}{p_{\ell-1}}$$

(ii) For any voltage V_ℓ, where $1 \leq \ell \leq m$, let $\mathbb{P}(V_\ell\text{-fail})$ denote the probability that the task execution fails at all voltages $V_0, V_1, \cdots, V_\ell$, and let $\mathbb{P}(V_\ell\text{-succ})$ denote the probability that the task execution fails at voltages $V_0, V_1, \cdots, V_{\ell-1}$ but succeeds at V_ℓ. We have

$$\begin{aligned} \mathbb{P}(V_\ell\text{-fail}) &= p_\ell \\ \mathbb{P}(V_\ell\text{-succ}) &= p_{\ell-1} - p_\ell \end{aligned}$$

3 Optimal Solution Via Dynamic Programming

We first consider the problem for a single task, as the solution for a linear chain of tasks is an extension of this algorithm. The main result is the following:

Theorem 1. *To minimize the expected energy consumption for a single task, the optimal sequence of voltages to execute the task with a preset voltage $V_p \in \mathcal{V}$ of the system can be obtained by dynamic programming with complexity $O(k^2)$.*

The main problem when considering a chain of tasks is the voltage switching cost. In other words, the schedule for one task (i.e., the voltage sequence) might be different from one task to another. This is due to the difference between the ending voltage of one task and the starting voltage of the next task. Let us say task one ends at voltage V_2. The optimal sequence for one task starts at voltage V_1, therefore we should pay the switching cost to decrease the voltage from V_2 to V_1. If the cost is too important, then it might be more efficient to start directly at voltage V_2. We have the following result:

Theorem 2. *To minimize the expected energy consumption for a linear chain of tasks, the optimal sequence of voltages to execute each task, given the terminating voltage of its preceding task or in case of the first task the preset voltage V_p of the system, can be obtained by dynamic programming with complexity $O(nk^2)$.*

Finally, experimental simulations available in the full version of the paper show possible improvements between the algorithm for a linear chain of tasks and an execution at nominal voltage. The problem remains open for general task graphs, though for independant tasks, we already know that the current solution for one task is not optimal.

References

[1] G. Karakonstantis and K. Roy. Voltage over-scaling: A cross-layer design perspective for energy efficient systems. In *European Conference on Circuit Theory and Design (ECCTD)*, pages 548–551, 2011.

[2] P.K. Krause and I. Polian. Adaptive voltage over-scaling for resilient applications. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, 2011.

[3] S.G. Ramasubramanian, S. Venkataramani, A. Parandhaman, and A. Raghunathan. Relax-and-retime: A methodology for energy-efficient recovery based design. In *Design Automation Conference (DAC)*, pages 1–6, 2013.

[4] Tyler M. Smith, Enrique S. Quintana-Orti, Mikhail Smelyanskiy, and Robert A. van de Geijn. Embedding fault-tolerance, exploiting approximate computing and retaining high performance in the matrix multiplication. In *1st Workshop On Approximate Computing (WAPCO)*, 2015.

The Robust Weighted Vertex Coloring Problem with Uncertain Data

Robert Benkoczi [*] Ram Dahal (Speaker) [†] Daya Gaur [‡]

Introduction: The vertex coloring problem (VCP) is a classical problem of assigning a color to each vertex of a graph such that no two adjacent vertices have the same color. In the weighted version of classical VCP (first introduced by Guan and Zhu [3]), we are given an undirected weighted graph $G = (V, E)$ where each vertex $v \in V$ is assigned a positive weight w_v. The objective of the weighted vertex coloring problem (WVCP) is to minimize the sum of the costs of the colors from a feasible vertex coloring. The cost of a color equals the maximum vertex weight among all vertices assigned to the color. WVCP is strongly NP-hard since it generalize the classical VCP [7]. WVCP has several applications in scheduling, timetabling, register allocation, train platforming [3, 5]. In addition, the problem captures the essence of scheduling data transmissions in a time division multiple access (TDMA) wireless network [5, 6]. Such networks are deployed everywhere. For example, networks using Worldwide Interoperability for Microwave Access (WiMAX) family of communication standards provide a large portion of the data mobility services of the world. WiMAX uses TDMA technology but the standard does not specify particular resource allocation algorithms, specifically to allow the most flexible and efficient use of resources possible.

In our work we consider a much more difficult setting for WVCP called robust WVCP (RWVCP). In robust optimization problems, some of the input parameters of the problem instances are uncertain. For these parameters, values lie between known lower and upper bounds. Any particular assignment of values to the uncertain parameters in the allowed ranges is called a scenario. The goal is to find a solution robust to the uncertainty that is, a solution X for which the difference between the cost of X under the worst possible scenario and the cost of the optimal solution for that scenario is minimized. RWVCP is completely open at present.

Robust optimization is a very active area of research (see for example the survey of [2]) since uncertainty is inherent to many applications. However, the structure of the WVCP is very different from that of the majority of the robust problems studied in the literature. In the latter case, once a solution is fixed, finding one corresponding worst case scenario usually leads to an easier optimization problem. For WVCP, the only combinatorial approach for finding a worst case scenario seems to rely on brute-force

[*] robert.benkoczi@uleth.ca. Department of Mathematics and Computer Science, University of Lethbridge, 4401 University Dr W, Lethbridge, AB T1K 6T5, Canada.

[†] ram.dahal@uleth.ca. Department of Mathematics and Computer Science, University of Lethbridge, 4401 University Dr W, Lethbridge, AB T1K 6T5, Canada.

[‡] daya.gaur@uleth.ca. Department of Mathematics and Computer Science, University of Lethbridge, 4401 University Dr W, Lethbridge, AB T1K 6T5, Canada.

enumeration and is intractable. Moreover, the objective function of WVCP employs the *maximum* operator which disallows the use of a powerful operation: distributing the cost of the objective function among the elements that make up a feasible solution. For this reason, the 2-approximation algorithm for a large class of robust optimization problems given by Kasperski et al. [4] does not apply to robust WVCP.

In our apporach, we use Benders decomposition to generate the scenarios of interest but, unlike the robust quadratic assignment problem (RQAP) [1], our master problem is a linear program with an exponential number of variables and is solved by column generation. Fortunately for us, the column generation sub-problem is not complicated by the addition of constraints from the Benders decomposition. Another difference concerns the generation of worst case scenarios. For RQAP, this amounts to solving another instance of a deterministic QAP. In our formulation, worst case scenarios for RWVCP can be obtained by local search or greedy heuristics using the additional information available from the optimal primal-dual pair of the linear programming relaxation of RWVCP.

Integer Linear Programming Formulation:

$$\min_{r \in \mathbb{R}} \quad r \tag{1}$$

$$\text{subject to: } r \geq \sum_{\alpha \in X} x_\alpha \max_{v \in \alpha} \{w_v^s\} - F^*(s), \ \forall s \in \Gamma \tag{2}$$

$$\sum_{\alpha \in X : v \in \alpha} x_\alpha \geq 1, \qquad \forall v \in V \tag{3}$$

$$x_\alpha \in \{0, 1\} \tag{4}$$

We formulate RWVCP as the problem of covering the vertices of graph G by independent sets α from a set X of independent sets that are generated by our column generation procedure. The corresponding binary variable x_α indicates whether set α is used in the cover or not. An independent set α with $x_\alpha = 1$ corresponds to a color class in RWVCP. We denote by Γ the set of worst case scenarios generated by our Benders decomposition. We let w_v^s represent the weight of vertex v under a scenario $s \in \Gamma$ and $F^*(s)$ represent the cost of the optimal solution to WVCP for a scenario s. In the formulation below, r is a variable representing the regret of the solution encoded by variables x_α relative to the set of worst case scenarios in Γ. Constraints (2) enforce that r be the maximum regret over the scenarios in Γ. We note that the cardinality of sets X and Γ is, in the worst case, exponential in the problem size.

We relax constraints (4), to obtain a *master problem* which is solved iteratively. Each iteration consists of the following steps: (i) compute a fractional solution x^* for the *master problem*; (ii) solve the *slave problem* which generates a constraint (2) that is violated by x^*; (iii) include this constraint in the *master problem*. The algorithm stops when the slave problem cannot generate any new constraints. An integer solution to the RWVCP can be obtained from the solution x^* of the last iteration, by rounding.

The solution x^* from Step (i) is obtained by column generation. Let $\pi_s, s \in \Gamma$ and $\pi_v, v \in V$ be the dual variables corresponding to constraints (2) and (3) respectively. Then, the column generation sub-problem is to find an independent set α^* with minimum

reduced cost, that is, an optimal solution to the following *slave problem* in variable z_v:

$$\min \quad \sum_{s\in\Gamma} \pi_s \max_{v:z_v=1}\{w_v^s\} - \sum_{v\in V} \pi_v z_v \tag{5}$$

$$\text{subject to: } z_v + z_{v'} \leq 1, \qquad \forall (v,v') \in E \tag{6}$$

$$z_v \in \{0,1\} \tag{7}$$

Variable z_v for $v \in V$ is the indicator variable for the independent set α^*, and $\max_{v:z_v=1} w_v^s$ is the weight of an independent set under a scenario s. The polytope of the linear programming relaxation for this problem has integral solution for a large class of graphs, such as the perfect graphs. To generate a new scenario s^* at Step (ii) we consider the constraint (2) corresponding to a scenario s' that is tight for the optimal solution x^*. Using x^* and s', we can find a gradient vector $\delta(s', x^*)$ that will possibly increase the value of the regret for solution x^*. If the regret of x^* at scenario $s^* = s' + \delta(s', x^*)$ increases strictly, then constraint (2) corresponding to s^* will separate solution x^*.

Conclusion: The robust version of the problem is of special interest in networking applications since it aims at finding a schedule of transmissions that is robust to short term and unpredictable changes in data traffic patterns. In addition, our work advances the research on robust optimization problems with very difficult objective functions for which very little is known in the literature.

References

[1] Mohammad Javad Feizollahi and Igor Averbakh. The robust (minmax regret) quadratic assignment problem with interval flows. *INFORMS Journal on Computing*, 26(2):321–335, 2013.

[2] Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235(3):471–483, 2014.

[3] DJ Guan and Z. Xuding. A coloring problem for weighted graphs. *Information Processing Letters*, 61(2):77–81, 1997.

[4] Adam Kasperski and Paweł Zieliński. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Information Processing Letters*, 97(5):177–180, 2006.

[5] Enrico Malaguti, Michele Monaci, and Paolo Toth. Models and heuristic algorithms for a weighted vertex coloring problem. *Journal of Heuristics*, 15(5):503–526, 2009.

[6] Arunesh Mishra, Suman Banerjee, and William Arbaugh. Weighted coloring based channel assignment for wlans. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(3):19–31, 2005.

[7] Sriram V Pemmaraju and Rajiv Raman. Approximation algorithms for the max-coloring problem. In *Automata, languages and programming*, pages 1064–1075. Springer, 2005.

On-line maintenance scheduling *

Claudio Telha † Mathieu Van Vyve ‡

1 Introduction

We study the following problem from the perspective of on-line algorithms: a machine receives a sequence of requests at times $t_1 < t_2 < \ldots < t_n$, aiming to serve as many of them as possible. Serving or rejecting a request only depends, respectively, on whether the machine is *operational* or *out-of-service* at the time the request arrives. After L requests have been served uninterruptedly in operational status, the machine becomes out-of-service. A maintenance job puts the machine out-of-service for T units of time, after which it becomes operational once again. We assume that the machine starts operational.

In the off-line version of the problem, given the service requests times $\{t_i\}_{i=1..n}$, the problem is to find a set of maintenance times so that the number of requests served by the machine is maximum; this can be efficiently done via dynamic programming. In the on-line version of the problem, carrying a maintenance at a time t must be decided without any information about the future service requests $\{t_i : t_i > t\}$. To deal with the uncertainty, we analyze the algorithms from the perspective of competitive analysis.

An on-line algorithm for a maximization problem is c-competitive when it produces solutions whose cost is at least c-times the off-line optimum. The framework of competitive algorithms [1] has been studied primarily in computer science, and more recently also in operations research. It provides advantages such as the no assumptions on the distribution of the uncertain data, and the frequent simplicity of its algorithms.

Our main result is a 0.585-competitive algorithm for the on-line maintenance problem, and an upper bound of 0.667 for its best competitive ratio. To our knowledge, competitive algorithms for this problem have not been considered before in the literature.

2 Results

It is easy to prove that placing maintenances exactly after L requests have been served leads to a $L/(2L-1)$-competitive algorithm. Moreover, this is the best competitive

*This text presents research results of the P7/36 PAI project COMEX, part of the Belgian Program on Interuniversity Poles of Attraction initiated by the Belgian State, Prime Minister Office, Science Policy Programming. The scientific responsibility is assumed by the authors.

†claudio.telha@uclouvain.be. Université Catholique de Louvain, Voie du Roman Pays 34, 1348 Louvain-la-Neuve, Belgium. Supported by FSR Incoming Post-doctoral Fellowship of the Catholic University of Louvain (UCL), funded by the French Community of Belgium.

‡mathieu.vanvyve@uclouvain.be. Université Catholique de Louvain, Voie du Roman Pays 34, 1348 Louvain-la-Neuve, Belgium.

ratio that can be obtained with deterministic algorithms. The only way to improve over this bound (that is asymptotically equal to 0.5 for L large), is through the use of randomization.

For maximization problems, a randomized on-line algorithm is c-competitive when it produces solutions whose expected cost is at least c-times the off-line optimum. It is well known that randomized algorithms can outperform deterministic algorithms in terms of competitive ratio; for our problem this is no exception:

Theorem 1 *Let x_i be i.i.d. random variables with the following distribution. It takes value $j \in \{K, \ldots, L-1\}$ with probability $\frac{1}{L}$ and it takes value L with probability $\frac{K}{L}$. Consider the algorithm that iteratively places a maintenance each time the machine has served x_i requests (incrementing i after each maintenance). This algorithm is c-competitive for $c = \frac{2KL+L(L-1)-K(K-1)}{2L(L+K-1)}$. The best value for K is $1 - L + \sqrt{2L(L-1)}$ rounded up or down, with c and $1 - \frac{K}{L}$ converging to $2 - \sqrt{2} \approx 0.585$ when $L \to \infty$.*

Note that the randomized algorithm just described behaves exactly as the deterministic algorithm when $x_i = L$, but otherwise it always preemptively places a maintenance. The analysis of the competitive ratio is technical, but it is based on a neat property, that essentially allows us to assume that in the bad instances for the algorithm, the off-line optimum serves all the requests.

On the other hand, we obtained an upper bound on the optimal competitive ratio for this problem by considering the behavior of an arbitrary algorithm against a fixed family of "simple" instances, and showing that in at least one of them the algorithm will not perform well. This simple approach gives the following result:

Theorem 2 *No randomized algorithm for the on-line maintenance scheduling problem can be f-competitive, for $f > \frac{2(L+1)}{3L+1}$.*

Table 1 summarizes the bounds obtained with randomized algorithms, for different values of L, according to Theorems 1 and 2.

L	LB	UB	L	LB	UB
1	1	1	11	0,606	0,706
2	0,750	0,857	12	0,604	0,703
3	0,667	0,800	13	0,603	0,700
4	0,650	0,769	14	0,602	0,698
5	0,633	0,750	15	0,600	0,696
6	0,625	0,737	16	0,599	0,694
7	0,619	0,727	17	0,598	0,692
8	0,614	0,720	18	0,598	0,691
9	0,611	0,714	19	0,597	0,690
10	0,608	0,710	∞	0,585	0,667

Table 1: Upper and lower bounds on the competitiveness of randomized algorithms.

Our results for the on-line maintenance scheduling problem leave room for improvement. In our view, improving either the lower or the upper bound in the competitive ratio may need to specifically consider algorithms that do not always place a maintenance just after serving a customer.

References

[1] Buchbinder, Niv and Naor, Joseph (2009). *The design of competitive online algorithms via a primal dual approach.* Foundations and Trends in Theoretical Computer Science, vol 3(2-3), p. 93-263.

Maximum Γ-robust flows over time

Corinna Gottschalk [*] Arie Koster [†] Frauke Liers [‡] Britta Peis [*]
Daniel Schmand [*] Andreas Wierz (Speaker) [*]

1 Introduction

Many applications in the context of routing or logistics call for a temporal component that is part of both, the input and the actual solution. In classical flow theory, flow traverses the network in a static fashion, that is, we are interested only in obeying capacity restrictions and maybe some additional constraints. In a real-world application, however, flow takes some time in order to traverse any of the network's arcs, hence, introducing a temporal dimension into the models. Dynamic flow problems take this into account and have been studied for over half a century [6]. However, despite the topic's relevancy and the existence of fascinating results, only few textbooks cover this topic. We refer to Skutella [7] for a comprehensive introduction to dynamic flow problems and we rely on their notation. The focus of our work is on robust approaches for disturbed dynamic flow problems, that is, travel times are subject to uncertainty.

For this work, we restrict ourselves to MAXIMUM FLOWS OVER TIME, a variant of dynamic flows which seeks to maximize the total throughput of a network. An instance consists of a directed graph $G = (V, E)$ with designated source and destination nodes $s, d \in V$ and a time horizon $T \in \mathbb{Z}_+$. Each arc is associated with a capacity $u : E \to \mathbb{Z}_+$ and a travel time $\tau : E \to \mathbb{Z}_+$. Flow entering an arc e at some time θ leaves the arc at its head at time $\theta + \tau_e$. The goal is to maximize the total amount of flow sent from node s to d within the time horizon while capacity restrictions are met during any point in time. It is common to assume that flow is relayed at intermediate nodes as soon as possible.

Although a temporal component is important, real-world applications are usually affected by measurement errors and a high degree of uncertainty, hence, indicating that the classical models are usually highly inaccurate. Especially road-traffic is affected by uncertain travel times due to accidents, traffic jams or even by the driver's condition. We provide a first step towards dynamic flows subject to uncertain travel times by merging the classical dynamic flow theory with the Γ-robustness framework introduced by Bertsimas and Sim [4] which provides a reasonable tradeoff between a solutions robustness and the inherited pessimism. Again, we refer to their work [4] for a comprehensive overview and notations.

We introduce additional delays for edges $\Delta : E \to \mathbb{Z}_+$ and assume that each edge can either assume its nominal travel time τ_e or a delayed travel time $\tau_e + \Delta_e$. Since we are interested in a merger of FLOWS OVER TIME with the Bertsimas and Sim framework, from now on denoted as Γ-ROBUST FLOW OVER TIME, we assume that at most $\Gamma \in \mathbb{Z}_+$ arcs may assume an increased travel

[*]Lehrstuhl für Management Science, RWTH Aachen University, D-52056 Aachen, {gottschalk,peis,schmand, wierz}@oms.rwth-aachen.de

[†]Lehrstuhl II für Mathematik, RWTH Aachen University, D-52056 Aachen, koster@math2.rwth-aachen.de

[‡]Lehrstuhl für Wirtschaftsmathematik, Universität Erlangen-Nürnberg, D-91058 Erlangen, frauke.liers@math.uni-erlangen.de

time simultaneously. That is, we consider a set of scenarios $\mathcal{S} \coloneqq \{z \in \{0,1\}^{|E|} : \sum_{e \in E} z_e \leq \Gamma\}$ each of which describes an augmented maximum flow over time instance with travel times $\tau_e^z \coloneqq \tau_e + \Delta_e z_e$, for $z \in \mathcal{S}$. We seek to find a flow f that is feasible in every scenario, that is, flow conservation constraints and capacity limits must not be violated. Among all such flows, we seek to maximize the worst-case flow value reaching the destination d.

2 Related Work

Skutella [7] presents a well-written introduction to dynamic flow problems in many variants including the special case of MAXIMUM FLOWS OVER TIME addressed in this paper. In the nominal case, the efficient construction of optimal solutions for the problem is known for more than half a century and was discussed by Ford and Fulkerson [6]. For a broad survey on dynamic flow problems, we refer to Aronson [2]. The notion of Γ-Robustness was introduced by Bertsimas and Sim [5] as a new approach towards uncertainty models with a reasonable degree of pessimism and since then has been used in a wide range of optimization problems including static flow problems. The static counterpart is shown to be NP-complete for $\Gamma \geq 2$ and polynomially solvable for $\Gamma = 1$ as shown by Aneja et al. [1]. Ben-Tal et al. [3] present a comprehensive introduction to related uncertainty models which have also been studied.

3 Our Contribution

To the best of our knowledge, the maximum flow over time problem in a Γ-robust context has not yet been studied in the literature. First, we address several issues that arise upon modelling the Γ-ROBUST MAXIMUM FLOW OVER TIME problem by means of the classical flow- and Γ-robustness theory. We argue that the classical encoding of solutions on an edge-level is no longer useful as it inherently introduces a high degree of pessimism in combination with flow conservation constraints. Imagine flow on a single path and consider any intermediate node: Flow particles may only be relayed if its existence is certain, hence, we need to assume that the worst-case travel time has appeared on the subpath up to this point. Since this holds for every intermediate node, the pessimism propagates towards the sink. Thus, we introduce a much more viable type of solution encoding in terms of a path decomposition with associated flow rates and dispatch-intervals which - in the nominal case - is as powerful as the classical encoding. In addition to the new model, we also argue that the classical notion of time-expanded networks is no longer usable in order to move from a dynamic flow problem towards their static counterparts of pseudopolynomial size. Due to uncertain travel times, it is not clear which edges are present, thus, the classical approach inherits a network design component. Finally, temporarily repeated flows are shown to no longer yield optimal solutions to the Γ-ROBUST MAXIMUM FLOW OVER TIME problem in contrast to the nominal counterpart.

Following, we discuss several complexity results for the newly introduced model. It follows immediately from the static counterpart that the optimization variant of Γ-ROBUST MAXIMUM FLOW OVER TIME is NP-hard for $\Gamma \geq 2$. This also holds for the class of temporarily repeated flows which, hence, are intractable for $\Gamma \geq 2$. We can show that the verification of feasibility of arbitrary solution candidates is an NP-complete problem by a reduction from the k-clique problem. This reduction, however, does not include temporarily repeated flows.

Still, temporarily repeated flows might be a good candidate for an approximate approach, so we study their computational complexity. We discuss how to compute an optimal temporarily

repeated flow for $\Gamma = 1$ and for a special class of instances with sufficiently large time horizon. That is, if we assume that the maximal worst-case path length of any simple path is bounded by the time horizon: $\max_{P \text{ simple path}, z \in \mathcal{S}} \sum_{e \in P} (\tau_e + \Delta_e z_e) \leq T$. The former can be computed in pseudopolynomial time, the latter in strongly polynomial time even if Γ is part of the input. Finally, temporarily repeated solutions are shown to have an optimality gap of at least 1.7 for $\Gamma = 1$ and a gap of at least $\Omega(\log \Gamma)$.

References

[1] Aneja, Y.P., Chandrasekaran, R., Nair, K.: Maximizing residual flow under an arc destruction. Networks 38(4), 194–198 (2001)

[2] Aronson, J.E.: A survey of dynamic network flows. Annals of Operations Research 20(1), 1–66 (1989)

[3] Ben-Tal, A., El Ghaoui, L., Nemirovski, A.: Robust optimization. Princeton University Press (2009)

[4] Bertsimas, D., Sim, M.: Robust Discrete Optimization and Network Flows. In: Math. Prog. vol. 98, pp. 49 – 71 (2003)

[5] Bertsimas, D., Nasrabadi, E., Stiller, S.: Robust and adaptive network flows. Operations Research 61(5), 1218–1242 (2013)

[6] Ford Jr, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. Operations research 6(3), 419–433 (1958)

[7] Skutella, M.: An introduction to network flows over time. In: Research Trends in Combinatorial Optimization, pp. 451–482. Springer (2009)

The Global EDF Scheduling of Systems of Conditional Sporadic DAG Tasks

Sanjoy K. Baruah * Vincenzo Bonifaci (Speaker) †
Alberto Marchetti-Spaccamela ‡

1 Introduction

We are concerned with scheduling algorithms and schedulability tests for task models that can encode parallel and/or conditional execution. The *sporadic DAG task model* [2] was introduced to permit the representation of parallelism that may be present within individual recurrent tasks. A task τ_i in this model is specified as a 3-tuple (G_i, D_i, T_i), where G_i is a directed acyclic graph (DAG), and D_i and T_i are positive integers representing the relative deadline and period parameters of τ_i respectively. The task τ_i repeatedly releases *dag-jobs*, each of which is a collection of (sequential) jobs. Successive dag-jobs are released a duration of at least T_i time units apart. The DAG G_i is specified as $G_i = (V_i, E_i)$, where V_i is a set of vertices and E_i a set of directed edges between these vertices. Each $v \in V_i$ represents the execution of a sequential piece of code (such execution is called a "job"), and is characterized by a worst-case execution time (wcet). The edges represent dependencies between the jobs: if $(v_1, v_2) \in E_i$ then job v_1 must complete execution before job v_2 can begin execution. (Job v_1 is called a *predecessor* job of v_2, and job v_2 is called a *successor* job of v_1.) Jobs that are not predecessors or successors of each other, either directly or transitively, may execute simultaneously upon different processors. A release of a dag-job of τ_i at time-instant t means that all $|V_i|$ jobs $v \in V_i$ are released at time-instant t. If a dag-job is released at time-instant t then all $|V_i|$ jobs that were released at t must complete execution by time-instant $t + D_i$.

As stated above, the sporadic DAG tasks model assumes that each release of a dag-job of τ_i causes the release of jobs corresponding to each and every vertex in V_i. However, control structures (such as conditional — *if-then-else* —constructs) within the code that is being modeled by the task may mean that different activations of the task (i.e., different dag-jobs) cause different parts of the code to be executed. Assuming that jobs corresponding to all the vertices in V_i will execute during each such activation is pessimistic; there is a need to be able to model the fact that different dag-jobs of the same task may cause different collections of jobs to be executed.

*baruah@cs.unc.edu. Department of Computer Science, University of North Carolina at Chapel Hill, CB 3175, NC 27599-3175, USA.

†vincenzo.bonifaci@iasi.cnr.it. Istituto di Analisi dei Sistemi ed Informatica, Consiglio Nazionale delle Ricerche, via dei Taurini 19, 00185 Roma, Italy.

‡alberto@dis.uniroma1.it. Dipartimento di Ingegneria Informatica Automatica e Gestionale, Sapienza Università di Roma, via Ariosto 25, 00185 Roma, Italy.

There may in general be exponentially many different execution flows through a graph. Consider for example code structured like this:

```
if (C1) then {S11} else {S12}
if (C2) then {S21} else {S22}
if (C3) then {S31} else {S32}
. . . . . .
. . . . . .
if (Cn) then {Sn1} else {Sn2}
```

where each `(Ci)` represents a boolean condition, and each `{Sij}` a block of straight-line code. It is evident that such a code fragment may have 2^n different execution flows through it; hence, requiring explicit enumeration of all execution flows and having the number of such flows be a determinant in the computational complexity of scheduling and schedulability analysis algorithms means that these algorithms all have exponential worst-case run-time.

2 Our Results

We propose the *conditional sporadic DAG task* model as an extension to the sporadic DAG task model [2] that is capable of modeling certain conditional control-flow constructs (including the cascade of `if-then-else` commands depicted above). We consider the Global Earliest-Deadline First (GEDF) scheduling of task systems that are modeled as collections of conditional sporadic DAG tasks; this is in contrast to the approach of [4], which develops entirely new (and rather complicated) server-based mechanisms for the run-time scheduling of systems of multi-DAG tasks. We quantitatively evaluate the effectiveness of GEDF as a scheduling mechanism for systems of conditional sporadic DAG tasks via the speedup factor metric [1, 5]. We show that the tight speedup bound of $(2 - 1/m)$ that was obtained in [3] for the GEDF scheduling of traditional (non-conditional) sporadic DAG task systems is easily shown to hold for systems of conditional sporadic DAG tasks as well. This means that at least from the perspective of speedup factor, the added expressive capabilities of the conditional sporadic DAG tasks model comes at no additional cost.

Theorem 1 *GEDF has a speedup factor of* $(2 - 1/m)$ *when scheduling systems of conditional sporadic DAG tasks upon* m *preemptive processors.*

Deriving an effective GEDF schedulability test for conditional sporadic DAG task systems turns out to be more challenging than showing the speedup bound – straightforward extensions of the techniques from [3] require the enumeration of all paths through the DAG-representation of the task (as in the approach of [4]), and result in exponential-time algorithms. We develop a novel transformation strategy that converts each conditional sporadic DAG task to a non-conditional one in polynomial time, and tests the system of transformed tasks for GEDF schedulability using the test provided in [3]. We show that the resulting GEDF schedulability test for conditional sporadic has a speedup factor equal to $(2 - 1/m + \epsilon)$ for any constant $\epsilon > 0$; once again, this is the same result as was available for traditional (i.e., not conditional) sporadic DAG task systems.

Theorem 2 *Let* $\epsilon > 0$. *There is an algorithm that, on input a set of conditional DAG tasks* τ*, returns a YES/NO output such that:*

- *If the output is YES, then τ is GEDF-schedulable on a platform of m processors of speed $2 - 1/m + \epsilon$.*
- *If the output is NO, then τ is not feasible on a platform of m processors of unit speed.*

The running time of the algorithm is pseudopolynomial in the size of τ and linear in $1/\epsilon$.

References

[1] B. Kalyanasundaram, K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM* 47(4): 617–643, 2000.

[2] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS 2012, pages 63–72, San Juan, Puerto Rico, 2012.

[3] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility analysis in the sporadic DAG task model. In *Proceedings of the Euromicro Conference on Real-Time Systems*, ECRTS 2013, pages 225–233, Paris (France), 2013.

[4] J. Fonseca, V. Nelis, G. Raravi, and L. M. Pinho. A Multi-DAG model for real-time parallel applications with conditional execution. In *Proceedings of the ACM/SIGAPP Symposium on Applied Computing*, Salamanca, Spain, April 2015. To appear.

[5] C.A. Phillips, C. Stein, E. Torng, J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica* 32(2): 163–200, 2002.

Scheduling with Time-Varying Reservation Costs*

Lin Chen † Nicole Megow ‡ Roman Rischke § Leen Stougie ¶
José Verschae ‖

February 12, 2015

1 Introduction

We consider a natural generalization of classical scheduling problems in which occupying a time slot incurs certain cost that may vary over time and which must be paid in addition to the actual scheduling cost. Such a framework has been proposed recently in [1] and [2]. It models, e.g., the opportunity cost associated with processing that may vary significantly over time. For instance, think of labor cost that may vary by the day of the week, or the hour of the day [1]. Similarly, users of cloud computing services, e.g. Amazon EC2, are offered pricing schemes that vary over time and it is a challenging task to optimize the tradeoff between resource provisioning cost and the quality of schedules. Another motivation, mentioned in [2], are the fluctuating electricity cost. On the one hand, they have economically a huge impact for facilities with enormous power consumption such as large data centers, and on the other hand, these fluctuations reflect the imbalance in power-consumption. Hence, cost aware scheduling is economically profitable and supports an eco-aware usage and in the end generation of energy.

2 Problem Definition

We first describe the underlying classical scheduling problems. We are given a set of jobs $J := \{1, \dots, n\}$ where every job $j \in J$ has a given processing time $p_j \in N$ and possibly a weight $w_j \in Q_{\geq 0}$. The task is to find a preemptive schedule on a single machine such that the total (weighted) completion time, $\sum_{j \in J} w_j C_j$, is minimized. Here C_j denotes the completion time of job j. In standard scheduling notation, this problem is denoted as $1 \,|\, pmtn \,|\, \sum(w_j)C_j$. We also consider the makespan minimization problem on unrelated machines, typically denoted as $R \,|\, pmtn \,|\, C_{\max}$. Here we are given a set

*Supported by the German Science Foundation (DFG) under contract ME 3825/1., by the EU-IRSIS EUSACOU grant, and by the Tinbergen Institute

†lchen@math.tu-berlin.de Department of Mathematics, Technische Universität Berlin, Germany.

‡nmegow@math.tu-berlin.de Department of Mathematics, Technische Universität Berlin, Germany.

§rischke@math.tu-berlin.de Department of Mathematics, Technische Universität Berlin, Germany.

¶stougie@cwi.nl Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam & CWI, The Netherlands.

‖jverschae@uc.cl Departamento de Matemticas & Departamento de Ingeniera Industrial y de Sistemas, Pontificia Universidad Catlica de Chile, Chile.

of machines M, and each job $j \in J$ has an individual processing time $p_{ij} \in N$ for running on machine $i \in M$. The task is to find a preemptive schedule that minimizes the makespan, that is, the completion time of the latest job.

We consider a generalization of these scheduling problems within a time-varying reservation cost model. We are given a piecewise constant cost function $e : N \to R$, where $e(t)$ denotes the reservation cost for processing job(s) at time t. More formally, we assume that time is discretized into unit-size time slots, and the time horizon is partitioned into given intervals $I_i = [s_i, d_i)$ with $s_i, d_i \in N$, $i = 1, \ldots, K$, within which unit-size time slots have the same *unit reservation cost* e_i. To ensure feasibility, let $d_K \geq \sum_{j \in J} \min_{i \in M} p_{ij}$.

Given a schedule $\mathcal{S}$, let $y(t)$ be a binary variable indicating if any processing is assigned to time t. The *reservation cost* for $\mathcal{S}$ is $E(\mathcal{S}) = \sum_t e(t)y(t)$. That means, for any time unit that is used in $\mathcal{S}$ we pay the full unit reservation cost, even if the unit is only partially used. We also emphasize that, in our model, in case of multiple machines, a reserved time slot can be used by all machines.

The overall objective that we consider is to find a schedule that minimizes the scheduling objective, $C_{\max}$ resp. $\sum_{j \in J} w_j C_j$, plus the reservation cost E. We refer to the resulting problems as $R \,|\, pmtn \,|\, C_{\max} + E$ and $1 \,|\, pmtn \,|\, \sum w_j C_j + E$.

3 Our Contribution

We present best possible approximation results for a generalization of standard scheduling problems to a framework with time-varying reservation cost.

Theorem 1 *The scheduling problem $R \,|\, pmtn \,|\, C_{\max}$ with time-varying reservation cost can be solved in polynomial time.*

The algorithm relies on an optimal algorithm for the problem without reservation cost [3] to determine the optimal number of time slots to be reserved, together with a procedure for choosing the time slots to be reserved.

Our main results concern single-machine scheduling to minimize the total (weighted) completion time.

Theorem 2 *The problem $1 \,|\, pmtn \,|\, \sum C_j + E$ can be solved in polynomial time.*

The algorithm is based on a dynamic programming formulation of the problem. While pseudo-polynomial time optimal algorithms are rather easy to derive, it is quite remarkable that the running time of our DP is polynomially bounded by the input size, in particular, independent of d_K. Our algorithm relies on a subtle analysis of the structure of optimal solutions and a properly chosen potential function.

Finally, we consider the strongly NP-hard problem variant where each job has its individual weight.

Theorem 3 *Let the maximum length of intervals with the same cost be bounded. For any fixed $\varepsilon > 0$, there is a polynomial-time algorithm that computes a $(1+\varepsilon)$-approximation for the problem $1 \,|\, pmtn \,|\, \sum_j w_j C_j + E(j)$. For arbitrary interval length, the algorithm runs in pseudo-polynomial time.*

This result improves on an earlier (pseudo-)polynomial $(4+\epsilon)$-approximation [2]. And in terms of approximation, our algorithm is best possible since the problem is strongly NP-hard even if there are only two different reservation costs [4].

Our approach is heavily inspired by a recent PTAS for scheduling on a machine of varying speed [5] and it uses some of its properties. As discussed above, there is no formal mathematical relation known between these two seemingly related problems which allows to directly apply the result from [5]. The key is a dual view on scheduling: instead of directly constructing a schedule in the time-dimension, we first construct a dual scheduling solution in the weight-dimension which has a one-to-one correspondence to a true schedule. We design an exponential time dynamic programming algorithm which can be trimmed to polynomial time using techniques known for scheduling with varying speed [5].

For both problems, the makespan and the min-sum problem, job preemption is crucial for obtaining worst case bounds. For non-preemptive scheduling, a straightforward reduction from 2-Partition shows that no approximation within a polynomial ratio is possible, unless P=NP, even if there are only two different reservation costs.

References

[1] G. Wan and X. Qi. Scheduling with variable time slot costs. Naval Research Logistics, 57:159171, 2010.

[2] J. Kulkarni and K. Munagala. Algorithms for cost-aware scheduling. In Proc. of WAOA 2012, pages 201214. Springer, 2012.

[3] E. L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. J. ACM, 25(4):612619, 1978.

[4] G. Wang, H. Sun, and C. Chu. Preemptive scheduling with availability constraints to minimize total weighted completion times. Ann. Oper. Res., 133:183192, 2005.

[5] N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In Proc. of ICALP 2013, pages 745756. Springer, 2013.

A Cycle Breaking approach for the Axial 3-Dimensional Assignment Problem with perimeter costs

A.M.C. Ficker (Speaker) * L.G. Afraimovich † F.C.R. Spieksma ‡

1 Introduction

In the Axial 3 Dimensional Assignment Problem ($A3DA$) we are given three disjoint $n-$sets of points (Red (R), Blue (B), Green (G)), and non-negative costs $c_{r,b,g}$ for each possible triple containing one point from each set. The goal is to select a set of triples with minimum total cost such that each point is covered. We consider a special case of $A3DA$ in which we are given a distance for every pair of distinctly coloured points and the cost of a triple is the sum of the three corresponding distances, i.e. $c_{r,b,g} = d_{r,b} + d_{b,g} + d_{r,g}$. The resulting problem is denoted by $A3DA$ with perimeter costs.

Related literature

The A3DA and its special cases have been studied in the literature; we refer to [8] for an overview. Also, the special case of perimeter costs has been studied; let us now describe the current state of affairs when it comes to complexity and approximability of this problem. In case the costs $d_{r,b}$, $d_{b,g}$ and $d_{r,g}$ do not necessarily satisfy the triangle-inequality, the problem is NP-hard, and even the existence of a constant-factor approximation algorithm would imply P = NP [3]. In case the costs do satisfy the triangle-inequality, a 4/3-approximation algorithm exists [1, 3], and the existence of a PTAS would imply P=NP [4]. The special case where the points lie in the plane remains NP-hard [7].

Practical relevance

The following application illustrates the relevance of our problem; for an extensive description see [5]. Consider an experiment where the effect of three distinct treatments needs to be assessed. The idea is to test the treatments on a set of n individuals, however, since the individuals differ (for instance with respect to age, gender, and other characteristics), it becomes relevant to make a sensible choice with respect to the subset of individuals that receive treatment A, B or C. In [6] it is explained how, for each individual i a probability $P_{i,A}$ ($P_{i,B}$, $P_{i,C}$) is computed that represents the probability that individual i should receive treatment A (B, C). Next, the idea is to use the

*annette.ficker@kuleuven.be. Faculty of Economics and Business, KU Leuven, Naamsestraat 69, Leuven, Belgium.

†levafraimovich@gmail.com. Faculty of Computational Mathematics and Cybernetics, Nizhny Novgorod State University, Gagarin Avenue, Nizhny Novgorod, Russia.

‡frits.spieksma@kuleuven.be. Faculty of Economics and Business, KU Leuven, Naamsestraat 69, Leuven, Belgium.

3-dimensional vector $(P_{i,A}, P_{i,B}, P_{i,C})$ as the vector describing individual i, and to compute triples of individuals that are, in some sense, close to each other (the idea being that the three individuals of a triple receive a distinct treatment). In [9] it is discussed explicitly what it means to be close, and they propose as a measure for closeness, the perimeter costs. Notice that this essentially corresponds to a special case where the points lie on a plane (and hence satisfy the triangle-inequality).

2 Our results

We assume that all edge costs satisfy the triangle-inequality. We present two results.

First, we propose an alternative 4/3-approximation algorithm called Cycle Breaking. We see this algorithm as a procedure not only computing a feasible solution; in fact, we see it as a procedure that simultaneously computes a lowerbound, V_{CB}^{rel}, and an upperbound, V_{CB}. Essentially, the algorithm computes an interval for the value of the optimal solution, OPT, and we will show that the length of this interval is bounded by $OPT/4$.

Second, we consider a natural ILP formulation for $A3DA$ and show that the value of its LP-relaxation, denoted by V_{LP}, lies within this interval. We point out that this LP-relaxation has been studied for a maximization objective in [2]. They show that, for general costs c_{rbg}, the integrality gap equals 2.

Our results can be summarized by the following series of inequalities,

$$V_{CB}^{rel} \leq V_{LP} \leq OPT \leq V_{CB} \leq 4/3 V_{CB}^{rel} \leq 4/3 \cdot OPT.$$

References

[1] Bandelt, H. J., Y. Crama and F.C.R. Spieksma (1994), Approximation algorithms for multidimensional assignment problems with decomposable costs, *Discrete Applied Mathematics* 49, 25-50.

[2] Chan Y.H., L.C. Lau (2012), On linear and semidefinite programming relaxations for hypergraph matching, *Mathematical Programming* 135, 123-148.

[3] Crama, Y. and F.C.R. Spieksma (1992), Approximation algorithms for three-dimensional assignment problems with triangle inequalities, *European Journal of Operational Research* 60, 273-279.

[4] Goossens, D., S. Polyakovskiy, F.C.R. Spieksma, and G.J. Woeginger (2012), Between a rock and a hard place: The two-to-one assignment problem, *Mathematical Methods of Operations Research* 76, 223-237.

[5] Higgins, M. (2013), Applications of integer programming methods to solve statistical problems, *PhD thesis, UC Berkeley.*

[6] Rassen, J.A., A.A. Shelat, J.M. Franklin, R.J. Glynn, D.H. Solomon, S. Schneeweiss (2013), Matching by propensity score in cohort studies with three treatment groups, *Epidemiology* 24, 401-409.

[7] Spieksma, F.C.R. and G.J. Woeginger (1996), Geometric three-dimensional assignment problems, *European Journal of Operational Research* 91, 611-618.

[8] Spieksma, F.C.R. (2000), Multi Index Assignment Problems: Complexity, Approximation, Applications, in: Nonlinear Assignment Problems, *edited by P. Pardalos and L. Pitsoulis, Kluwer,* 1-12.

[9] Xu, Z. and J. Kalbfleisch (2013), Repeated randomization and matching in Multi-Arm trials, *Biometrics* 69, 949-959.

Star scheduling

Nadia Brauner Hadrien Cambazard Benoit Cance Nicolas Catusse
Pierre Lemaire * Anne-Marie Lagrange Pascal Rubini †

For astrophysicists, the best telescopes are scarce and expensive resources. Therefore, within the available time, they want to maximize the scientific gain of observations that are to be scheduled on the telescope. Those observations are choosen from a list of candidate stars together with their observation constraints. This schedule should be rapidly updated in case of climatic or technical variations.

1 Problem definition

We consider a list of nights $\mathcal{N}$ when the telescope is available for the concerned team of astrophysicists, and a list $\mathcal{S}$ of stars the researchers want to observe. Physical parameters on each star allow to calculate the interval $[r_s^n, d_s^n)$ when the star s can be observed in night n, and the duration p_s^n of this observation; note that, due to some technical reasons, the duration is at least the half of the observation window ($2p_s^n \geq d_s^n - r_s^n$). Each star s also has a weight w_s that represents the interest of the star for the researchers. The objective is to maximize the total weight of the observed stars by deciding the allocation of the stars to the nights and the schedule of the observations within the nights.

Astrophysicists already have a software that implements a constructive heuristic which rapidly proposes feasible solutions. It serves as a baseline for the numerical experiments.

2 Complexity and MIP formulation

If the observation intervals do not depend on the night, then nights can be viewed as identical machines, and the star scheduling problem becomes a special case of the parallel machine scheduling problem $P|r_j|\sum_j w_j U_j$, which is well-known to be NP-hard. However, the special case when $p_s = d_s - r_s$ is an interval scheduling with given machines, which can be solved in polynomiam time [2].

In our case, the assumption $2p_s^n \geq d_s^n - r_s^n$ implies a particular structure of a feasible solution: if scheduled, a star is under observation at the meridian instant ($\frac{r_s^n + d_s^n}{2}$) and, as a consequence, observations must be ordered by non-decreasing meridian instant. Nevertheless, as we prove, the star scheduling problem is NP-complete, even when considering only one night, and unary NP-hard when there are several nights.

*{nadia.brauner,hadrien.cambazard,nicolas.catusse}@g-scop.grenoble-inp.fr. Grenoble Alpes, CNRS, G-SCOP, F-38000 Grenoble

†{anne-marie.lagrange@obs.ujf-grenoble.fr, pascal.rubini@free.fr}. CNRS, IPAG, F-38000 Grenoble, France

A first formulation as an integer linear program allows us to rapidly propose solutions for small instances and to validate the data, the constraints and the solutions. The variables of this linear program are:

- $z_s = 1$ if star s is scheduled, 0 otherwise;
- $t_s \geq 0$ is the starting time of the observation of star s;
- $z_s^n = 1$ if star s is scheduled within night n, 0 otherwise;
- $z_{ss'} = 1$ if star s is scheduled before s' within the same night, 0 otherwise.

The linear program is as follows:

$$\begin{array}{rcll} \max \sum_{s\in\mathcal{S}} w_s z_s & & & \\ \sum_n z_s^n & = & z_s & \forall s \in \mathcal{S} \\ r_s^n z_s^n & \leq & t_s & \forall s \in \mathcal{S} \text{ et } n \in \mathcal{N} \\ t_s + p_s^n z_s^n & \leq & d_s^n z_s^n + M(1 - z_s^n) & \forall s \in \mathcal{S} \text{ et } n \in \mathcal{N} \\ z_s^n + z_{s'}^n - 1 & \leq & z_{ss'} + z_{s's} & \forall s, s' \in \mathcal{S} \text{ et } n \in \mathcal{N} \\ z_{ss'} + z_{s's} & \leq & 1 & \forall s \in \mathcal{S} \\ t_s + p_s^n & \leq & t_{s'} + M(1 - z_{ss'}) & \forall s, s' \in \mathcal{S} \text{ et } n \in \mathcal{N} \end{array}$$

where M is the length of the longest night.

3 Solution procedures

We propose three different solution procedures to solve the star scheduling problem.

3.1 A Benders-type decomposition

The preceding linear program does not lead to solutions for realistic instances. However the problem allows a natural logic-based Benders type of decomposition [1]: the master program copes with the allocation of the observations to the nights whereas the slave program (sub-problem) derives the scheduling of each night or returns to the master a new constraint for incompatible observations in order to reconsider the allocation.

The master provides an upper bound at each iteration and the first assignment of observations to nights that turns into a feasible schedule is thus an optimal solution.

3.2 A column-generation algorithm

A different approach is based on the fact that a dynamic programming algorithm can be designed for the single night case. Let S be a set of m observations that can be performed a given night of length T; an ordering of the observation is known : $s_1 \leq \ldots \leq s_m$. We want to maximize the sum of weights of the observations scheduled in the night.

We denote by $f^*(i,t)$ the optimal value of a schedule involving observations $s_1, \ldots s_i$ and such that s_i finishes before t. Considering $f^*(0,t) = 0$ for all $t \geq 0$, we can write:

$$f^*(i,t) = \begin{cases} \min(f^*(i-1,t), f^*(i-1,t-p_i) + w_i) & \forall i \in [1,m], t \in [r_i + p_i, T] \\ f^*(i-1,t) & \forall i \in [1,m], t \in [0, r_i + p_i[\\ -\infty & \forall i \in [1,m], t < 0 \\ 0 & i = 0, \forall t \in [0,T] \end{cases}$$

We are looking for $f^*(m,T)$ which can be computed in $O(mT)$.

Because solving the single-night case is not that hard, this allows for a classical column generation procedure: the set of all possible schedules for every night is considered and a MIP selects an optimal subset of them. This is done iteratively by adding the column (feasible schedule) with maximum reduced cost, which can be computed using the dynamic program for the single-night case.

3.3 A local-search procedure

A local-search procedure has also been implemented. Several moves are considered (e.g, moving an observation from one night to the other; inserting an observation; swapping two observations). The moves modify the set of possible observations for each night, and the actual assignment is optimaly computed using the dynamic program.

4 Experimentations

Implementation and experimentations are still preliminary. The computations were performed on real, but limited, data: nine instances from 200 to 800 observations and 32 to 142 nights.

On one hand, the column generation (which only provides an upper bound) dominates the Benders decomposition on almost all instances. On the other hand, the local-search procedure achieves very good feasible solutions. The average gap is 0.04%, with a maximum of 0.2%.

5 Perspectives

More detailed experiments are to be carried out in order to better qualify the actual performance of the various solution methods. Several implementation improvements could also be considered, e.g., handling symetries and the similarity of different nights.

From a theoretical point-of-view, a more detailed analysis of the complexity of the star scheduling problem would be of interest (e.g., polynomial cases and approximability issues).

References

[1] John N. Hooker Planning and Scheduling by Logic-Based Benders Decomposition. In *Operations Research* 55(3): 588–602, 2007.

[2] Antoon W.J. Kolen, Jan Karel Lenstra, Christos H. Papadimitriou, and Frits C.R. Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

Scheduling over scenarios on two machines

Esteban Feuerstein [*] Alberto Marchetti-Spaccamela [†] Frans Schalekamp [‡]
René Sitters [§¶] Suzanne van der Ster (Speaker) [§] Leen Stougie [§¶]
Anke van Zuylen [‡]

1 Introduction

A scheduling problem over *scenarios* is considered, where the goal is to find *a priori* one solution that performs well for each scenario in a predefined set. In particular, we are given a set J of jobs, where job j has processing time p_j, and a set of k scenarios $\mathcal{S} = \{S_1, S_2 \ldots, S_k\}$, where each scenario is specified by a subset of jobs in J that must be executed in that scenario. We will restrict ourselves to the case of two machines. Our goal is to find an assignment of jobs to machines that is the same *for all scenarios* and optimizes a function of the makespan *over all scenarios*. The two objectives considered are minimizing the *maximum* makespan over all scenarios and minimizing the *sum* of the makespans of all scenarios. In the remainder, we will denote the problem of minimizing the maximum makespan over scenarios on two machines by MinMax2 and the problem of minimizing the sum of makespans over scenarios on two machines by MinSum2. The results presented here have been publisher earlier [4].

As an example, suppose that J contains three jobs, numbered 1, 2 and 3, that must be executed on two machines. The processing time of job 1 is 2, while the processing times of job 2 and 3 are 1. The three scenarios are $S_1 = \{1, 2, 3\}$ and $S_2 = S_3 = \{2, 3\}$. Assigning job 1 to the first machine and jobs 2 and 3 to the second machine minimizes the maximum makespan over all scenarios, while assigning jobs 1 and 2 to the first machine and job 3 to the second machine minimizes the sum of the makespans.

Both the MinMax2 and the MinSum2 problem are NP-hard, since the single-scenario version (the well-known Makespan Minimization problem) is NP-hard. However, the single-scenario version is only weakly NP-hard for two machines and an FPTAS exists [1], whereas the problems defined here are strongly NP-hard. We will give approximability and inapproximability results for several special cases.

Kasperski et al. [6] have studied the same setting, but for slightly different objective functions.

[*] efeuerst@dc.uba.ar Universidad de Buenos Aires, Argentina
[†] alberto@dis.uniroma1.it Sapienza University of Rome, Italy
[‡] {frans, anke}@wm.edu College of William and Mary, Williamsburg VA, USA
[§] {r.a.sitters, suzanne.vander.ster, l.stougie}@vu.nl Vrije Universiteit, Amsterdam
[¶] {r.a.sitters, stougie}@cwi.nl CWI, Amsterdam, The Netherlands

2 Minimizing the maximum makespan

Using a recent result [2] on the hardness of Hypergraph Balancing (given a hypergraph, find a 2-coloring of the vertices such as to minimize over all hyperedges the discrepancy between the number of vertices of the two colors), we obtain an inapproximability result for the MinMax2 problem.

Theorem 1 *It is NP-hard to approximate the MinMax2 problem to within ratio* $2 - \epsilon$*, even if all jobs have unit length.*

Note that this is a remarkable result, since, for two machines, any solution (irrespective of job lengths) is 2-approximate. Tighter results are found for two special cases: the case where each scenario consists of at most three jobs (inapproximable to within a factor 3/2) and the case where each scenario contains exactly two jobs (solvable in $k \log k$ time).

However, the inapproximability results only hold when k, the number of scenarios, is arbitrary. When k is a constant and jobs have unit length, the problem can be solved to optimality in polynomial time by guessing the correct assignment of the jobs to the machines. This also applies for a constant k and unit-length jobs in the MinSum2 problem that we consider in the next section.

Considering an arbitrary number of machines, the MinMax problem reduces to the Vector Scheduling problem (see [3] for a definition), where each coordinate corresponds to one scenario. This reduction gives us the following results.

Theorem 2 *For the problem of minimizing the maximum makespan over k scenarios on an arbitrary number of machines,*

1. *there exists a PTAS if k is a constant,*
2. *there exists a polynomial-time $O(\log^2 k)$-approximation for arbitrary k,*
3. *for any $c > 1$, there exists no c-approximation for arbitrary k.*

3 Minimizing the sum of makespans

The problem of minimizing the sum of makespans on two machines where all jobs have unit length and each scenario contains two jobs only, can be reduced from Max Cut. This reduction, combined with inapproximability results for Max Cut [5, 7], give the results in the following theorem.

Theorem 3 *The problem MinSum2 is NP-hard to approximate to within a factor* 1.0196 *and UGC-hard to approximate to within a factor of* 1.0404*, even if all jobs have unit length and each scenario contains two jobs.*

For an arbitrary number of jobs per scenario, we present a randomized approximation algorithm. The following lemma is needed.

Lemma 4 *Consider a scenario S, and let $A, \bar{A}$ be* any *partitioning of the jobs in S. When assigning each job of S to the two machines independently with equal probability, the expected load of the least loaded machine for this scenario is at least $\frac{1}{2}\min\{p(A), p(\bar{A})\}$, where $p(A)$ is the sum of processing times of job set A.*

Consider a scenario S and let B and $\bar{B} = S \backslash B$ be the sets of jobs processed on the first and second machine, respectively, in a schedule of minimum makespan. Hence, the optimal makespan for scenario S is $\max\{p(B), p(\bar{B})\}$. By Lemma 4, when assigning the jobs randomly with equal probability to each machine, the load on the least loaded machine is at least $\frac{1}{2}\min\{p(B), p(\bar{B})\}$. Hence, the load of the machine with the highest load is at most $p(B) + p(\bar{B}) - \frac{1}{2}\min\{p(B), p(\bar{B})\} = \max\{p(B), p(\bar{B})\} + \frac{1}{2}\min\{p(B), p(\bar{B})\} \leq \frac{3}{2}\max\{p(B), p(\bar{B})\}$. Hence, for scenario S, the expected makespan is at most $\frac{3}{2}$ times the optimal makespan for scenario S, which implies that the expected sum of makespans over all scenarios is at most $\frac{3}{2}$ times the optimal sum of makespans.

Theorem 5 *Randomly assigning each job to the two machines independently with equal probability is a* 3/2*-approximation for MinSum2.*

For problem instances with scenario sizes up to 4, better approximations can be found by reductions to Max Cut and Weighted Max Not-All-Equal Satisfiability [8]. These reductions, and the best known approximation ratios for these two problems, give an approximation ratio of 1.12144 for instances with scenario sizes up to three and 3/2 when the scenarios have no more than four jobs.

References

[1] G. Ausiello, P. Crescenzi, V. Kann, G. Gambosi, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties.* Springer Verlag, 1999.

[2] P. Austrin, V. Guruswami, and J. Håstadt. $(2 + \epsilon)$-SAT is NP-hard. *Electronic Colloquium on Computational Complexity*, TR13–159, 2013.

[3] C. Chekuri and S. Khanna. On multidimensional packing problems. *SIAM Journal on Computing*, 33(4): 837–851, 2004.

[4] E. Feuerstein, A. Marchetti-Spaccamela, F. Schalekamp, R. Sitters, S. van der Ster, and A. van Zuylen. Scheduling over scenarios on two machines. In Z. Cai, A. Zelikovsky, and A.G. Bourgeois (Eds.), *Proceedings of 20th International Computing and Combinatorics Conference*, volume 8591 of *Lecture Notes in Computer Science*, pages 559–571. Springer, 2014.

[5] J. Håstadt. Some optimal inapproximability results. *Journal of the ACM*, 48(4): 798–859, 2001.

[6] A. Kasperski, A. Kurpisz, and P. Zielinski. Parallel machine scheduling under uncertainty. In *Proceedings of 14th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems* (IPMU), 2012.

[7] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1): 319–357, 2007.

[8] J. Zhang, Y. Ye, and Q. Han. Improved approximations for max set splitting and max NAE SAT. *Discrete Applied Mathematics*, 142(1–3): 133–149, 2004.

Unrelated Machine Scheduling with Stochastic Processing Times*

Martin Skutella † Maxim Sviridenko ‡ Marc Uetz §

1 Introduction

We consider the problem of minimizing the total weighted completion time on unrelated parallel machines, denoted $R|r_{ij}|\sum w_jC_j$. Each job j has a weight w_j, possibly an individual release date r_{ij} before which job j must not be scheduled on machine i, and the processing time of job j on machine i is p_{ij}. Each job has to be processed non preemptively on any one of the machines, and each machine can process at most one job at a time. The objective is to find a schedule minimizing the total weighted completion time $\sum_j w_jC_j$, where C_j denotes the completion time of job j in the schedule.

Hoogeveen et al. [1] prove MaxSNP-hardness and hence there is no polynomial time approximation scheme. On the positive side, the currently best known approximation algorithms for unrelated parallel machines have performance guarantees 3/2 and 2, for the problem without and with release dates, respectively [3]. Improving these bounds is considered to be among the most important open problems in scheduling which is also an indication of the high significance of unrelated machine scheduling.

Stochastic scheduling. We consider for the first time the stochastic variant of unrelated machine scheduling. Here, the processing time of a job j on machine i is given by random variable P_{ij}. In stochastic scheduling, we are asked to compute a non-anticipatory scheduling policy. Roughly spoken, a scheduling policy makes scheduling decisions at certain decision times t, and these decisions are based on the observed past up to time t as well as the a priori knowledge of the input data of the problem. The policy, however, must not anticipate information about the future, such as the actual realizations of the processing times of jobs which have not yet been completed by time t. Here we confine ourselves with an intuitive description that puts stochastic scheduling in the framework of stochastic dynamic optimization: Actions of a scheduling policy at a time t consists of a set of jobs, possibly empty, to be started on a set of idle machines, together with a tentative next decision time $t^* > t$. The next action of the policy is due at t^*, or the time of the next job completion, or the time when the next job is released, whatever occurs first. Depending on the action of the policy, the next decision

*extended abstract was published in Proceedings of STACS2014

†TU Berlin, Institut für Mathematik, MA 5-2, Strasse des 17. Juni 136, 10623 Berlin, Germany, martin.skutella@tu-berlin.de

‡Yahoo! Labs, 229 West 43rd Street, Floors 9, 10, 11, 12, New York, NY, 10036, US, sviri@yahoo-inc.com

§University of Twente, Dept. Applied Mathematics, P.O. Box 217, 7500 AE Enschede, The Netherlands, m.uetz@utwente.nl

stochastic scheduling model	worst case performance guarantee arbitrary P_{ij}	$CV[P_{ij}] \leq 1$	reference
$P\|\|E[\sum w_jC_j]$	$1+\frac{(m-1)(\Delta+1)}{2m}$	$2-1/m$	[2]
$P\|r_j\|E[\sum w_jC_j]$	$2+\Delta$	3	Schulz[2008]
$R\|\|E[\sum w_jC_j]$	$1+\frac{\Delta+1}{2}+\epsilon$	$2+\epsilon$	this paper
$R\|r_{ij}\|E[\sum w_jC_j]$	$2+\Delta+\epsilon$	$3+\epsilon$	this paper

Table 1: Performance bounds for nonpreemptive stochastic machine scheduling problems. Parameter $\epsilon > 0$ can be chosen arbitrarily small. Parameter Δ upper bounds the squared coefficient of variation $CV^2[P_{ij}] = Var[P_{ij}]/E^2[P_{ij}]$ for all P_{ij}. The third column shows the results for $CV[P_{ij}] \leq 1$; e.g., uniform, exponential, or Erlang distributions. As usual in stochastic scheduling, these bounds hold with respect to the expected performance of any non-anticipatory scheduling policy.

time as well as the state of the schedule at the next decision time is realized according to the probability distributions of the jobs' processing times. A non-anticipatory policy may learn over time, but it has only access to distributional information about remaining processing times of unfinished jobs, conditioned on the state of the schedule at time t. As all previous work in the area, we assume that the random variables P_{ij} are stochastically independent across jobs. For any given non-anticipatory scheduling policy, the possible outcome of the objective function $\sum_j w_jC_j$ is a random variable, and our goal is to minimize its expected value, which by linearity of expectation equals $\sum_j w_jE[C_j]$.

Related Work. Generalizing a well known result of Smith[1956] for deterministic single machine scheduling, Rothkopf[1966] proved that the WSEPT rule minimizes the expected total weighted completion time on a single machine. The first constant factor approximation algorithms for stochastic scheduling on identical parallel machines have been obtained in 1999 by Möhring et al. [2]. Next to a linear programming (LP) based analysis of the WSEPT rule, they define list scheduling policies which are based on linear programming relaxations in completion time variables. The performance bounds are constant whenever the coefficients of variation of the jobs' processing times are bounded by a constant. As usual in stochastic scheduling, all bounds hold with respect to any non-anticipatory scheduling policy. Note that all results obtained so far are restricted to identical parallel machines. Table 1 gives an overview of currently best known performance bounds in nonpreemptive stochastic scheduling with minsum objective, next to the results obtained in this paper.

Our contribution. We obtain the first approximation algorithms for stochastic scheduling on unrelated machines. Despite the fact that the unrelated machine scheduling model is significantly richer than identical machine scheduling, our bounds essentially match all previous performance bounds that have been obtained for the corresponding stochastic scheduling problems on identical parallel machines; see Table 1. We also give a tight lower bound, showing that the dependence of the performance bound on the squared coefficient of variation Δ is unavoidable for the class of policies that we use. For the first time we completely depart from the LP relaxation of Möhring et al. [2], and show how to put a novel, time-indexed linear programming relaxation to work in stochas-

tic machine scheduling. We are optimistic that this novel approach will inspire further research and prove useful for other stochastic optimization problems in scheduling and related areas.

Time-indexed linear programming relaxations have played a pivotal role in the development of constant factor approximation algorithms for deterministic scheduling on unrelated parallel machines. In spite of that, it remained unclear and a major open problem how to come up with a meaningful time-indexed LP relaxation for stochastic scheduling problems. Here the main difficulty is that, in contrast to deterministic schedules that can be fully described by time-indexed 0-1-variables, scheduling *policies* feature a considerably richer structure including complex dependencies between the execution of different jobs which cannot be easily described by time-indexed variables.

We show how to overcome this difficulty and present the first time-indexed LP relaxation for stochastic scheduling on unrelated parallel machines. Here, the value of the time-indexed variable x_{ijt} represents the probability of job j being started on machine i at time t. While writing down the machine capacity constraints is rather easy for deterministic scheduling in this formulation, the situation is somewhat more complicated in the stochastic setting and we require a fair amount of information about the exact probability distributions of random variables P_{ij}.

Notice that, due to the stochastic nature of processing times, even a schedule produced by an optimal policy can be arbitrarily long such that infinitely many variables x_{ijt} may take positive values. Nonetheless, in the full version of the paper we show how to overcome this difficulty. Indeed, we can compute an LP-solution in polynomial time that approximates the optimal LP solution with arbitrary precision.

We show how to turn a feasible solution to the time-indexed LP relaxation into a simple scheduling policy. Each job j is randomly assigned to a machine i with probability $\sum_t x_{ijt}$; then, on each machine i, the WSEPT policy is used to schedule the jobs assigned to i. The analysis, however, is based on a somewhat more elaborate, random sequencing of jobs which is determined by a two-stage random process.

Since each job is immediately and irrevocably assigned to a machine, our scheduling policies fall into the special class of *fixed assignment policies*. Notice that these policies ignore the additional information that evolves over time in the form of the actual realizations of processing times. Not surprisingly, this ignorance comes at a price. We prove a lower bound of $\Delta/2$ on the performance guarantee of *any* fixed assignment policy. Moreover, we also show that the LP relaxation can have an optimality gap in the same order of magnitude. These negative results nicely complement our positive results; see Table 1.

References

[1] Han Hoogeveen, Petra Schuurman, Gerhard J. Woeginger, Non-Approximability Results for Scheduling Problems with Minsum Criteria. INFORMS Journal on Computing 13(2): 157-168 (2001).

[2] Rolf H. Möhring, Andreas S. Schulz, Marc Uetz, Approximation in stochastic scheduling: the power of LP-based priority policies. J. ACM 46(6): 924-942 (1999).

[3] Martin Skutella, Convex quadratic and semidefinite programming relaxations in scheduling. J. ACM 48(2): 206-242 (2001).

Interval Selection on Unrelated Machines

Kateřina Böhmová (Speaker) *† Yann Disser ‡ Enrico Kravina
Matúš Mihalák Peter Widmayer

Introduction

We survey our ongoing work based on [2, 3, 4][1]. We study a *fixed-interval* scheduling problem with m machines and n jobs, called IntervalSelection *on unrelated machines*, where each job has on every machine an open interval of the reals (denoting the exact time interval when the job can be processed on the machine). By scheduling a job on a machine, one implicitly *selects* the corresponding interval of the job (and makes the machine unavailable for that time period). The goal is to schedule the maximum number of jobs such that no two selected intervals from the same machine intersect.

If $m = 1$, the problem becomes the classic interval scheduling problem which is solvable in $O(n \log n)$ time by a *single-machine greedy* algorithm: Scan iteratively the right endpoints of the intervals from left to right, and in each iteration *select* the considered interval, if and only if it does not intersect any of the previously selected intervals.

Unfortunately, we show that already for $m \geq 2$, the problem is NP-hard even if all the intervals have the same length. A straightforward generalization of the single-machine greedy, *multi-machine greedy*, has an approximation ratio of 1/2 (see [11]): Consider the machines one by one in an arbitrary order, run the single-machine greedy on the intervals of the currently considered machine, add all the selected intervals to the solution and remove the jobs that correspond to them from all the subsequent machines.

There are many variants of the broad class of interval scheduling, see, e.g., recent surveys by Kolen et al. [8] and by Kovalyov et al. [9]. In a classic variant (with identical machines), each job is identified with exactly one interval, and a job (i.e., the interval) can be scheduled on any of the machines. This problem is a special variant of IntervalSelection where the intervals of each particular job are the very same time interval (on all the machines). This problem is polynomially solvable even in the weighted case [1, 5] (and for any m).

On the other hand, IntervalSelection can be seen as a special case of JISP$_k$, the *job interval selection problem*, where each job has exactly k intervals on the real line (and the goal is to schedule a maximum number of jobs). Any instance of JISP$_k$ where the real line can be split into k parts (by $k-1$ vertical lines) so that every part contains

*This work was partially supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities and Sustainability), under grant agreement no. 288094 (project eCOMPASS). Kateřina Böhmová is a recipient of the Google Europe Fellowship in Optimization Algorithms, and this research is supported in part by this Google Fellowship.

†katerina.boehmova@inf.ethz.ch, matus.mihalak@inf.ethz.ch, enrico.kravina@inf.ethz.ch, widmayer@inf.ethz.ch. Department of Computer Science, ETH Zürich, Switzerland.

‡disser@math.tu-berlin.de. Department of Mathematics, TU Berlin, Berlin, Germany

[1]Many parts of this text are literal copies of existing manuscripts.

exactly one interval for each job, is also an instance of IntervalSelection, where each part represents one machine. JISP$_k$ for $k \geq 2$ was shown to be NP-hard [7, 10] and even APX-hard [11]. There is a *deterministic* 1/2-approximation algorithm for JISP$_k$ [11] which works similarly as the multi-machine greedy, and a *randomized* $\frac{e-1}{e}$-approximation algorithm [6]–the only algorithm that beats the barrier of 1/2 in a general setting. Thus, beating the approximation ratio of 1/2 in the *deterministic* case is a main open problem.

We show that IntervalSelection is NP-hard already for $m \geq 2$. For $m = 2$, we give a 2/3-approximation algorithm (see the details in the next section); and, moreover, we generalize it for any constant m, achieving approximation ratio $(\frac{1}{2} + \frac{1}{2m(m-1)})$.

Furthermore, we study a special case of IntervalSelection, where all the m intervals of every job have a point in common. In other words, the intersection of all the m intervals is non-empty. We call such a common point a *core* of the job. IntervalSelection with cores can be solved optimally in a running time exponential in m by a dynamic programming algorithm [12] (in polynomial time, whenever m is a constant). However, for a non-constant m, we give a reduction from 2-MaxSat and prove that the problem is APX-hard even when all intervals have the same length. Finally, we give a $\frac{501}{1000}$-approximation algorithm for IntervalSelection with cores and intervals of same length to show this problem can be (deterministically) approximated better than 1/2.

Approximation Algorithm for IntervalSelection

Due to the space constraints, we omit the details on most of the results mentioned above and present only the 2/3-approximation algorithm for IntervalSelection for the case $m = 2$. We choose this particular algorithm because of its simplicity and because it generalizes to other optimization problems.

The algorithm is based on the fact that IntervalSelection on a single machine can be solved optimally in polynomial time. It works as follows: Let M_1, M_2 be the two machines. First, let $\text{Res}_{12} = S_1 \cup S_{12}$ be the jobs scheduled by multi-machine greedy when considering the machines in the order $M_1 \to M_2$ (S_1 is scheduled on M_1, S_{12} on M_2). Next, let $\text{Res}_{21} = S_2 \cup S_{21}$ be the jobs scheduled by multi-machine greedy for the order $M_2 \to M_1$ (S_2 is scheduled on M_2, S_{21} on M_1). Finally, the algorithm returns the larger of the two sets Res_{12} and Res_{21}. (Note that all steps can be computed in polynomial time.) Surprisingly, this simple strategy gives a 2/3-approximation.

To analyze the performance, we relate the output of the algorithm in one of the directions, i.e., $\text{Res}_{12} = S_1 \cup S_{12}$ to an optimum solution $O = O_1 \cup O_2$, where O_1 and O_2, are the jobs scheduled on M_1 and M_2, respectively, with $O_1 \cap O_2 = \emptyset$. Clearly, S_1 and S_2 are maximum non-overlapping sets of intervals on M_1 and M_2, respectively. Thus, $|S_1| \geq |O_1|$ and $|S_2| \geq |O_2|$. After the algorithm picks S_1, M_2 still contains the non-overlapping intervals of the jobs $O_2 \setminus S_1$, and hence $|S_2| \geq |O_2| - |S_1 \cap O_2|$. The only reason that S_{12} contains less than $|O_2|$ jobs is that S_{12} cannot contain jobs already selected into S_1. But if $S_1 \cap O_2$ is large, then it means that M_1 contains two large (disjoint) non-overlapping sets of intervals $O_1 \setminus S_1$ and $S_1 \setminus O_1$, which makes it "easier" for the algorithm to pick a large non-overlapping set of intervals in M_1 without S_2 when going in the other direction $M_2 \to M_1$ (i.e., when computing S_{21}). Formally, let S_1^O be the set of jobs of S_1 that are also in the optimum O_1, and let S_1^R be the remaining jobs of S_1. Analogically, we subdivide S_2 into S_2^O and S_2^R. We can prove the following.

Lemma 1 *The size of S_{12} is at least $|O_2| - |S_1^R|$* **(a)**, *and at least $\frac{|O_2|+|S_2|-|S_2^O|-|S_1|}{2}$* **(b)**.

Theorem 2 *The algorithm is a 2/3-approximation for* IntervalSelection *with* $m = 2$.

Proof. Assume, wlog, that $|S_2^R| \geq |S_1^R|$. We distinguish two cases. First, assume that $|S_1^R| \leq 1/3 \cdot |O|$. By Lemma 1(a), $|S_{12}| \geq |O_2| - |S_1^R|$, thus the solution Res_{12} is of size $|S_1| + |S_{12}| \geq |O_1| + |O_2| - |S_1^R| \geq |O_1| + |O_2| - 1/3 \cdot |O| = 2/3 \cdot |O|$.

Now, assume that ($|S_2^R| \geq$) $|S_1^R| \geq 1/3 \cdot |O|$. By Lemma 1(b), $|S_{12}| \geq \frac{1}{2}(|O_2| + |S_2| - |S_2^O| - |S_1|)$. We bound the term $-|S_2^O|$ from below by observing that $|S_2^O| = |S_2| - |S_2^R| \leq |S_2| - 1/3 \cdot |O|$.Thus, we get $|S_{12}| \geq \frac{1}{2}(|O_2| - |S_1| + 1/3 \cdot |O|)$. Therefore, Res_{12} is again of size $|S_1| + |S_{12}| \geq |S_1| - \frac{1}{2}|S_1| + \frac{1}{2}|O_2| + \frac{1}{6}|O| \geq \frac{1}{2}(|O_1| + |O_2|) + \frac{1}{6}|O| = \frac{2}{3}|O|$. ♡

We further study generalizations of this algorithm. We can modify it to produce the same approximation guarantee even if jobs have weights and the goal is to maximize the sum of weights of scheduled jobs. For m machines, we prove that a similar strategy (try all the permutations of multi-machine greedy and take the best), yields an approximation ratio of at least $(\frac{1}{2} + \frac{1}{2m(m-1)})$ (sadly, the running time grows exponentially in m).

Most interestingly, the same technique generalizes to other problems that can be formulated as optimization of several maximization subproblems defined over the same ground set S such that the feasible solutions of every subproblem are subsets of S that form an independence system. The goal is to find a pairwise disjoint feasible solutions to subproblems so that the size of the union of these solutions is maximized. Many natural problems, besides IntervalSelection, can be formulated in this way. For all, let us name the problem of maximum IndependentSet in Red-Blue graphs.

Open Questions. We believe that it is interesting to further study approximation algorithms for different variants of IntervalSelection as well as its generalization. In particular, we are interested in deterministic combinatorial algorithms that give a better approximation ratio, but remain reasonably simple.

References

[1] E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Appl Math*, 1987.

[2] K. Böhmová, Y. Disser, M. Mihalák, and P. Widmayer. Interval selection with machine-dependent intervals. In *WADS*, pages 170–181, 2013.

[3] K. Böhmová, E. Kravina, and M. Mihalák. Approximating interval selection on unrelated machines with unit-length intervals and cores. Unpublished manuscript.

[4] K. Böhmová, E. Kravina, and M. Mihalák. Maximization problems competing over a common ground set and their black-box approximation. Unpublished manuscript.

[5] K. I. Bouzina and H. Emmons. Interval scheduling on identical machines. *J Glob Optim*, 1996.

[6] J. Chuzhoy, R. Ostrovsky, and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. In *FOCS*, pages 348–356, 2001.

[7] J. M. Keil. On the complexity of scheduling tasks with discrete starting times. *Oper Res Lett*, 1992.

[8] A. W. J. Kolen, J. K. Lenstra, C. H. Papadimitriou, and F. C. R. Spieksma. Interval scheduling: a survey. *NRL*, 2007.

[9] M. Y. Kovalyov, C. Ng, and T. E. Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *E J Oper Res*, 2007.

[10] K. Nakajima and S. L. Hakimi. Complexity results for scheduling tasks with discrete starting times. *J Algo*, 1982.

[11] F. C. R. Spieksma. On the approximability of an interval scheduling problem. *J Sched*, 1999.

[12] S. C. Sung and M. Vlach. Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *J Sched*, 2005.

Online non-preemptive scheduling to optimize stretch

Pierre Francois Dutot [*] Erik Saule [†] Abhinav Srivastav (Speaker) [‡]
Denis Trystram [§]

1 Introduction

Scheduling independent jobs that arrive over time is a fundamental problem that arises in a variety of applications. The goal of a scheduling algorithm is to optimize one or more metrics of the quality of service (QoS) delivered to jobs. Some well-studied metrics of interest include throughput, maximum completion time, sum of completion times and total flow time. More often the problem of *fair* scheduling comes into picture. Such problems often arise in the context of web server accommodating requests for retrieving database or web contents. Bender et al. [1] propose the idea of stretch optimization in the context of *fair* scheduling. They defined stretch as a factor by which a job is slowed down with respect to time it takes on unloaded system. Formally, we are given a set of n jobs to schedule on a single processor. Job i is associated with a processing time p_i and a release date r_i before which it cannot be scheduled. In a schedule A, the stretch S_i^A of job i is defined as $S_i^A = \frac{C_i^A - r_i}{p_i}$, where C_i^A denotes the completion time of job i. Our objective is to schedule non-preemptively the stream of the jobs arriving over time (online) so as to minimize the maximum stretch on a single machine.

Bender et al. [1] showed that the problem of optimizing the maximum stretch in a non-preemptive setting cannot be approximated within $O(n^{1-\epsilon})$, unless $P = NP$. Assuming the bound of Δ on the ratio of largest to smallest job processing time, they showed that any online algorithm has at least a competitive ratio of $\Omega(\Delta^{\frac{1}{3}})$. Finally, they provided an online preemptive algorithm using earliest deadline first strategy and showed that it achieves the performance guarantee of $O(\sqrt{\Delta})$.

Later, Legrand et al. [2] showed that FCFS is Δ and Δ^2-competitive for the problem of minimizing the maximum stretch and of minimizing the total stretch problem on a single machine, respectively. They also showed that the problem of optimizing the maximum stretch on single machine with preemption cannot be approximated within the factor of $\frac{1}{2}\Delta^{\sqrt{2}-1}$. *Additionally, they showed that for any algorithm which has a competitive ratio of $O(\Delta^2 - \epsilon)$ for the sum-stretch, there exists an instance that leads to starvation, thus obtaining unbounded max-stretch.* Saule et al. [3] provided tighter lower bounds for the problem on a single machine with the additional constraints that job cannot be preempted. They showed that all approximation algorithms for the single machine problem can not have a competitive ratio better than $\frac{1+\Delta}{2}$.

[*] University of Grenoble Alpes.
[†] esaule@uncc.edu. University of North Carolina at Charlotte. Dept. of Computer Science.
[‡] srivasta@imag.fr. University of Grenoble Alpes.
[§] University of Grenoble Alpes, Institut Universitaire de France.

2 Contributions

Our main contributions are:

- *Non preemptive Max-stretch.* We present an algorithm that achieves a competitive ratio of $1 + \alpha\Delta$, where $\alpha = \frac{\sqrt{5}-1}{2}$, for the single machine non preemptive problem of optimizing the maximum stretch. We also show, using an adversary technique, that there does not exist any algorithm which achieve an approximation ratio better than $1 + \alpha\Delta$.

  The intuition behind the algorithm is derived from instances with just two job sizes, where a large job has to wait some time before it can be considered for scheduling. Additionally, if all jobs are released at time $t = 0$, we can determine the feasibility of scheduling the tasks with a given stretch S. This is due to the fact that each job can be associated with a deadline by which it must complete in order to obtain a stretch less than (or equal to) S. This problem can be solved with the *earliest deadline first (EDF) policy* [4]. Our online algorithm combines these ideas to find a feasible schedule such that the maximum stretch of any job is at most $1 + \alpha\Delta$ times the stretch of some job in the optimal schedule. First for more general case with job sizes varying between 1 and Δ, we show that a suitable partitioning of job sets can be defined, i.e set $\mathcal{L}$ consisting of large jobs and similarly set $\mathcal{S}$, consisting of small jobs. Then our algorithm schedules the jobs from set $\mathcal{L}$ after some initial waiting time (α time the job's processing time) while jobs in set $\mathcal{S}$ are available to the scheduler as soon as they are released. At any time when the processor is idle, our algorithm selects a job (among set of available jobs) using the *earliest deadline first* policy where deadline d_j of a job j is defined as $r_j + Sp_j$ such that S is the smallest stretch estimate with which all the available jobs can be scheduled.

- *Non preemptive Total stretch.* We show that no deterministic algorithm can approximate the minimum total stretch within a factor better than Δ. Then we show that our algorithm for minimizing the maximum stretch achieves at most a competitive ratio of Δ^2 for minimizing the total stretch on a single machine. Therefore, combining this result with the work of Legrand et al. [2], it follows that *there does not exist an algorithm which can have a bi-objective competitive ratio better than* $\{1 + \alpha\Delta, \Delta^2\}$ *for problem of minimizing the maximum stretch and the total stretch, simultaneously.*

3 Analysis of our algorithm

Now, we provide the sketch of the analysis of our algorithm on a single machine. Our proof is based on the comparison of the stretch of the last job in our algorithm to some job in the optimal schedule. We use Alg to denote our algorithm and let Opt denote the optimal algorithm. Let i be the job in Alg that attains the maximum stretch among all the jobs. Without loss of generality, we can assume that i is the last job that runs in Alg. Our aim is to show that $\frac{S_i}{S^*} \leq 1 + \alpha\Delta$, where S^* is the maximum stretch of some job in optimal schedule. Let j be the job in the optimal schedule that completes at/or after $C_i - \alpha\Delta$, where C_i is the completion time of job i in Alg. First, we show that if $p_j \leq p_i$, then our bound holds. For the rest of the analysis, we focus only on cases where no job with processing time smaller or equal to p_i completes after $C_i - \alpha\Delta$.

Later we show some useful properties of the optimal schedule under the constraints where job j is scheduled in Alg. More specifically, we argue that if job j is scheduled later in the optimal schedule than in Alg, then we can lower bound the optimal maximum stretch with

the stretch of job j in Alg. This is due to the two following facts. First if the estimate of maximum stretch is very small, then jobs with large size have higher priorities. Second if the estimate of maximum stretch is very large, then jobs with small size have earlier deadlines and higher priority. Using these properties and EDF policy for scheduling, we split our analysis into two main sub cases: where both job i and j are in set $\mathcal{S}$ and where j is in set $\mathcal{L}$. For each case separately, we find an interesting lower bound on the maximum stretch obtain in the optimal schedule. Then using these bounds, we show in both cases that our algorithm achieves a competitive ratio of $1 + \alpha\Delta$.

The analysis for the total stretch is a proof by induction. By indexing the jobs based on their release times, we bound the total stretch of the schedule produced by our algorithm within a Δ^2 factor of the optimal schedule which is optimal for the total stretch. Additionally we provide a simple instance for which this bound is tight.

4 Concluding remarks

We conjecture that similar waiting strategies can be used for minimizing other maximum objective functions. Further, we conjecture that such algorithms will also have bounded objective values for minimizing sum objective functions. Another interesting issue is how to extend these techniques on parallel machines with identical, related and unrelated processors. We believe that single machine algorithm can be modified to parallel machines with a constant increase in competitiveness.

References

[1] M. Bender, S.Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–279, 1998.

[2] A. Legrand, A. Su, and F. Vivien. Minimizing the stretch when scheduling flows of divisible requests. *Journal of Scheduling*, 11(5):381–404, 2008.

[3] E. Saule, D. Bozdag, and U. V. Çatalyürek. Optimizing the stretch of independent tasks on a cluster: From sequential tasks to moldable tasks. *Journal of parallel and distributed computing*, 2012.

[4] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Handbook on Scheduling*. Springer-Verlag, Inc., 2007.

Flow shop problems with synchronous movement

Stefan Waldherr Sigrid Knust (Speaker) *

1 Introduction

A flow shop with synchronous movement ("synchronous flow shop" for short) is a variant of a non-preemptive permutation flow shop where transfers of jobs from one machine to the next take place at the same time (cf. [2, 3, 4]). Given are m machines $M_1, \ldots, M_m$ and n jobs where job j consists of m operations $O_{1j} \to O_{2j} \to \ldots \to O_{mj}$. Operation O_{ij} has to be processed without preemption on machine M_i for p_{ij} time units. In a feasible schedule each machine processes at most one operation at any time, each job is processed on at most one machine at any time, and the jobs are processed in the predefined order.

The processing is organized in synchronized cycles since jobs have to be moved from one machine to the next by an unpaced synchronous transportation system. This means that in a cycle all current jobs start at the same time on the corresponding machines. Then all jobs are processed and have to wait until the last one is finished. Afterwards, all jobs are moved to the next machine simultaneously. The job processed on the last machine M_m leaves the system, a new job (if available) is put on the first machine M_1. As a consequence, the processing time of a cycle is determined by the maximum processing time of the operations contained in it. Furthermore, only permutation schedules are feasible, i.e. the jobs have to be processed in the same order on all machines. The time at which a job j has been processed on all machines and leaves the system is called its completion time C_j. We assume that a job can only be accessed after the whole cycle has been completed, i.e. the job may have to wait until all jobs on the other machines in the corresponding cycle are finished.

The goal is to find a sequence (permutation) of the jobs such that a given objective function (e.g. the makespan $C_{\max} = \max C_j$ or the maximum lateness $L_{\max}$ involving due dates) is minimized. With each sequence a corresponding (left-shifted) schedule is associated in which each operation starts as early as possible.

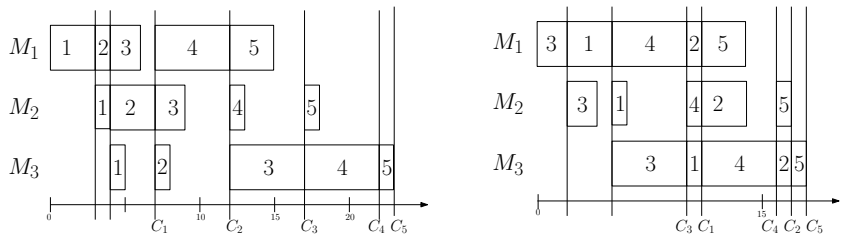


Figure 1: A feasible and an optimal schedule for a synchronous flow shop

*{swaldher,sknust}@uni-osnabrueck.de. Institute of Computer Science, University of Osnabrück, Germany, supported by the Deutsche Forschungsgemeinschaft, KN 512/7-1.

Figure 1 shows two feasible synchronous flow shop schedules for a small example with three machines and five jobs. The vertical lines indicate the cycles and show when the jobs are transferred to the next machine. In the left part of the figure the schedule corresponding to the job sequence $(1, 2, 3, 4, 5)$ is shown. Here, we can observe a long waiting period (idle time) on machine M_3 between processing jobs 2 and 3. In the right part of the figure a better (and even optimal) schedule corresponding to the job sequence $(3, 1, 4, 2, 5)$ can be found.

In order to indicate synchronous movements, in [2] the notation "synmv" was added to the β-field of the well-known $\alpha|\beta|\gamma$ scheduling classification. Hence, the notation $F|synmv|f$ refers to a synchronous flow shop with objective function f.

Motivated by practical applications (cf. [5]) and previous work concerning classical flow shop scheduling problems, we also investigate special cases where the processing times of the cycles are only determined by a subset of so-called dominating machines. We say that a machine M_k dominates M_l whenever $\min_{j=1,\ldots,n} p_{kj} \geq \max_{j=1,\ldots,n} p_{lj}$, i.e. the smallest processing time of any job on machine M_k is at least as large as the largest processing time of any job on machine M_l. More generally, we call a set $\{M_i \mid i \in \mathcal{I}\}$ of machines dominating, if $\min_{i\in\mathcal{I}} \min_{j=1,\ldots,n} p_{ij} \geq \max_{h\notin\mathcal{I}} \max_{j=1,\ldots,n} p_{hj}$, i.e. there is a set of machines which dominate all other machines. If the set $\{M_i \mid i \in \mathcal{I}\}$ describes a set of dominating machines, we use the notation "dom($\mathcal{I}$)" in the β-field of the classification scheme.

For the processing times on non-dominating machines we may assume that they are either arbitrary values, or that they are job-independent, i.e. $p_{ij} = p_i$ for all j and all $i \notin \mathcal{I}$. The second variant may be reasonable in practice if on the non-dominating machines work processes like insertion or removal of jobs are performed where the processing time is independent of the individual job. In this situation we may even assume that all processing times on the non-dominating machines are the same and equal to zero. To denote this special situation, we add "$p_{ij}^{ndom} = 0$" to the β-field.

2 Complexity results and solution algorithms

As already observed in [3], the two-machine synchronous flow shop problem $F2|synmv|C_{\max}$ is equivalent to the two-machine flow shop problem $F2|no\text{-}wait|C_{\max}$. It can be solved as a traveling salesman problem with the special costs $c_{ij} = \max\{b_i, a_j\}$ in $\mathcal{O}(n \log n)$ by the algorithm of Gilmore and Gomory [1].

In [4] the following new complexity results for synchronous flow shop problems with and without machine dominance were obtained:

Theorem 1 *Problem $F3|synmv|C_{\max}$ is strongly $\mathcal{NP}$-hard.*

Theorem 2 *Problems $F|synmv, dom(\mathcal{I})|C_{\max}$ and $F|synmv, dom(\mathcal{I})|\sum C_j$ are strongly $\mathcal{NP}$-hard for $|\mathcal{I}| = 1$ if the processing times on non-dominating machines are arbitrary.*

Theorem 3 *Problem $F|synmv, dom(\mathcal{I}), p_{ij}^{ndom} = 0|\sum C_j$ can be solved in $\mathcal{O}(n \log n)$ and problem $Fm|synmv, dom(\mathcal{I})|\sum C_j$ can be solved in $\mathcal{O}(n^m \log n)$ for $|\mathcal{I}| = 1$.*

Theorem 4 *Problem $F|synmv, dom(\mathcal{I}), p_{ij}^{ndom} = 0|L_{\max}$ can be solved in $\mathcal{O}(n^3 \log n)$ and problem $Fm|synmv, dom(\mathcal{I})|L_{\max}$ can be solved in $\mathcal{O}(n^{m+2} \log n)$ for $|\mathcal{I}| = 1$.*

Theorem 5 *Problem* $F|synmv, dom(\mathcal{I}), p_{ij}^{ndom} = 0|C_{\max}$ *is strongly* $\mathcal{NP}$*-hard for* $|\mathcal{I}| = 2$*.*

Theorem 6 *Problem* $Fm|synmv, dom(\mathcal{I}), p_{ij}^{ndom} = 0|L_{\max}$ *is strongly* $\mathcal{NP}$*-hard for each fixed* $m \geq 2$ *and each set* $\mathcal{I}$ *with* $|\mathcal{I}| = 2$*.*

The strongly $\mathcal{NP}$-hard problem $F|synmv, dom(\mathcal{I})|C_{\max}$ with two dominating machines can be formulated as a special vehicle routing problem with costs $c_{ij} = \max\{b_i, a_j\}$ where every route must contain a fixed number of nodes. Besides an integer linear programming formulation we present heuristic methods which are based on the Gilmore/Gomory algorithm.

References

[1] P.C. Gilmore and R.E. Gomory (1964). Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research* 12, 655–679.

[2] K.-L. Huang (2011). *Flow shop scheduling with synchronous and asynchronous transportation times*. Ph.D. Thesis, The Pennsylvania State University.

[3] B. Soylu, Ö. Kirca and M. Azizoğlu (2007). Flow shop-sequencing problem with synchronous transfers and makespan minimization. *International Journal of Production Research* 45, 3311–3331.

[4] S. Waldherr and S. Knust (2015). Complexity results for flow shop problems with synchronous movement. *European Journal of Operational Research* 242, 34–44.

[5] S. Waldherr and S. Knust (2014). Two-stage scheduling in shelf-board production: A case study. *International Journal of Production Research* 52, 4078–4092.

Synchronous flow shops with setup times

Stefan Waldherr (Speaker) Sigrid Knust *

1 Introduction

Synchronous flow shops are a variant of a non-preemptive permutation flow shop. Jobs have to be moved from one machine to the next by an unpaced synchronous transportation system, which implies that the processing is organized in synchronized cycles. In each cycle the current jobs start at the same time on their corresponding machines and are moved to the next machine simultaneously once all jobs have finished processing. The time of a cycle is thus determined by the maximum processing times of the jobs. The synchronous movement is indicated by the notation "synmv" in the β-field of the well-known $\alpha|\beta|\gamma$ notation (cf. [3]). For two machines the problem $F2|\text{synmv}|C_{\max}$ is equivalent to the problems $F2|\text{no-wait}|C_{\max}$ and $F2|\text{block}|C_{\max}$ and can be solved in polynomial time via the algorithm of Gilmore and Gomory (cf. [1, 4]). For more than two machines the synchronous flow shop is no longer equivalent to the no-wait flow shop or the flow shop with blocking constraints. In [6] it was shown that the synchronous flow shop problem is $\mathcal{NP}$-hard for three or more machines.

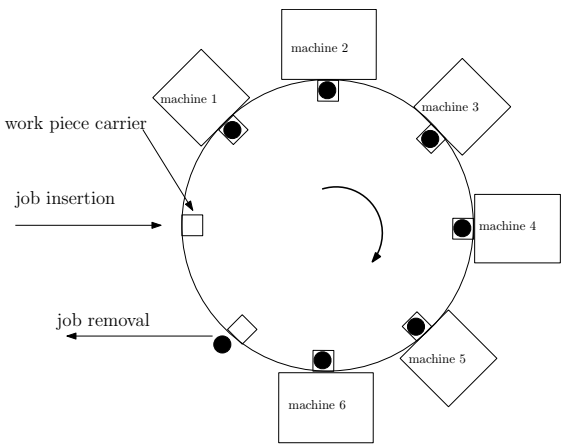


Figure 1: Production unit resembling a synchronous flow shop with setup times

In [5] a practical application of synchronous flow shops in furniture manufacturing is considered. Figure 1 depicts an exemplary production unit. It resembles an unpaced synchronous assembly line in which all jobs are transported to the next machine synchronously by the underlying conveyor system. The problem is distinct from robotic cell

*{swaldher,sknust}@uni-osnabrueck.de. Institute of Computer Science, University of Osnabrück, Germany, supported by the Deutsche Forschungsgemeinschaft, KN 512/7-1.

scheduling (e.g. [2]) where jobs can be transported independently between the machines and the transportation of the jobs has to be coordinated. Specialized resources or work piece carriers are required to transport the jobs between machines and each job can be transported by a subset of all available carriers. The work piece carriers are fixed to the conveyor system and are required by a job from its insertion until its removal. After a job is removed from its carrier and another job can be inserted, we can either choose a new job that is compatible with the carrier or elect to change the carrier, requiring a setup time during which the production unit can not move.

In the practical application the jobs belong to different families $\mathcal{F}$ with sizes $n_1, \ldots, n_{|\mathcal{F}|}$. All jobs in the same family may be transported using the same resources. If a job belonging to family g occurs m positions after a job from family f, the setup time s_{fg} occurs. For $f = g$ we assume $s_{ff} = 0$. The setup times may be sequence independent ($s_{fg} = s_g$) or even constant ($s_{fg} = s$ for $f \neq g$). Initially (i.e. for the first m jobs), setup times s_{0g} occur. In the presence of job families, the two-machine synchronous flow shop problem with constant setup times is already $\mathcal{NP}$-hard.

2 Decomposition approaches

Due to the complexity of the problem we will discuss decomposition strategies to find good solution for the synchronous flow shop with setup times. Therein, we consider the two subproblems which determine the makespan of the problem:

(P1) Find a permutation of jobs which minimizes the sum of cycle times, i.e. the makespan of the schedule ignoring setup times.

(P2) Find a permutation of job families that minimizes the sum of setup times without assigning the actual jobs thus ignoring processing times in the cycles.

We propose the following two decomposition strategies:

(D1) Solve problem (P1) and then assign resources to the obtained sequence such that setup times are minimized without altering the sequence of jobs unless no feasible assignment of resources is possible.

(D2) Solve problem (P2) and then determine the actual sequence of individual jobs within the assigned job families to minimize the sum of cycle times.

Within the first decomposition, the first part can be solved to optimality in polynomial time in the two-machine case while for three or more machines we will present efficient heuristics. The second part can then be solved by a greedy approach. Within the second decomposition, the sum of setup times can be minimized in polynomial time if setup times are constant. For the second part we present heuristic methods.

We compare the two decomposition strategies for practical instances obtained from an industrial partner as well as for generated instances. Further, within the first decomposition approach, we improve heuristics presented in [4] for the case of synchronous flow shops without setup times.

References

[1] P.C. Gilmore and R.E. Gomory (1964). *Sequencing a one state-variable machine: A solvable case of the traveling salesman problem*. Operations Research 12, 655–679.

[2] N.G. Hall and H. Kamoun and C. Sriskandarajah (1997). *Scheduling in robotic cells: Classification, two and three machine cells*. Operations Research 45, 421–439.

[3] K.-L. Huang (2011). *Flow shop scheduling with synchronous and asynchronous transportation times*. Ph.D. Thesis, The Pennsylvania State University.

[4] B. Soylu, Ö. Kirca and M. Azizoğlu (2007). *Flow shop-sequencing problem with synchronous transfers and makespan minimization*. International Journal of Production Research 45, 3311–3331.

[5] S. Waldherr and S. Knust (2014). *Two-stage scheduling in shelf-board production: A case study*. International Journal of Production Research 52, 4078–4092.

[6] S. Waldherr and S. Knust (2015). *Complexity results for flow shop problems with synchronous movement*. European Journal of Operational Research 242, 34–44.

Personalized nurse rostering through linear programming *

Han Hoogeveen †and Tim van Weelden ‡

1 Introduction

Our purpose is finding good, personalized rosters for the personnel of the Cardiothoracic Department of UMC Utrecht. The department contains several treatment rooms at different care levels. Taking care of the patients is done 24/7 by approximately 52 nursing employees, each of which is available for a given number of hours, which differs per nurse. There are four types of nurses: student nurses, basic nurses, Medium Care nurses, and Senior nurses. The student nurses are trainees, who work mostly in the day-shift. The basic nurses have a basic qualification only; they can work in all shifts. Next to the basic qualification, the MC and senior nurses have an MC/senior qualification.

Rosters are created for a period of 6 weeks. Each day is divided in three shifts: day (early) shift, late shift, and night shift. For all shifts in the planning period, we know the minimally required number of nurses per qualification. The nurses each have there own preferences regarding their schedule; the hospital tries to obey these as much as possible. Currently, rosters are created by hand, since the planning system in use is not able to find reasonable rosters. We present an ILP-based algorithm to solve this problem; it is based on the approach in [2] that was used to solve a simpler problem without qualifications. Moreover, we present a repair heuristic to adapt to changing nurse availability.

Our contribution. We describe an algorithm that can find near-optimal solutions for real-life rostering problems while taking personal preferences into account. This hot topic (in the Netherlands) has not been well-studied in the literature (see [1, 2]).

2 Constraints

The goal is to find a feasible roster for each employee such that there are enough nurses with the right qualifications in each shift, and nurse preferences are satisfied as much as possible. Having surplus employees is allowed, preferably well-spread over the day-shifts, but never for night-shifts. Moreover, the presence of student nurses should be evenly divided over the day-shifts.

*The working paper this abstract is based can be found at http://www.staff.science.uu.nl/ hooge109/Weelden.pdf

†j.a.hoogeveen@uu.nl. Department of Information and Computing Sciences, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.

‡timvanweelden@gmail.com. Department of Information and Computing Sciences, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.

The individual rosters have to obey several hard constraints. First of all, the number of workshifts should equal the appointment size, but a deviation of 1 is allowed; moreover, every employee should work at most 6 days per week. Next, the night-shifts must be rostered in prespecified blocks. There should be at most 4 night-shifts per 6 weeks, and after a night-shift the nurse must have at least two days off. Furthermore, a nurse must have the same shift on Saturday and Sunday. Moreover, days off must come in blocks of size two or more. Because of healthy rostering, a late-shift cannot be followed by a day-shift. Finally, senior nurses are entitled to two 'quality days'.

Furthermore, there are many soft constraints concerning individual rosters, which are used to determine the quality of the individual rosters. In general, the number of shifts per week should be the same each week, and the night-shifts and weekend-shifts should be well spread over the 6-week period. Examples of personal preferences are to have a regular or occasional day off, to have a specific shift on a day (some nurses perform self-scheduling), to have as many/few late/night shifts as possible, to have as many days off after night-shift as possible, and to have bounds on the number of consecutive shifts.

3 Solution approach

The basic idea (see [2]) is to formulate the problem as an ILP. For each nurse, we construct a set of rosters; each one satisfies all hard constraints issued by both the hospital and the nurse. In constrast to [1], we generate these up to 200.000 rosters per nurse beforehand. For each roster we compute its cost on basis of how well it respects the personal preferences of that nurse. The cost is then scaled to $[0, 1]$ and the ones with cost > 0.5 are removed.

For each roster $s \in S$ we introduce a binary variable x_s indicating whether s is chosen. Roster s contains information as to which nurse it belongs and which shifts it covers. Using these variables, we formulate the constraint that each nurse gets one roster, and that for each combination of a shift and a qualification, we meet the minimum occupancy. We further have variables measuring shortage, surplus, and excessive surplus per shift per qualification; these are penalized in the objective function. The objective is to minimize the total cost of the chosen rosters plus the total shortage, surplus, and excessive surplus penalty.

As the ILP cannot handle these large numbers of variables, we restrict ourselves to a selection of these rosters, for which we then solve the ILP. To make this selection, we solve the LP-relaxation through revised simplex; we put each variable that gets selected in the revised simplex in a column pool. Furthermore, we add variables to this pool that have small reduced cost based on the dual multipliers of the optimum solution. Moreover, we find additional columns for the pool by applying small mutations. Finally, we let the ILP run for this pool of rosters for 15 minutes. In the resulting solution almost all occupancy constraints are met (sometimes one weekend night-shift needs one more nurse) with good rosters (most of the soft preferences were fulfilled, but sometimes there was a single day off).

We further have implemented a set of heuristics for repairing roster problems, which also can be used to resolve new requests. The philosophy behind these was to make small changes, where each change would resolve an 'important' problem at the expense of creating an 'unimportant' problem. In this way, we can also resolve requests like people who should cooperate regularly. Furthermore, in this way we can determine

consecutive 6-weeks rosters independently, which are then glued together.

References

[1] F. He and R. Qu (2012). A constraint programming based column generation approach to nurse rostering problems. *Computers & Operations Research, 39*, pp. 3331–3343.

[2] H. Hoogeveen and E. Penninkx (2007). Finding near-optimal rosters using column generation. Technical Report UU-CS-2007-002, Department of Information and Computing Sciences, Utrecht University. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.8622&rep=rep1&type=pdf

Simulation-guided Tree Search for Optimization of PET-CT Images Acquisition Planning

François Roucoux (Speaker) * Renaud Florquin †

1 Introduction

PET-CT is a medical imaging technique that combines a positron emission tomography scanner (PET) and a more classical x-ray computed tomography scanner (CT) [1]. The PET functional images depict the spatial distribution of metabolic activity in the body. This information is superposed with CT anatomical images. PET-CT is increasingly used in oncology for diagnosis, treatment evaluation and follow-up.

PET-CT requires the injection of radiopharmaceuticals into the patient's body. Fluorodeoxyglucose marked with radioactive fluorine$_{18}$ (FDG) is the most commonly used. Due to radioactive decay, the half-life period for FDG is 109 minutes. It means that every 109 minutes, you should inject twice the dose of FDG to obtain the same image quality.

Each day in a PET-CT unit, a series of patients are planned for imaging. Patients arrive one after the other. Each imaging process begins with a blood test, then FDG is injected. The radioactive activity injected into the patient depends on his body mass index and the type of examination to be carried out.

After the injection, the patient needs to rest ideally 60 to 70 minutes. During this so called incorporation period, the FDG is preferentially taken up by tissues with high metabolism. If the incorporation period is too short, the images will be of poor quality. A period of more than 70 minutes disqualifies the patient to access clinical studies. So, conformity of incorporation time is crucial.

After incorporation, the patient goes on the PET-CT. The images acquisition takes between 15 and 30 minutes depending on patient's size and the type of examination. Several patients can incorporate the tracer at the same time but they are imaged one by one.

A wide range of adverse events causing delays are possible: patients arriving too late, nonfasting or diabetic patients requiring lengthy glucose balancing, urgent patient to be interleaved, detection of blood anomalies, technical problems...

The PET-CT imaging is a stressful process that requires a high level of experience from the staff in charge. Due to the radioactive decay, a succession of small delays will cause a much higher tracer consumption. Because the total dose of tracer for one day is ordered the day before, there is a risk of not having enough FDG to process all patients.

*francois.roucoux@uclouvain.be. Faculty of Medicine, Academic Center for Family Practice, Catholic University of Louvain, 1, Place de l'Université, B-1348 Louvain-la-Neuve (Belgium).

†renaud.florquin@palantiris.com. Palantiris Ltd., Einstein Business Center, Rue du bosquet, 15 A, B-1435 Mont-St-Guibert (Belgium).

2 Goals

One obstacle to the wider use of PET-CT is the high cost of tracers production. Our first goal is radiopharmaceuticals savings by planning optimally the sequence of patients. This planning should take into account patients characteristics, the type of imaging and the risk of adverse events occurrence whose impact shall be minimized.

Our second goal is to improve the incorporation period conformance. In the PET-CT facilities that we visit, the periods of incorporation tends to become erratic over the day. On average, 70% of patients have a non-comformant incorporation.

PET-CT is increasingly used to verify the effectiveness of treatments. Patients undergo iterative imaging and quantitative comparisons of the metabolic information are performed. To ensure reliable comparisons, the reproducibility of incorporation durations from one imaging to another becomes critical. For now, it is very difficult for the teams in place to ensure this reproducibility. We will help them on this point too.

3 Method

We investigated the activity of one representative PET-CT department. We started by mining information contained into the logs of the PET-CT, injection devices, agenda and medical records. Based on this information covering 9 months of activity, we inferred a first model of the imaging process. It contained the case mix, sequences of activities, duration distributions, and probabilities of adverse events occurence.

In a second phase, we accompanied the staff members in their daily work to experience problems they faced and how they solved them. This information allowed us to refine our model and elicitate domain specific problem solving heuristics and strategies.

Based on this second model, we implemented an imaging process simulator. This simulator is able to mimic the behavior of the unit until the end of the day according to a selected sequence of patients.

To find the patients sequence optimizing our goals, we coupled the simulator with a simulation-guided tree search procedure also called Monte Carlo Tree Search (MCTS). MCTS finds near-optimal decision sequences in domains representable as decision trees like planning problems [2].

MCTS constructs incrementally an asymetrical tree whose nodes represent the states of the domain. Addition of child nodes corresponds to actions selection. At each iteration, a policy attempting to balance exploration vs. exploitation is used to find the next node to consider. From that node, simulations are run to determine the best action to choose. The child node resulting from the selected action is added and statistics (average rewards) of its ancestors are updated.

MCTS is a rich framework of optimization procedures [2]. Each steps of the framework (child node selection policy, simulation policy, tree statistics update policy...) is adaptable. As child node selection policy, we use a slight variation of UCB1 [3]. As simulation policy, transitions (selection of the next patient to be processed) are made according to a statistically biased policy. Most of the time, the policy makes uniform random choices but from time to time, it uses some of the domain specific heuristics that we elicited. The ratio random/heuristic choices is adapted at runtime and changes with the depth of exploration.

In case of FDG shortage, we adapt the MCTS procedure: two series of simulations

are now performed. One considers a minimal injected dose reduction for each patients (impact on image quality). The second favors earlier injection time (impact on incorporation duration). The two strategies are implemented as a "what if" feature of the planner. It is up to the end user to choose the strategy to apply.

4 Results

Our planner is used offline a few days before the imaging day to plan the sequence of patients, and computes the total ordered FDG. It is also used online during the imaging day to give real-time directives to the staff. The state of the simulation is continuously updated by feedback received from staff members. This two stages planning achieves average savings of 15% of the total FDG daily dose. It also has a positive impact on incorporation conformance and reproducibility. More data are needed to quantify precisely this impact.

An unexpected benefit is the reduction of the stress level. The planner provides a forward-looking vision of the dose consumption. It reassures the team about the possibility of finishing the day or suggests strategies to mitigate a possible FDG shortage.

5 Conclusion and future works

By coupling a simulation-guided tree search planning procedure with a simulator, we where able to spare 15% of the total daily ordered FDG in a PET-CT unit. Our planner also improves significantly the conformance and reproducibility of incorporation periods and diminish the stress level of the staff.

Beyond performance, the appeal of the technique is its simplicity of implementation. It only requires a simulator of the domain. Dedicated heuristics can be used but are not mandatory. It copes well with unforseen adverse events and mitigates their impact. It is an anytime procedure usable offline and online for real time planning.

Currently, our planner depends on human feedback to update its model. We will streamline this process by retreiving automatically these data. We also plan to address larger problem instances by planning the activities of a whole nuclear medicine department.

References

[1] E. Lin, and A. Alavi (2009). *PET and PET/CT: A Clinical Guide. 2nd ed.* Thieme Publishers, Stuttgart, Germany.

[2] C. Browne, E.J. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton (2012). *A Survey of Monte Carlo Tree Search Methods.* IEEE Trans. Comput. Intellig. and AI in Games 4 (1), 1–43.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer (2002). *Finite-time Analysis of the Multiarmed Bandit Problem.* Mach. Learn., vol. 47, no. 2, pp. 235–256.

Speed Scaling with Variable Electricity Rates and Speed Limits

Antonios Antoniadis [*] Peter Kling [†] Sebastian Ott [‡]
Sören Riechers (Speaker) [§]

1 Introduction

Our work joins the prominent line of research that studies the *speed scaling* technique (see [1] for a survey), where a device can adapt its speed and, thus, energy consumption to the current requirements. The first theoretical study of speed scaling models is due to Yao et al. [4]. The authors model the power consumption of a processor running at speed s by a *power function* $P(s) = s^\alpha$, $\alpha > 1$. They design a polynomial-time algorithm to compute an energy-minimal schedule for a given number of jobs, each with a release time, deadline, and workload. The focus of our work lies in two aspects: *dynamic speed limits* and *dynamic electricity costs*. In practice, devices being sensitive to environmental conditions, speed limits become highly dynamic. Also, dynamic electricity costs can have a huge impact on the operating costs in data centers. In the literature, these aspects are often assumed to be non-existing (speed limits) or constant (energy costs).

Problem Description & Results. We study scheduling of n jobs $J \coloneqq \{1, 2, \dots, n\}$ on a single, speed-scalable processor. Here, speed-scalable means that the processor's speed $s \in \mathbb{R}_{\geq 0}$ is controlled by the scheduler. Its power consumption is modeled by a *power function* $P\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}, s \mapsto s^\alpha$. That is, while running at speed s, energy is consumed at a rate of $P(s) = s^\alpha$, $\alpha > 1$. In addition to these classical speed scaling properties, we have the constraint that the maximal speed at time t is bounded. We model this via a *maximum speed function* $s_{\max}\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. Further, there is a cost factor associated with every time point $t \in \mathbb{R}_{\geq 0}$, specifying the cost per unit of energy. The cost factor is modeled via a *cost factor function* $c\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{>0}$.

Each job $j \in J$ comes with a *release time* $r_j \in \mathbb{R}_{\geq 0}$, a *deadline* $d_j \in \mathbb{R}_{\geq 0}$, and a *workload* $w_j \in \mathbb{R}_{\geq 0}$. For each time $t \in \mathbb{R}_{\geq 0}$, a *schedule* S must decide which job to process at what speed. Preemption is allowed, so that a job may be suspended and

[*] aantonia@mpi-inf.mpg.de. Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany.
[†] pkling@sfu.ca. School of Computing Science, Simon Fraser University, Burnaby, Canada. Work done while at the University of Pittsburgh and supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).
[‡] ott@mpi-inf.mpg.de. Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany.
[§] soeren.riechers@upb.de. Heinz Nixdorf Institute and Computer Science Department of University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany. This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

resumed later on. We model a schedule S by a *speed function* $s\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ and a *scheduling policy* $\mathcal{J}\colon \mathbb{R}_{\geq 0} \to J$. Here, $s(t)$ denotes the speed at time t, and $\mathcal{J}(t)$ the job that is scheduled at time t. A *feasible* schedule must finish all jobs within their release time/deadline intervals $[r_j, d_j)$ without exceeding the maximum speed function $s_{\max}$. More formally, we require $s(t) \leq s_{\max}(t)$ for all $t \in \mathbb{R}_{\geq 0}$ and $\int_{\mathcal{J}^{-1}(j)\cap[r_j,d_j)} s(t)\,\mathrm{d}t \geq w_j$ for all $j \in J$. The total energy consumption of a schedule S is given by $\int_0^\infty P(s(t))\,\mathrm{d}t$, and its total energy cost by $E(s) \coloneqq \int_0^\infty c(t) \cdot P(s(t))\,\mathrm{d}t$. For technical reasons, we restrict ourselves to functions in C_{pr} (i.e., to functions that are right-continuous with finitely many discontinuities), which covers all practically relevant schedules. Our goal is to find a feasible schedule of minimum cost.

As many recent results in this area, we use techniques known from convex programming to design and analyze an algorithm. However, since we allow almost arbitrary dynamics for the speed limits, our problem entails some interesting challenges. The naive way to model this leads to an infinite number of variables, rendering standard convex programming techniques infeasible. Using techniques from calculus of variations, we provide a framework to deal with such situations. In particular, we derive an optimality condition similar to the well-known KKT conditions. Building on top of this condition, we develop a polynomial-time optimal algorithm. Assured by recent work (e.g., [2, 3]) that has evidenced of the KKT conditions' usefulness in speed scaling problems, we expect our techniques to be helpful even beyond the scope of our problem.

2 Balance for Optimality

The following theorem is crucial for our polynomial time algorithm.

Theorem 1 *A feasible schedule S is optimal if and only if it is non-wasting and work-balanced.*

Here, *non-wasting* and *work-balanced* are natural structural properties expressing schedules that distribute the jobs' workload "as evenly as possible" while taking constraints (e.g., the release times/deadlines or the speed limits) and cost factors into account.

Let $\mathcal{S} \coloneqq \prod_{j\in J} \mathcal{S}_j$, where $\mathcal{S}_j$ denotes a class of functions containing possible speed functions for a job $j \in J$. In order to prove Theorem 1, we can formulate our optimization problem as the mathematical program (SP)

$$\begin{aligned}
\min_{s\in\mathcal{S}} \quad & E\left(\sum_{j\in J} s_j\right) && \\
\text{s.t.} \quad & \sum_{j\in J} s_j(t) \leq s_{\max}(t) && \forall t \geq 0 \qquad (1)\\
& \int_{r_j}^{d_j} s_j(t)\,\mathrm{d}t \geq w_j && \forall j \in J \qquad (2)
\end{aligned}$$

Having an infinite number of constraints in (SP), standard convex programming techniques become infeasible. In order to develop an extension of the well known KKT conditions to solve our problem, we now consider a generalized infinite program.

Let $N, m, n \in \mathbb{N}$ and $\mathcal{F}_j \coloneqq \{\, g\colon \mathbb{R} \to \mathbb{R} \mid g \in C_{pr} \wedge \forall x \notin I_j\colon g(x) = 0 \,\}$ for $j \in \{1, \ldots, N\}$ and I_j any interval. We consider an optimization problem for functionals

over the set $\mathcal{F} \coloneqq \prod_{j=1}^{N} \mathcal{F}_j$ together with $m+n$ constraints. The j-th component of $f \in \mathcal{F}$ therefore is a piecewise continuously differentiable function g with $g|_{\mathbb{R}\setminus I_j} = 0$. We view the vectors $f \in \mathcal{F}$ as vector-valued functions $f\colon \mathbb{R} \to \mathbb{R}^N$.

We have an *objective function* $L\colon \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}$ as well as two types of *constraint functions* $G_k, H_l\colon \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}$ for $k \in \{1, 2, \ldots, m\}$ and $l \in \{1, 2, \ldots, n\}$. All these functions are assumed to be piecewise differentiable and convex in their second argument. We write ∇L (and similar for the other functions) to reference to the gradient of L taken with respect to the components of the second argument $y \in \mathbb{R}^N$ and $\nabla_j L$ for the j-th component of L's gradient. Let I be any interval in $\mathbb{R}$. The considered general optimization problem (GP) is

$$\begin{aligned}
\min_{f \in \mathcal{F}} \quad & \int_I L(x, f(x))\,\mathrm{d}x & & \\
\text{s.t.} \quad & G_k(x, f(x)) \leq 0 & \forall x \in I, k \in \{1, 2, \ldots, m\} \quad & \text{(I)} \\
& \int_I H_l(x, f(x))\,\mathrm{d}x \leq 0 & \forall l \in \{1, 2, \ldots, n\} \quad & \text{(II)}
\end{aligned}$$

Using techniques from calculus of variations, we derive the following optimality condition for this generalized program.

Theorem 2 (Extended KKT conditions) *Assume that $f \in \mathcal{F}$ is a feasible solution for (*GP*) with finite solution value. Furthermore, assume that there exist functions $\lambda_k\colon I \to \mathbb{R}_{\geq 0}$, $\lambda_k \in C_{pr}$, and constants $\mu_l \in \mathbb{R}_{\geq 0}$ such that the following properties hold:*

1. *For all $j \in \{1, 2, \ldots, N\}$ and $x \in I_j$, we have*

$$\nabla_j L(x, f(x)) + \sum_{k=1}^{m} \lambda_k(x) \cdot \nabla_j G_k(x, f(x)) + \sum_{l=1}^{n} \mu_l \cdot \nabla_j H_l(x, f(x)) = 0. \quad (3)$$

2. *For all $k \in \{1, 2, \ldots, m\}$ and $x \in I$, we have $\lambda_k(x) \cdot G_k(x, f(x)) = 0$.*

3. *For all $l \in \{1, 2, \ldots, n\}$, we have $\mu_l \cdot \int_I H_l(x, f(x))\,\mathrm{d}x = 0$.*

*Then f is an optimal solution to (*GP*).*

We can apply this theorem to our mathematical program (SP) and thus solve our problem. Also, we expect that this theorem can be useful for other models where one needs to cope with continuous functions resulting in infinite mathematical programs.

References

[1] Susanne Albers. Algorithms for dynamic speed scaling. In *Proc. of STACS 2011*, pages 1–11.

[2] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):3:1–3:39, 2007.

[3] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with a solar cell. *Theoretical Computer Science*, 410(45):4580–4587, 2009.

[4] Frances Foong Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Proc. of FOCS 1995*, pages 374–382.

Semi-Online Minimum Makespan Scheduling with Restricted Assignment

Matthias Hellwig * Csanád Imreh (Speaker) †

1 Introduction

Minimum makespan scheduling is a prominent online problem introduced by Graham [5] already in 1966. In this problem we have n jobs $J_1, \ldots, J_m$ and m machines $M_1, \ldots, M_m$. At each time $t = 1, \ldots, n$ we are presented the job J_t and we have to assign it without any knowledge of the future incoming jobs to one of the machines, which causes an additional load of p_t on the machine it was assigned to. At some time fixed time t the total load of machine M_j is denoted with $\ell(j)$. The goal of minimum makespan scheduling is to minimize the maximum load $\max_{j=1,\ldots,m} \ell(j)$ on the machines after all jobs have been assigned. We consider a variant in which there are also machine eligibility constraints. In this variant for each job J_t there is a subset $E_t \subseteq \{M_1, \ldots, M_m\}$ of the machines specified and the job J_t may be placed only on one of the machines in E_t. In classical online minimum makespan scheduling jobs have to be assigned irrevocably to the machines at the time at they arrive. By contrast in semi-online makespan scheduling with job migration jobs may be reassigned up to a limited amount. This variant we study is also known as load balancing of permanent tasks for current load with machine restrictions and job migrations [2]. In this problem we compare the makespan of our algorithm at any time to the optimal offline makespan.

It is not difficult to see that a lower bound on the competitiveness for any deterministic algorithm of $\Omega(\log m)$ given by Azar et al. [3] for online minimum makespan with machine restrictions transfers to the model with job migrations when we allow that a number of $o(n)$ jobs may be reassigned. Thus it is common to use an amortized notion to account for job migration. We measure the total processing times of jobs migrated divided by the total processing times of the jobs we have to assign. This notion was also used in [9], and a similar measure can be found e. g. in [8].

In this talk we present our results obtained so far on this problem. First we show an algorithm that has sublogarithmic competitiveness using sublogarithmic migration. And later we consider a special case where a better result can be achieved.

We note that there are many models for minimum makespan scheduling with machine eligiblity constraints. Several of them consider special structures of the sets of machines to which the jobs are eligible. See [7] for a survey. For restricted machines not using job migration Azar et al. give a lower bound of $\Omega(\log m)$ in [3]. This bound holds even

*hub01@matthias-hellwig.de. Noser Engineering AG,

†cimreh@inf.u-szeged.hu. Institute of Informatics, University of Szeged, P. O. Box 652, H-6701 Szeged Hungary.

for the special case of unit sized jobs. [2] provides a survey on different models in load balancing, in particular for machine restrictions and job reassignments.

For the specific model we consider Chaudhuri et al. present a lower bound in [6]. They show that any algorithm that is optimal in terms of competitiveness has to use a job migration of $\Omega(\log m)$. This bound holds even for unit-sized jobs, but in the construction each job is eligible only to two machines. Awerbuch et al. [1] give an upper bound of $O(1)$ where $O(\log m)$ migration is used. However, they required the optimal makespan to be $\Omega(\log m)$. Westbrook [10] removed this assumption and gave the same bounds on competitiveness and migration. Moreover both results hold only for unit-sized jobs. For jobs of arbitrary size Awerbuch et al. [1] give a more general result that gives a competitiveness of $O(\log m)$ using $O(\log m)$ migration.

2 The results

Let $\varepsilon > 0$ be a constant. It is well-known that the optimum makespan can be guessed using a doubling technique, cf. [2], losing a factor 4 in competitiveness. More precisely, if λ denotes the competitiveness of an online algorithm that knows the optimimum makespan, then this can be transformed in a 4λ-competitive algorithm that is not given this information. Therefore we can suppose that we know the optimal makespan denoted by OPT. We study the following algorithm.

If there is an incoming job J_t

1. assign J_t to a machine M_j, such that $j = \text{argmin}\{\ell(j) \;\; M_j \in E_t\}$
2. while there is a job $J_{t'}$ residing on a machine M_j and a machine $M_{j'} \in E_{t'}$ such that $\ell(j) \geq \ell(j') + (1+\varepsilon)OPT$
   reassign $J_{t'}$ from M_j to $M_{j'}$.

We can show the following results.

Lemma 1 *For any $\varepsilon > 0$ the amortized migration of the algorithm is bounded by $3\lambda/\varepsilon$, where λ is the competitive ratio this algorithm achieves.*

Theorem 2 *The algorithm is $O(\log(m)/\log\log(m))$-competitive.*

We also consider such special cases where we have some a priori information about the possible set of machines which can be eligible for the jobs. We consider the problems where there is hierarchy on the servers, and if a job can be assigned to a machine it can be assigned to any machine which is on a higher level in the hierarchy. The online hierarchical restricted assignment problem is considered in [4], where several more general versions (fractional, temporary jobs) are also studied.

In the line structure the hierarchy forms a line. Formally, this means, if for some job J_t we have $M_j \in E_t$, then also $M_1, \ldots, M_{j-1} \in E_t$. If we do not allow migration, then the best known algorithm is e-competitive for the unit-sized case and this is provably optimal. For jobs of arbitrary size the best known algorithm is $e+1$-competitive. We can prove the following results for the case of unit-sized jobs.

Theorem 3 *There exists a 1-competitive algorithm which migrates at most one job after the arrival of each job.*

3 Further questions

The most important question is whether is possible to achieve a constant competitive algorithm with constant migration in the case of unit sized jobs for the general machine hierarchy case. On the other hand there are many questions concerning the special machine hierarchies. We can consider the three hierarchy which is a generalization of the line hierarchy. In the tree hierarchy, the machines form a rooted tree. If a job can be assigned to a machine, then it can be also assigned to any of its ancestors in the tree. In this case the best algorithms without migration are 4-competitive (unit sized jobs) and 5-competitive (general jobs). We can also consider nested sets. Two eligible sets are disjoint or one of them contains the other. In this case no better algorithm exists than $\Omega(\log m)$ without migration.

Acknowledgement The authors are grateful to Jiri Sgall for suggesting to study the special cases of machine hierarchies.

References

[1] B. Awerbuch, Y. Azar, S. Plotkin, O. Waarts. Competitive routing of virtual circuits with unknown duration. *SODA 1994*, 321–327, 1994.

[2] Y. Azar, On-line Load Balancing. *Online Algorithms. The State of the Art*, chapter 8, 178–195, Springer, 1998.

[3] Y. Azar, J. Naor, R. Rom, The competitiveness of on-line assignments, *J. of Algorithms*, 18, 221–237, 1995.

[4] A. Bar-Noy, A. Freund, J.S. Naor, Online load balancing in a hierarchical server topology, *J. on Computing*, 31(2), 527–549, 2001.

[5] R. L. Graham, Bounds for Certain Multiprocessing Anomalies, *The Bell System Technical Journal*, 45(9), 1563–1581,1966.

[6] K. Chaudhuri, C. Daskalakis, R. D. Kleinberg, H. Lin, Online Bipartite Perfect Matching with Augmentations. *INFOCOM*, 1044–1052, 2009.

[7] K. Lee, J. Y.-T. Leung, M. L. Pinedo, Makespan Minimization in Online Scheduling with Machine Eligibility. *4OR*, 8(4), 331–364, 2010.

[8] P. Sanders, N. Sivadasan, M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2), 481–498, 2009.

[9] S. Phillips, J. Westbrook. Online Load Balancing and Network Flow. *Proc. of STOC'93*, 402–411, 1993.

[10] J. Westbrook. Algorithms - Load Balacing for Response Time. *ESA'95. LNCS* 979, 355–368, 1995.

Competitive Algorithms from Competitive Equilibria: Non-Clairvoyant Scheduling under Polyhedral Constraints *

Sungjin Im (Speaker) † Janardhan Kulkarni ‡ Kamesh Munagala §

1 Introduction

We introduce and study a general scheduling problem that we term the Packing Scheduling problem (PSP). In this problem, jobs can have different arrival times and sizes; a scheduler can process job j at rate x_j, subject to an arbitrary packing polytope $\mathcal{P}$ over the set of rates ($\mathbf{x}$) of the outstanding jobs. The PSP framework captures a variety of scheduling problems, including the classical problems of unrelated machines scheduling, broadcast scheduling, and scheduling jobs of different parallelizability. It also captures scheduling constraints arising in diverse modern environments ranging from individual computer architectures to data centers. More concretely, PSP models multidimensional resource requirements and parallelizability, as well as network bandwidth requirements found in data center scheduling.

In this work, we design *non-clairvoyant* online algorithms for PSP and its special cases – in this setting, the scheduler is unaware of the sizes of jobs. Our main result is as follows.

Theorem 1 *For the weighted completion time objective, there exists a $O(1)$-competitive non-clairvoyant scheduling algorithm for PSP.*

We show this result by a simple algorithm that has been widely studied in the context of fairness in resource allocation, dating back to Nash. This is the Proportional Fairness (PF) algorithm [8, 7, 4]. Let $\mathcal{A}_t$ denote the set of jobs alive at time t. At time t, the rates are set using the solution to the following convex program.

$$\mathbf{x}^*(t) = \text{argmax} \left\{ \sum_{j \in \mathcal{A}_t} w_j \log x_j \;\; | \;\; \mathbf{x} \in \mathcal{P} \right\}$$

To develop intuition, let's consider a special case of PSP, the multi-dimensional scheduling. In this setting, each job j is associated with resource demand vector $\mathbf{f}_j = (f_{j1}, f_{j2}, ..., f_{jM})$ so that it requires f_{jd} amount of the d^{th} resource. At each time instant,

*Supported by an award from Cisco, and by NSF grants CCF-0745761, CCF-1008065, CCF-1409130, and IIS-0964560.

†sim3@ucmerced.edu. EECS, University of California, Merced CA 95344, USA. This work was done while the author was at Duke.

‡kulkarni@cs.duke.edu. Dept. of Computer Science, Duke University, Durham NC 27708-0129, USA.

§kamesh@cs.duke.edu. Dept. of Computer Science, Duke University, Durham NC 27708-0129, USA.

the resources must be feasibly allocated among the jobs. If job j is allocated resource vector $(a_{j1}, a_{j2}, \ldots, a_{jM})$ where $a_{jd} \leq f_{jd}$, it is processed at a rate that is determined by its bottleneck resource, so that its rate is $x_j = \min_d(a_{jd}/f_{jd})$. Put differently, the rate vector $\mathbf{x}$ needs to satisfy the set of packing constraints:

$$\mathcal{P} = \Big\{ \sum_j x_j f_{jd} \leq R_d \;\; \forall d \in [M]; \qquad \mathbf{x} \leq \mathbf{1}; \qquad \mathbf{x} \geq 0 \Big\}$$

In this setting, the PF algorithm implements a *competitive equilibrium* on the jobs. Resource d has price λ_d per unit quantity. Job j has budget w_j, and sets its rate x_j so that it spends its budget, meaning that $x_j = \frac{w_j}{\sum_d \lambda_d f_{jd}}$. The convex program optimum guarantees that there exists a set of prices $\{\lambda_d\}$ so that the market clears, meaning that all resources with non-zero price are completely allocated.

In the same setting, when there is $M = 1$ dimension, the PF solution reduces to *Max-Min Fairness* – the resource is allocated to all jobs at the same rate (so that the increase in $f_j x_j$ is the same), with jobs dropping out if $x_j = 1$. Such a solution makes the smallest allocation to any job as large as possible, and is fair in that sense. Viewed this way, our result seems intuitive – a competitive non-clairvoyant algorithm needs to behave similarly to round-robin (since it needs to hedge against unknown job sizes), and the max-min fair algorithm implements this idea in a continuous sense. Therefore, fairness seems to be a requirement for competitiveness. However this intuition can be misleading – in a multi-dimensional setting, not all generalizations of max-min fairness are competitive – in particular, the popular Dominant Resource Fair (DRF) allocation and its variants [4] are $\omega(1)$ competitive. Therefore, though fairness is a requirement, not all fair algorithms are competitive.

Multidimensional scheduling is not the only application where the "right" notion of fairness is not clear. As discussed before, it is not obvious how to generalize the most intuitively fair algorithm Round Robin (or Max-Min Fairness) to unrelated machine scheduling – in [5], a couple of natural extensions of Round Robin are considered, and are shown to be $\omega(1)$-competitive for total weighted completion time. In hindsight, fairness was also a key for development of online algorithms in broadcast scheduling [2]. Hence, we find the very existence of a unified, competitive, and fair algorithm for PSP quite surprising!

We next consider the weighted flow time objective for PSP. We note that even for classical single machine scheduling, any deterministic algorithm is $\omega(1)$-competitive [1]. Further, in the unrelated machine setting, there is no online algorithm with a bounded competitive ratio [3]. Hence to obtain positive results, we appeal to speed augmentation which is a popular relaxation of the worst case analysis framework for online scheduling [6]. Here, the online algorithm is given speed $s \geq 1$, and is compared to an optimal scheduler which is given a unit speed.

Theorem 2 *For PSP, the PF algorithm is $O(\log n)$-speed, $O(\log n)$-competitive for minimizing the total weighted flow time. Furthermore, there exists an instance of PSP for which no deterministic non-clairvoyant algorithm is $O(n^{1-\epsilon})$-competitive for any constant $0 < \epsilon < 1$ with $o(\sqrt{\log n})$-speed.*

We remain it as an open problem whether there is a $O(1)$-speed $O(1)$-competitive (clairvoyant) algorithm for PSP for minimizing the total weighted flow time.

References

[1] N. Bansal and H.-L. Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *SODA*, pages 1238–1244, 2009.

[2] N. Bansal, R. Krishnaswamy, and V. Nagarajan. Better scalable algorithms for broadcast scheduling. In *ICALP (1)*, pages 324–335, 2010.

[3] N. Garg and A. Kumar. Minimizing average flow-time : Upper and lower bounds. In *FOCS*, pages 603–613, 2007.

[4] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, I. Stoica, and S. Shenker. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, 2011.

[5] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *SODA*, pages 1242–1253, 2012.

[6] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *JACM*, 47(4):617–643, 2000.

[7] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *The Journal of the Operational Research Society*, 49(3):pp. 237–252, 1998.

[8] J. Nash. The bargaining problem. *Econometrica*, 18(2):155–162, 1950.

The Sequential Price Of Anarchy for Atomic Congestion Games*

Jasper de Jong (Speaker) † Marc Uetz ‡

February 16, 2015

1 Introduction

The price of anarchy measures the costs to society due to the selfishness of players. More formally, it is a lower bound on the quality of any Nash equilibrium relative to the quality of the global optimum. However, in particular games some Nash equilibria are not realistic, therefore the price of anarchy (PoA) gives an overly pessimistic view. Instead of assuming that all players choose their strategies simultaneously, we consider games where players choose their strategies sequentially. The sequential price of anarchy ($SPoA$) is then a lower bound on the quality of any subgame perfect equilibrium of such a game relative to the quality of the global optimum. This idea was introduced in a recent paper by Paes Leme, Syrgkanis, and Tardos [6], where they indeed give examples where sequential decision making leads to better equilibria. We consider the sequential price of anarchy for Linear atomic congestion games.

Formally, the input of an instance $I \in \mathcal{I}$ consists of a finite set of resources R, a finite set of players $N = \{1, \ldots, n\}$, and for each player $i \in N$ a collection $\mathcal{A}_i$ of possible actions $A_i \subseteq R$. We say a resource $r \in R$ is chosen by player i if $r \in A_i$, where A_i is the action chosen by player i. By $A = (A_i)_{i \in N}$ we denote a possible outcome, that is, a complete profile of actions chosen by all players $i \in N$.

Each resource $r \in R$ has a constant activation cost $d_r \geq 0$ and a variable cost or weight $w_r \geq 0$ that expresses the fact that the resource gets more congested the more players choose it. The total cost of resource $r \in R$, for some outcome A, is then $f_r(A) = d_r + w_r \cdot n_r(A)$, where $n_r(A)$ denotes the number of players choosing resource r in outcome A. Given outcome A, the total cost of all resources chosen by player i is $\text{cost}_i(A) = \sum_{r \in A_i} f_r(A)$. Players aim to minimize their costs.

Note that this class of problems includes as special cases the celebrated network routing games [7] as well as load balancing (singleton congestion) games [1, 5].

Pure Nash equilibria are outcomes $(A_i)_{i \in N}$ in which no player can decrease his costs by unilaterally deviating from choosing A_i. The price of anarchy PoA [3], measures

*Research supported by CTIT (www.ctit.nl) and 3TU.AMI (www.3tu.nl), project "Mechanisms for Decentralized Service Systems".

†j.dejong-3@utwente.nl. University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

‡m.uetz@utwente.nl. University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

the quality of any Nash equilibrium relative to the quality of a globally optimal allocation, *OPT*. Here *OPT* is an outcome minimizing the total costs over all players. More specifically, for an instance I,

$$PoA(I) = \max_{NE \in \mathcal{N}(I)} \frac{\text{cost}(NE)}{\text{cost}(OPT)}, \tag{1}$$

where $\mathcal{N}(I)$ denotes the set of all Nash equilibria for instance I. The price of anarchy of a class of instances $\mathcal{I}$ is defined by $PoA(\mathcal{I}) = \sup_{I \in \mathcal{I}} PoA(I)$.

Our goal is to compare the quality of Nash equilibria to the quality of subgame perfect equilibria of an extensive form game as introduced in [4, 8]. We assume that the players choose their actions in an arbitrary, predefined order $1, 2, \ldots, n$, so that the i-th player must choose his action A_i, observing the actions of players preceding i, but of course not knowing the actions of the players succeeding him. A strategy S_i then specifies for player i the actions he chooses, one for each potential profile of actions chosen by his predecessors $1, \ldots, i-1$. We denote by S a strategy profile $(S_i)_{i \in N}$. The outcome $A = (A_i)_{i \in N}$ of a game is then the set of actions chosen by each player resulting from a given strategy profile S.

Subgame perfect equilibria, defined by Selten [8], are defined as strategy profiles that induce Nash equilibria in any subgame. Analogous to (1), the sequential price of anarchy of an instance I is defined by

$$SPoA(I) = \max_{SPE \in \mathcal{S}(I)} \frac{\text{cost}(SPE)}{\text{cost}(OPT)}, \tag{2}$$

where $\mathcal{S}(I)$ denotes the set of all subgame perfect equilibria of instance I in extensive game form. The sequential price of anarchy of a class of instances $\mathcal{I}$ is defined as in [6] by $SPoA(\mathcal{I}) = \sup_{I \in \mathcal{I}} SPoA(I)$. Throughout the paper, when the class of instances is clear from the context, we write *PoA* and *SPoA*.

2 Results

In this section we give theorems and short descriptions of the methods we use to prove them. For more extensive arguments for all theorems except for theorem 6, we refer you to our conference paper [2].

Theorem 1 *SPoA* $= 1.5$ *for atomic congestion games with two players and affine cost functions.*

We prove the theorem by considering only the relevant part of the game tree. Then, using linear inequalities, which we derive from the properties of subgame perfect actions, we lower bound the total cost in the subgame perfect equilibrium in terms of the total cost in the social optimum.

Theorem 2 *SPoA* $= 2\frac{63}{488} \approx 2.13$ *for atomic congestion games with three players and affine cost functions.*

We first use simple combinatorial arguments to argue that a worst case instance is moderate in size. Then we prove the theorem by using a linear programming approach which maximizes the *SPoA* over all instances.

Theorem 3 *Asymptotically for $n \to \infty$, SPoA $\geq 2 + \frac{1}{e} \approx 2.37$ for singleton atomic congestion games with linear cost functions.*

The proof is by a parametric set of lower bound instances.

Theorem 4 *For singleton atomic congestion games with affine cost functions, SPoA $\leq n - 1$.*

We prove the theorem by contradiction.

Theorem 5 *For symmetric singleton atomic congestion games with affine cost functions, SPoA $= 4/3$.*

We prove the theorem by showing that any *SPE* outcome of a sequential game is also an *NE* outcome of the corresponding strategic game. The theorem then follows from the fact that *PoA* $= 4/3$, as shown in [5], and a matching lower bound example.

Theorem 6 *For $n \to \infty$, the SPoA is not bounded by any constant, even in the special case of network routing games with symmetric players*

The proof is by a parametric set of lower bound instances.

References

[1] I. Caragiannis, M. Flammini, C. Kaklamanis, P. Kanellopoulos, and L. Moscardelli. Tight Bounds for Selfish and Greedy Load Balancing, In:Proceedings 33rd ICALP, Lecture Notes in Computer Science, Vol. 4051, 311-322, 2006.

[2] J. de Jong, and M. Uetz. The sequential price of anarchy for atomic congestion games, In:Proceedings WINE 2014, Lecture Notes in Computer Science, Vol. 8877, 429-434, 2014.

[3] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In:Proceedings 16th STACS, Lecture Notes in Computer Science 1563, 404-413, Springer, 1999.

[4] H. W. Kuhn. Extensive Games and the Problem of Information, Contribution to the Theory of Games, Vol. II, Annals of Math. Studies, 28, 193-216, 1953.

[5] T. Lücking, M. Mavronicolas, B. Monien, and M. Rode. A New Model for Selfish Routing, In:Proceedings STACS 2004, Lecture Notes in Computer Science, Vol. 2996, 547-558, 2004.

[6] R. Paes Leme, V. Syrgkanis, and É Tardos. The Curse of Simultaneity. In: Proceedings 3rd ITCS, 60-67, ACM, 2012.

[7] T. Roughgarden. Selfish routing with atomic players, In:Proceedings 16th SODA, 1184-1185, 2005.

[8] R. Selten. A simple model of imperfect competition, where 4 are few and 6 are many, International Journal of Game Theory, Vol. 2, 141-201, Physica Verlag 1973.

General Caching Is Hard: Even with Small Pages

Lukáš Folwarczný (Speaker) * Jiří Sgall *

1 Introduction

Caching (also known as *uniform caching* or *paging*) is a classical problem in the area of online algorithms and has been studied since 1960s. It models a two-level memory system: There is the fast memory of size C (the *cache*) and a slow but large main memory where all data reside. The problem instance comprises a sequence of requests, each demanding a page from the main memory. No cost is incurred if the requested page is present in the cache (a *cache hit*). If the requested page is not present in the cache (a *cache fault*), the page must be loaded at the fault cost of one; some page must be evicted to make space for the new one when there are already C pages in the cache. The natural objective is to evict pages in such a way that the total fault cost is minimized.

In 1990s, with the advent of World Wide Web, a generalized variant called *file caching* or simply *general caching* was studied [5, 6]. In this setting, each page p has its $\textsc{size}(p)$ and $\textsc{cost}(p)$. It costs $\textsc{cost}(p)$ to load this page into the cache and the page occupies $\textsc{size}(p)$ units of memory there. Uniform caching is the special case satisfying $\textsc{size}(p) = \textsc{cost}(p) = 1$ for every page p. Other important cases of this general model are

- the cost model (also known as *weighted caching*) where $\textsc{size}(p) = 1$ for every page;
- the bit model where $\textsc{cost}(p) = \textsc{size}(p)$ for every page;
- the fault model where $\textsc{cost}(p) = 1$ for every page.

In our work, we consider the problem of finding the optimal service in the offline variant of the problem where the whole request sequence is known in advance.

Uniform caching is solvable in polynomial time with a natural algorithm known as Belady's rule [2]. Caching in the cost model is a special case of the *k-server problem* and is also solvable in polynomial time [3]. In late 1990s, the weak NP-hardness of the bit model was known and the questions whether caching in the fault model is NP-hard and whether general caching is strongly NP-hard were raised [1].

These questions remained unanswered until 2010 when Chrobak et al. [4] showed not only that general caching is strongly NP-hard, but also that it is strongly NP-hard already in the case of the fault model as well as in the case of the bit model. However, pages of arbitrary sizes are used (e.g. as big as half of the cache) in the hardness proofs. In real-life problems, pages are likely to be small in comparison with the size of the cache. We strengthen the hardness results to cover these situations as well.

Theorem 1 *General is strongly* NP*-hard, even in the case when the page sizes are limited to* $\{1, 2, 3\}$*, in the fault model as well as in the bit model.*

*E-mails: folwar@iuuk.mff.cuni.cz and sgall@iuuk.mff.cuni.cz. Address: Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, CZ-11800 Praha 1, Czech Republic.

Caching, as described so far, requires the service to load the requested page when a fault occurs (caching under *the forced policy*). Allowing the service just to pay the fault cost and leave the page outside the cache gives us another useful variant of caching (*the optional policy*). Our result holds under either of the policies.

Chrobak et al. proved the hardness in the fault model using a reduction from an interval scheduling (or packing) problem: Each interval has its weight assigned and the objective is to choose a set of intervals of the maximum cardinality such that each point is contained in intervals of the total weight at most C. In fact, our proof can be regarded as a proof of the strong NP-hardness of this interval scheduling problem.

Complexity status of general caching with page sizes restricted to $\{1, 2\}$ remains an interesting open problem and leads to two natural questions: Can caching with page sizes $\{1, 2\}$ be solved in polynomial time, at least in the bit or fault model? Is caching with page sizes $\{1, 2\}$ (strongly) NP-hard, at least with general weights?

2 The reduction

The key ideas of our reduction (in the case of the fault model) are presented in this section. First, we consider the optional policy. We reduce the independent set to caching in the fault model with pages of sizes $\{1, 2, 3\}$. Suppose we have a graph $G = (V, E)$ with n nodes and m edges. There are n phases in the request sequence, one for each vertex. Each vertex v has its associated page p_v of size one; p_v is requested exactly twice during the request sequence, at the beginning and at the end of the v-phase.

We want the size of the maximum independent set to be equal to the number of faults saved by these vertex-pages in the optimal service. To do this, there are H groups of pages associated with each edge. In each group, there are two pages of size three, α and β, and four pages of size two, $\bar{a}, a, \bar{b}$ and b. There are $4H$ blocks associated with each edge, $2H$ of them in each phase associated with a vertex incident to this edge.

The cache size is $2mH+1$. In each block, all edges are processed in a fixed order. For each edge, a set of pages is requested according to a given scheme (a precise definition is omitted here). Requests on pages associated with one edge are depicted in Figure 1; each row corresponds to a block and the order of requests goes from top to bottom. Blocks $B_1, \ldots, B_{12}$ are the blocks associated with this edge.

The ordering of requests is designed in such a way that for sufficiently large H we are able to prove that for each edge, there is a block where one of its α-pages or β-pages is cached together with other $mH - 1$ pages of size two. Therefore, the cache is full in this case and the vertex-page corresponding to the current phase cannot be cached. This ensures that the set we obtain is indeed independent.

We also note that when the weights of the vertex-pages are set to $1/(n + 1)$, we are able to prove the strong NP-hardness using only $H = 1$. This way we obtain a short proof of the strong NP-hardness for general caching.

To obtain the hardness result also for the forced model, we reduce caching in the optional fault model to the forced model as follows: We take the instance for the optional policy, increase the cache size by three and after each request we insert a request on a new page of size three (different for each request). Forced policy on this new instance simulates the optional behavior on the original one and so the optimal fault cost on the original pages is the same in both instances.

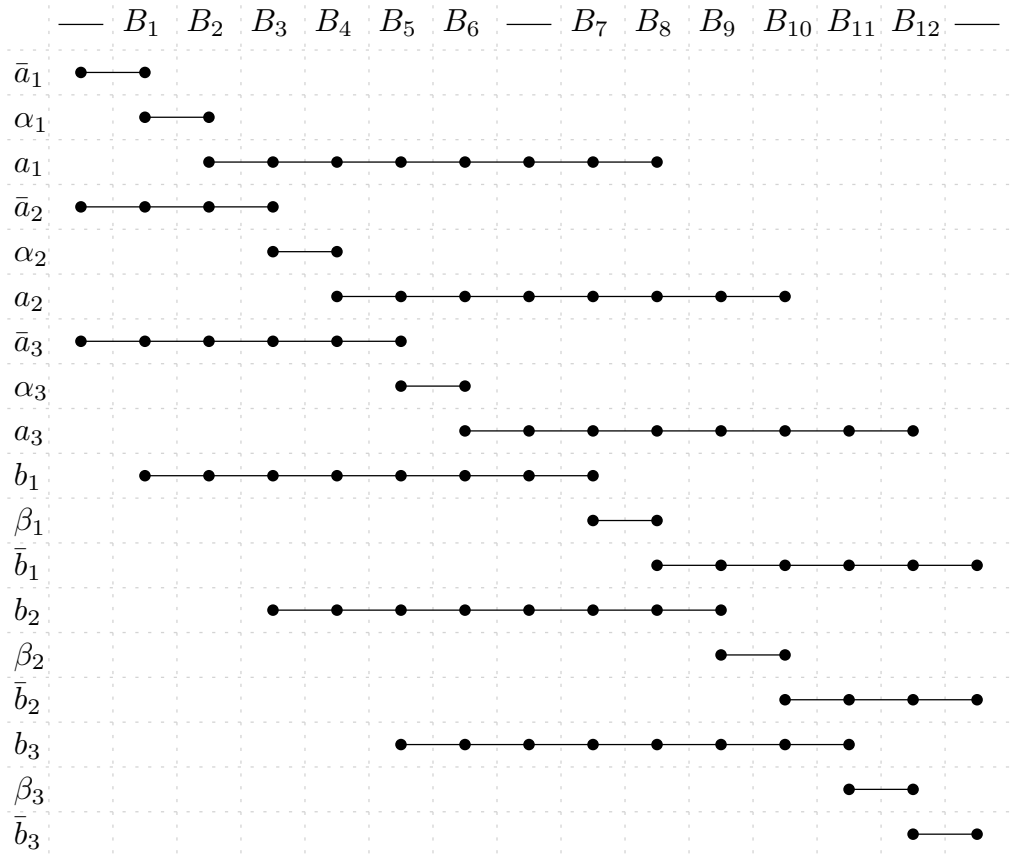


Figure 1: Requests on pages associated with one edge when $H = 3$.

References

[1] S. Albers, S. Arora, and S. Khanna, *Page replacement for general caching problems*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 31–40.

[2] L. A. Belady, *A study of replacement algorithms for a virtual-storage computer*, IBM Systems Journal, 5 (1966), pp. 78–101.

[3] M. Chrobak, H. J. Karloff, T. H. Payne, and S. Vishwanathan, *New results on server problems*, SIAM Journal on Discrete Mathematics, 4 (1991), pp. 172–181. A preliminary version appeared at SODA 1990.

[4] M. Chrobak, G. J. Woeginger, K. Makino, and H. Xu, *Caching is hard – even in the fault model*, Algorithmica, 63 (2012), pp. 781–794. A preliminary version appeared at ESA 2010.

[5] S. Irani, *Page replacement with multi-size pages and applications to web caching*, Algorithmica, 33 (2002), pp. 384–409. A preliminary version appeared at STOC 1997.

[6] N. E. Young, *On-line file caching*, Algorithmica, 33 (2002), pp. 371–383. A preliminary version appeared at SODA 1998.

Scheduling fixed tasks while maximizing the minimum idle interval via Precoloring extension on unit interval graphs

Guillerme Duvillie(Speaker) * Marin Bougeret † Rodolphe Giroudeau ‡

1 Introduction

In this paper, we consider an hotel booking management problem motivated by the following situation. Consider an hotel with m rooms. Suppose that the booking management system already accepted n_r future reservations (where each reservation r_l requires one room between a starting time $s(r_l)$ and an end date $d(r_l)$), *i.e.* the system attributed a room for each reservation, without overlap. Now, suppose that n_t new reservations are submitted. Our objective in this paper is to accept these n_t reservations while maximizing the size of the minimum idle period (where an idle period is an unoccupied room).

For example, consider the case where $m = 2$, $n_r = 3$, $s(r_1) = 0$, $d(r_1) = 1$, $s(r_2) = 0$, $d(r_2) = 2$, $s(r_3) = 0$, $d(r_3) = 0$, r_1 and r_3 are booked in room 1, and r_2 in room 0. Suppose now that three new reservations are submitted: $s(r_4) = 2$, $d(r_4) = 3$, $s(r_5) = 4$, $d(r_5) = 5$, $s(r_6) = 5$, $d(r_6) = 6$. In this case, we consider that booking r_4 and r_6 in room 0 and r_5 in room 1 is better than booking r_4 in room 1 and r_5, and r_6 in room 0. Thus, the idea of this objective function is to avoid fragmented schedules with a lot of small idle periods that could be hard to fill with future reservations.

(a) First Scenario: min idle interval = 2 (b) Second Scenario: min idle interval = 1

Figure 1: Different possible scenarios

1.1 Problem formalization

Let us now define formally the previous problem.

*duvillie@lirmm.fr. LIRMM - CNRS UMR 5506 - 161 rue Ada 34090 Montpellier, France
†bougeret@lirmm.fr. LIRMM - CNRS UMR 5506 - 161 rue Ada 34090 Montpellier, France
‡rgirou@lirmm.fr. LIRMM - CNRS UMR 5506 - 161 rue Ada 34090 Montpellier, France

Optimization Problem 1 MAX MIN IDLE

Input: *The number of machines m, a set of n_t tasks $\mathcal{T} = \{t_1, \ldots, t_{n_t}\}$ and a set of n_r reservations $\mathcal{R} = \{r_1, \ldots, r_{n_r}\}$, function $s : \mathcal{E} = \mathcal{R} \cup \mathcal{T} \mapsto \mathbb{N}$ which describes the starting times of events, a function $p : \mathcal{E} = \mathcal{R} \cup \mathcal{T} \mapsto \mathbb{N}^+$ which describes the processing time of the events, and a function $\sigma : \mathcal{R} \mapsto \{1, \ldots, m\}$ which gives the machine on which each reservation is scheduled.*

Output: *A feasible extension of σ to $\mathcal{E}$, i.e. determinate a machine for each task so that there is no overlap between any pair of event (we say that two events e_1 and e_2 of $\mathcal{E}$ overlap iff $\sigma(e_1) = \sigma(e_2)$ and $s(e_1) < s(e_2) < s(e_1) + p(e_1)$), that minimizes the size of the smallest hole, where a hole is an interval of idle time on a machine.*

Notice that as stated above, the MAX MIN IDLE problem is not correctly defined. Indeed, the output requires to schedule all the tasks $\mathcal{T}$, but deciding if all the tasks $\mathcal{T}$ can be scheduled corresponds exactly to the BASIC INTERVAL SCHEDULING WITH MACHINE AVAILABILITIES, and is proved to be NP-hard in [3]. Thus, from now on we only consider the unit version (named UNIT MAX MIN IDLE), where $p(e) = 1$ for any $e \in \mathcal{E}$, and we suppose that $\forall t$, $|\{e \in \mathcal{E} \text{ such that } s(e) = t\}| \leq m$, implying that all events can be scheduled. Finally, we also define a special case called DISJOINT UNIT MAX MIN IDLE problem where two events cannot be adjacent, *i.e.* $\forall (e_1, e_2) \in \mathcal{E}^2, \quad d(e_1) \neq s(e_2)$.

For any $p \in \mathbb{N}$, we will denote by UNIT MAX MIN IDLE(p) (resp. DISJOINT UNIT MAX MIN IDLE(p)) the problem of deciding if the optimal value is a least p.

2 Complexity and approximation results

All our results concern DISJOINT UNIT MAX MIN IDLE.

Negative result

Let us denote by PRECOL-EXT-INT (resp. PRECOL-EXT-INT$_c$) the problem of precoloring extension on a proper interval graph (resp. where all intervals have length exactly $c \in \mathbb{N}$).

We show that deciding an instance of PRECOL-EXT-INT$_c$ (which has been proved to be **NP**-hard by [2] when there is no constraint on c) reduces to deciding an instance of DISJOINT UNIT MAX MIN IDLE(c). This implies of course that DISJOINT UNIT MAX MIN IDLE is **NP**-hard as well. Consequently, any improvement of the reduction of [2] showing that PRECOL-EXT-INT$_{c_0}$ remains **NP**-hard even for a constant c_0 would imply that there is no **PTAS** for DISJOINT UNIT MAX MIN IDLE.

Positive results

Let us start with exact results. Notice first that it is easy to see that DISJOINT UNIT MAX MIN IDLE is polynomial for fixed m, using for example a dynamic programming

algorithm that parses the input from time 0 to $+\infty$ and memorizes at each time the size of the current idle interval for every machine. However, we can even get that Disjoint Unit Max Min Idle is fixed parameter tractable when parameterized by m using a reduction from Disjoint Unit Max Min Idle(p) to Precol-Ext-Int$_{p+1}$, which is proved in [1] to **FPT** when parameterized by k (the total number of colors) and the treewidth of the graph.

Concerning approximation, we provide a greedy algorithm with constant ratio for Disjoint Unit Max Min Idle. A simple example shows that this ratio is tight and that no online algorithm can provide a better ratio.

Moreover, toward getting a **PTAS** (which is still open), we also consider a classical following rounding strategy. Let us first define the notion of regularity we target.

Definition 1 (k-regular instance) *Given an integer k, we say that an instance of* Disjoint Unit Max Min Idle *is $k-$regular iff* $\forall e \in \mathcal{E},\ s(e) \equiv 0 \pmod{k}$.

Then, as claimed in the next lemma, we can round the instance while controlling the value of the optimum solution.

Lemma 2 *Given an integer ω and a positive instance I of* Disjoint Unit Max Min Idle(ω), *then $\forall\ 0 < \varepsilon < \frac{1}{2}$, we can construct a $\lceil \varepsilon\omega \rceil$-regular instance I', that is positive for integer $\omega' = \lfloor \omega(1-2\varepsilon) \rfloor$.*

Finally, as it is possible to reduce a $\lceil \varepsilon\omega \rceil$-regular instance to an instance of $PRECOL - EXT - INT_{O(\frac{1}{\epsilon})}$, we get the following result (the second item is only a restatement of the previous remark about the negative results).

Theorem 3

- *if $\forall$ fixed c, $PRECOL - EXT - INT_c$ is polynomial, then there is a* **PTAS** *for* Disjoint Unit Max Min Idle
- *if $\exists c_0$ such that $PRECOL - EXT - INT_{c_0}$ is* **NP***-hard, then there is no* **PTAS** *unless P=NP for* Disjoint Unit Max Min Idle

In conclusion, we see that the approximability of Disjoint Unit Max Min Idle is completely determined by PRECOL-EXT-INT. As it seems that the reduction of [2] cannot be easily adapted to a fixed c_0, and that PRECOL-EXT-INT(c) is polynomial for $c \leq 2$, we conjecture that $PRECOL - EXT - INT_c$ is polynomial for any fixed c.

References

[1] J. Kratochvil. Precoloring extension with fixed color bound. *Acta Mathematic University Comen*, 62:139–153, 1993.

[2] D. Marx. Precoloring extension on unit interval graphs. *Discrete Applied Mathematics*, 154(6):995–1002, 2006.

[3] C.H. Papadimitriou. Private communication to J.K. Lenstra. *April*, 25, 1982.

Approximation for scheduling on uniform processors with at most one downtime on each machine

Liliana Grigoriu (Speaker) * Donald K. Friesen †

1 Introduction

We consider the problem of non-preemptively scheduling a set of independent tasks on uniform processors, with at most one period of unavailability on each machine, in order to minimize the maximum completion time. We present a Multifit-based algorithm, LMULTIFIT, the schedules of which end within 1.5 times the optimal maximum completion time or 1.5 times the latest end of a downtime when scheduling on uniform processors with at most one downtime on each machine. Even for same-speed processors, when there is at most one downtime on each machine, it is NP-hard to obtain a schedule that ends within less than 1.5 times the end of the optimal schedule or within 1.5 times the latest end of a downtime[8].

For the special case when all unavailability periods are at the beginning at the schedule, the schedules of our algorithm end within 1.382 times the end of an optimal schedule.

For the problem with same-speed processors, when there is at most one downtime on each machine, an LPT-based algorithm the schedules of which end within the mentioned bound was given in [5]. A Multifit-based algorithm which achieves this bound when there are at most two downtimes on each machine was given in [4].

Given that all downtimes could be infinite, the NP-hardness of multiprocessor scheduling results in the NP-hardness of the problem of finding a schedule that ends within a multiple of the time needed by the optimal schedule, unless assumptions about the downtimes are made.

In [8] Scharbrodt *et al.* give a polynomial-time approximation scheme for the problem of scheduling with "fixed" jobs, that is, jobs that have to execute at certain predefined times. These are equivalent to downtimes, except that the optimal schedule also needs to execute them. The approximation scheme is for minimizing the makespan of the schedule for all the jobs, it does not consider the number of processors as a part of the input, and there can be more than one fixed job on one machine.

2 A MULTIFIT variant

A problem instance is given by a tuple $(P, T, s : P \to \mathbb{Q}, r : P \to \mathbb{N}, d : P \to \mathbb{N}, l : T \to \mathbb{N})$. $\mathbb{N}$ represents the set of natural numbers, while $\mathbb{Q}$ is the set of rational numbers.

*liliana.grigoriu@uni-siegen.de Department of Economic Sciences, Economic Computer Science and Economic Law, Siegener Mittelstandsinstitut, University of Siegen, Hölderlinstr. 3, 57068 Germany.

†friesen@cs.tamu.edu Department of Computer Science, Texas A&M University College Station, Texas, 77840-3112, USA.

Here, P is a set of processors, T is a set of tasks, and $l(X)$ denotes the length of a task X, or the time needed to execute the task on the slowest processor. For each processor $p \in P$, $d(p)$ and $r(p)$ denote the start and respectively the end of the downtime of p, and $s(p)$ is the speed factor of p, meaning that the time a task X takes to execute on p is $\frac{l(X)}{s(p)}$. Given a problem instance (P, T, s, r, d, l) and suitably chosen upper and lower bounds for the schedule's duration (for example the sum of task lengths added to the earliest end of a downtime and 0), our algorithm works as follows:

LMULTIFIT(ϵ, upper bound, lower bound) {
(1) Order all tasks in nonincreasing order of their lengths;
(2) if (upper bound − lower bound $< \epsilon$) return the saved best schedule;
(3) Set schedule deadline at $b = \frac{upper\ bound + lower\ bound}{2}$;
(4) for all $p \in P$ determine the *length* of each time slot on p by multiplying its duration with $s(p)$: $pre_p = \min(b, d(p)) * s(p)$ and $post_p = \max(0, (b - r(p)) * s(p))$;
(5) Order all time slots pre_p and $post_p$ with $p \in P$ in nondecreasing order of their lengths as determined in step (4) and record them in an array $TS[1..2|P|]$;
(6) Execute the FFD algorithm: assign tasks in nonincreasing order of their lengths in the first time slot in TS in which they fit (together with the already assigned tasks);
(7) if all tasks were scheduled {
    Save the schedule to a variable representing the best schedule found so far;
    Set upper bound = b and loop back to step (2);
    }
  Set lower bound = b and loop back to step (2);
}

When ϵ is chosen to be small enough the schedules returned by this algorithm are within 1.5 times the end of an optimal schedule or 1.5 times the latest end of a downtime. When used for the special case of non-simultaneous uniform processors, that is, when all periods of unavailability are at the beginning of the schedule, this algorithm has a worst-case approximation bound equal to the worst-case approximation bound for uniform processors (with simultaneous processing start times), which is currently known to be at most 1.382 [2]. For the special case of non-simultaneous same-speed processors the exact worst-case approximation ratio of MULTIFIT has been recently shown to be 24/19 [7].

The time complexity of our algorithm when there is at most one period of unavailability on each machine is $O(n \log n + \log(\frac{ub-lb}{\epsilon})nm)$, as there are $\log(\frac{ub-lb}{\epsilon})$ repetitions of the MULTIFIT loop, and since the task ordering takes $O(n \log n)$ time in step (1), the ordering of time slots takes $O(m \log m)$ time, and assigning of tasks to time slots takes $O(nm)$ time. Here, we assumed that $n \geq \log m$, which should be the case for all instances of interest. Considering that the steps within the loop are repeated a constant number of times results in a time complexity of $O(n \log n + nm)$.

3 Proving the worst-case bounds

While proving the upper bound results we used several argument types. We used the well-known method of assuming that there exists a minimal counterexample, that is, a problem instance for which our algorithm's schedule does not obey the upper bound to be proved, with a minimal number of processors, of tasks, of downtimes that do not

start at the beginning of the schedule, and with minimal task lengths (if in a minimal counterexample a task length is reduced the resulting instance is not a counterexample). A minimal counterexample exists whenever there is a counterexample, thus showing that it does not exist proved our theorems.

We also used the concept of a compensating processor, that is, a processor the optimal schedule of which has tasks with a greater total length (sum of task lengths) than those in the schedule produced by our algorithm, when our algorithm did not schedule the smallest task. In a minimal counterexample, when FFD fails to successfully schedule all tasks for a deadline $d \geq b * opt$ (where b is the bound to be proved), it produces a schedule containing all tasks except the smallest task. Here, *opt* stands for the maximum among the end of the optimal schedule and the latest end of a downtime. When such a FFD schedule is considered, a compensating processor exists, since the optimal schedule must include all tasks. Together with other properties of a minimal counterexample, the existence of a compensating processor in the described situation allowed us to prove upper bounds by contradiction. Weighing arguments were also used.

Proving that bounds for simultaneous uniform processors also apply to our variant of MULTIFIT for nonsimultaneous uniform processors involved showing that the worst-case approximation bounds in the first case also hold for our slightly more general variant of the algorithm. As opposed to previous MULTIFIT variants, our algorithm does not have the upper and lower bounds wired into its statements, but they can be passed on by the user as parameters. This property was required as an essential part of the proof.

References

[1] E. G. Coffman Jr., M. R. Garey and D.S.Johnson: An Application of Bin-Packing to Multiprocessor Scheduling. SIAM J. on Computing, **7**(1), 1 - 17 (1978)

[2] B.Chen:Tighter bound for MULTIFIT scheduling on uniform processors. Discrete Applied Mathematics **31**(3), 227-260 (1991)

[3] D.K. Friesen and M.A. Langston, Bounds for MULTIFIT Scheduling on Uniform Processors, SIAM Journal on Computing, **12**(1), Feb. 1983, pp. 60–69.

[4] L. Grigoriu. Multiprocessor scheduling with availability constraints. Ph.D. thesis, Texas A&M University (2010)

[5] L. Grigoriu and D. K. Friesen. Scheduling on same-speed processors with at most one downtime on each machine. Discrete Optimization, **7**(4), pp. 212 – 221 (2010)

[6] L. Grigoriu and D. K. Friesen. Scheduling on uniform processors with at most one downtime on each machine. Discrete Optimization, **17**, pp. 14 – 24 (2015)

[7] H. Hwang and K. Lim: Exact performance of MULTIFIT for scheduling on non-simultaneous machines. In: Discrete Applied Mathematics. **167**, 172 – 187 (2014)

[8] M. Scharbrodt, A. Steger and H. Weisser. Approximability of Scheduling with Fixed Jobs. Journal of Scheduling, **2**(6), pp.267 – 284 (1999)

Model and decomposition algorithm for scheduling the bottling operations line of a large winery

Alejandro F. Mac Cawley (Speaker) *

1 Introduction

In large wineries, it is normal to find a large number of different Stock Keeping Units (SKUs), that need to be bottled in a given planning period. The number of SKUs is due to the different varieties (Cabernet Sauvignon, Merlot, Sauvignon Blanc, Chardonnay, etc.), qualities of wine (varietal, reserve); bottle type, size, and label that a winery has in its portfolio. The number of combinations can be in the order of 200, which can be scheduled over multiple different bottling lines.

Typical efficiency of a wine bottling line, measured as the effective productive time versus the total available time, is in the 50% to 80% range [4]. This is due to the different types of machine setups, the number of SKUs and the constant changes in the schedules. A large amount of the non-productive time is spent on set-ups, when the line is stopped to perform configuration changes to process the next SKU. These setups can be divided into two groups: major setups, which involve a change in the type of bottle in which the wine is being bottled, and minor setups, which involve changes in the color of wine, box or the label. The minor setup changes, such as changing the color of the wine is path dependent on what was previously processed. The dependency is due to the fact that if the line starts bottling red wine and changes to white wine, the line must be thoroughly cleaned to avoid contamination of the new product. Otherwise the first batch of the bottling will be a rose wine instead of a white.

In this research we developed a new formulation of the General Lot Sizing and Scheduling Problem for Parallel production Lines (GLSPPL) with sequence dependent setup time that takes into account the particularities of the bottling problem. The proposed formulation is based on the previous work by Fleischmann and Meyr [3], later modified by Meyr [5] to capture sequence dependent times, and finally, it was expanded to capture multiple lines [6]. The model also incorporates particular aspects of the wine bottling problem such as: major/minor setups, sequence dependent setup times, crewing limitations and finally, sanitation and traceability constraints. Since the GLSPPL is an NP-Hard problem [1], this makes it a good candidate for the development and implementation of an optimization heuristic. We propose a decomposition algorithm that produces good solutions within a reasonable time-frame. Finally, using a bicriteria approach, previously used by Ehrgott and Ryan [2], we introduce a robust schedule approach.

*amac@ing.puc.cl. Departamento de Ingeniería Industrial y de Sistemas, Pontificia Universidad Católica de Chile, Vicuña Mackenna 4860, Santiago, Chile.

2 Results and Extensions

Comparing our results with the executed bottling plans of a large winery, we observed on average, total cost reductions of 27% for large instances (1 month planing horizon) when compared with the implemented bottling plans. The cost reductions originate from reductions in the required set-up times and the inventory levels. The model has been validated with two large wineries and is currently being implemented in one.

The proposed solution method uses the structure of the problem by taking advantage of the existence of major and minor set-ups, decomposing the solution process in a two step iteration process. The first step optimizes the lot-size and sequence at an aggregate family level performing an assignment of the production shifts to a specific family of bottle. In a second stage, using the previous shift family assignment, we optimize at the SKU level a lot sizing and scheduling problem with sequence dependent setup times. This solution method was tested in real life size instances and compared with the full monolithic model.

Our computational tests show that for real size instances, the decomposition approach produces solutions, with an acceptable optimal gap, in running times that range from 300 to 600 seconds. This running times can be reduced significantly if the algorithm is executed in parallel on multi-threaded computers, reductions of 5% to 60% have been reported for the parallel execution.

Finally, we present two mechanism to add robustness into the model. This is performed by adding constraints that force the model to produce a percentage of the demand at an early stage and also keep some production capacity idle in a rolling horizon. The addition of these constraints did not add significant running time into the model and could be solved with an acceptable optimal gap. The introduction of the robustness constraints produced an increment of 3% to 12% in the total costs. When the solutions were presented to the decision makers, they indicated that the benefit obtained by adding robustness outperformed the cost increment and they would implement the proposed robustness in their production process.

References

[1] Andreas Drexl and Alf Kimms. Lot sizing and scheduling: survey and extensions. *European Journal of Operational Research*, 99(2):221–235, 1997.

[2] Matthias Ehrgott and David M Ryan. Constructing robust crew schedules with bicriteria optimization. *Journal of Multi-Criteria Decision Analysis*, 11(3):139–150, 2002.

[3] Bernhard Fleischmann and Herbert Meyr. The general lotsizing and scheduling problem. *Operations-Research-Spektrum*, 19(1):11–21, 1997.

[4] Fernanda A Garcia, Martin G Marchetta, Mauricio Camargo, Laure Morel, and Raymundo Q Forradellas. A framework for measuring logistics performance in the wine industry. *International Journal of Production Economics*, 135(1):284–298, 2012.

[5] H. Meyr. Simultaneous lotsizing and scheduling by combining local search with dual reoptimization. *European Journal of Operational Research*, 120(2):311–326, 2000.

[6] H. Meyr. Simultaneous lotsizing and scheduling on parallel machines. *European Journal of Operational Research*, 139(2):277–292, 2002.

Fixed sequence integrated production and routing problems

Azeddine Cheref [*] Alessandro Agnetis [†] Christian Artigues [‡]
Jean-Charles Billaut [§]

1 Introduction

The problem considered in this paper can be described as follows. A set of n jobs must be produced by a single machine and then delivered by a single vehicle.. Each job j requires a certain *processing time* p_j on a the machine, and must be delivered after its completion to the location j. We denote by t_{ij} the transportation time from destination i to destination j. We use M to denote the depot (manufacturer) and we assume that transportation times are symmetric and satisfy the triangle inequality. The set of jobs delivered during a single round trip constitutes a *batch*. The vehicle has a capacity c which is the maximum number of jobs it can load and hence deliver in a round trip. These models for coordinating production and delivery schedules have been largely analyzed and reviewed by Chen [2], who proposed a detailed classification scheme. In this paper the production sequence is *fixed* and the jobs must be delivered in the order in which they are released, hence a production sequence also specifies the sequence in which the customers have to be reached. Since the production sequence is given, the problem consists in determining a partition of all jobs into *batches*, and each batch will then be routed according to the manufacturing sequence.

The performance measures we consider in this paper (denoted f) is the *total delivery time*, i.e., $f = \sum_{j=1}^{n} D_j$, where D_j is the time at which the job $j = 1, \ldots, n$ is delivered at its destination.

Li et al [1] proved the NP-hardness of the general problem in which the sequence is not fixed and has to be decided. In this paper we show that the problem (denoted P) is already NP-hard when the sequence is fixed and we deal with the special case in which all distances t_{ij} are identical. For this special case, we show that the problem can be efficiently solved. Finally, we briefly enounce additional results. Detailed proofs are given in [4].

[*] cheref@laas.fr
Université François-Rabelais Tours / CNRS, 64 av. J. Portalis, 37200 Tours, France
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France
[†] agnetis@dii.unisi.it
Dipartimento di Ingegneria dell'Informazione, Universitá degli Studi di Siena, via Roma 56, 53100 Siena, Italy
[‡] artigues@laas.fr
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France
[§] jean-charles.billaut@univ-tours.fr
Université François-Rabelais Tours / CNRS, 64 av. J. Portalis, 37200 Tours, France

2 Complexity

For our purposes, we introduce the EVEN-ODD PARTITION (EOP) problem. Given a set of n pairs of positive integers $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$, in which, for each i, $a_i > b_i$. Letting $K = \sum_{i=1}^{n}(a_i + b_i)$, is there a partition $(S, \bar{S})$ of the index set $I = \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} a_i + \sum_{i \in \bar{S}} b_i = K/2$? EOP is proved NP-hard in the ordinary sense by Garey et al [3]. We will actually use a slightly modified version of the problem which remains equivalent to (EOP).

MODIFIED EVEN-ODD PARTITION (MEOP). A set of n pairs of positive integers $(a_1, b_1), \ldots, (a_n, b_n)$ is given, in which, for each i, $a_i > b_i$. Letting $Q = \sum_{i=1}^{n}(a_i - b_i)$, is there a partition $(S, \bar{S})$ of the index set $I = \{1, 2, \ldots, n\}$ such that $\sum_{i \in S}(a_i - b_i) = Q/2$?

Theorem 1. *$P(\sum_{j=1}^{n} D_j)$ is NP-hard.*

Proof (Sketch). Given an instance of MEOP, we build an instance of P as follows. There are $3n + 3$ jobs. The processing times of the jobs are defined as follows: $p_1 = p_2 = p_3 = p_{3n+2} = p_{3n+3} = 0$, $p_{3n+1} = 4x_n + b_n + Q/2$ and for all $i = 1, \ldots, n-1$, $p_{3i+1} = 4x_i + b_i - 2$, $p_{3i+2} = 1$ and $p_{3i+3} = 1$. Where the x_i are defined as $x_i = (3a_i - 2b_i + 3(n-i)(a_i - b_i))/2$ for all $i = 1, \ldots, n$ and $x_{n+1} = 0$.

In the following, we refer to the set of jobs $(3(i-1)+1, 3(i-1)+2, 3i)$, $i = 1, \ldots, n$, as the *triple* T_i. For what concerns the travel times, we let:
$t_{M,3(i-1)+1} = t_{3(i-1)+1,M} = t_{M,3(i-1)+2} = t_{3(i-1)+2,M} = t_{M,3i} = t_{3i,M} = x_i \quad \forall i = 1, \ldots, n$
$t_{3(i-1)+1,3(i-1)+2} = a_i, t_{3(i-1)+2,3i} = b_i, t_{3i,3i+1} = x_i + x_{i+1} \quad \forall i = 1, \ldots, n$
$t_{M,3n+1} = t_{3n+1,M} = t_{M,3n+2} = t_{3n+2,M} = t_{M,3n+3} = t_{3n+1,3n+2} = t_{3n+2,3n+3} = 0$

Finally, vehicle capacity is $c = 2$. The problem consists in determining whether a solution exists such that the total delivery time does not exceed f^*.

$$f^* = \sum_{i=1}^{n}(3C_{3i} + 7x_i + b_i) + C_{3n+1} + C_{3n+2} + C_{3n+3} - Q/2. \tag{1}$$

Then we use the following Lemma.

Lemma 2. *If a schedule satisfying (1) exists, then there exists one satisfying the following property: for all $i = 1, \ldots, n+1$, jobs $3i$ and $3i+1$ are NOT in the same batch.*

Proof (omitted) □

Since the schedule satisfies Lemma 2 and $c = 2$, for each triple T_i, $i = 1, \ldots, n$, there are *exactly* two batches, and only two possibilities (see Figure 1), namely:

- *option A*: the first batch is $\{3(i-1)+1, 3(i-1)+2\}$ and the second is $\{3i\}$.
- *option B*: the first batch is $\{3(i-1)+1\}$ and the second is $\{3(i-1)+2, 3i\}$

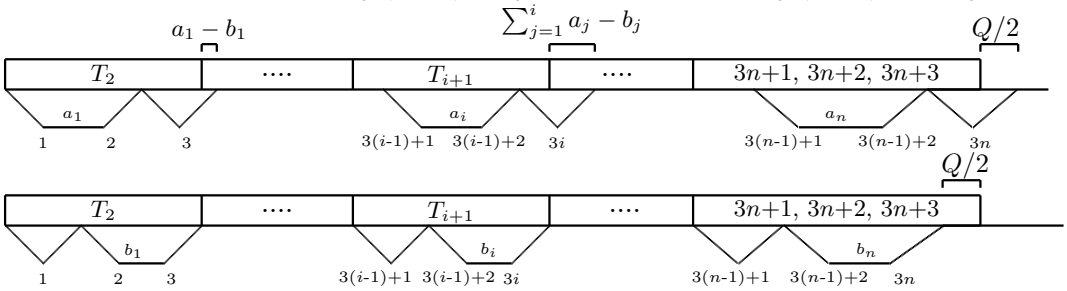


Figure 1: Round trips with options A or B.

From there, one can see that a schedule of value f^* exists if and only if EOP is a yes-instance. □

3 A special case: constant travel times

In this section we address the special case in which all travel times are identical. We start by analyzing some properties of the optimal solution then we give polynomial solution algorithm.

Clearly, every time the vehicle is back at the depot, it can either (i) restart immediately with a new batch consisting of jobs already completed, or (ii) it can wait for the completion of some jobs to be delivered. Suppose that a vehicle that departed at time t returns at the depot, and starts again at a certain time t'. Let $\mathcal{J}$ be the set of jobs released between t and t' (extremes included). The round trip starting at time t' is called *maximal* if either (i) the batch contains c jobs of $\mathcal{J}$, or (ii) it contains all jobs of $\mathcal{J}$. The next proposition gives a key feature of the optimal solutions.

Proposition 3. *There exists an optimal solution in which all round trips are maximal.*

Proof (omitted) □

Theorem 4. *Problem P with constant travel times can be solved in polynomial time.*

Proof (Sketch). Following Li et al [1], we call NSS (*non stop shipment*) a sequence of consecutive round trips during which the vehicle is never waiting at the depot, followed by a waiting time. We denote by $[i, j]$ an NSS starting at time C_i (hence, i is the last job of the first round trip of the NSS) and ending before C_j (when another NSS will start).

Starting from the proposition 3, we can construct in polynomial time our $[i, j]$ for all $i < j$. Note that, the last trip of $[i, j]$ contains a maximum number of jobs in order to finish the trip before C_j. We denote by $f(j)$ the following value of the optimal solution of the problem restricted to the first j jobs. Then, we have that recursive formula: $f(j) = \min_{\forall i<j}\{f(i) + F_{ij}\}$, where $F_{ij} = \sum_{k=i+1}^{j} D_k$ and $[i, j]$ an NSS. □

We also prove that the preemptive case in NP-hard. We study also the case where there is a fixed number of different locations. We propose pseudo-polynomial algorithms for the general case.

Acknowledgment

This work was supported by the ANR project no. ANR-13-BS02-0006-01 named Athena.

References

[1] C-L. Li, G. Vairaktarakis, C-Y. Lee (2005). Machine scheduling with deliveries to multiple customer locations, *European Journal of Operational Research*, 164 (1), pp. 39–51.

[2] Z-L. Chen (2010). Integrated production and outbound distribution scheduling: review and extensions, *Operations Research*, 58(1), pp. 130–148.

[3] M.R. Garey, R.E. Tarjan, G.T. Wilfong (1988). One processor scheduling with earliess and tardiness penalities, *Mathematics of Operations Research*, 13, pp. 330–348.

[4] A. Agnetis, C. Artigues, J-C. Billaut, A. Cheref (2015). Integrated production and routing problems: algorithms and complexity, *LAAS report.*

Heuristics for a rich tour scheduling problem in retail

Matthieu Gérard (Speaker) * François Clautiaux † Ruslan Sadykov ‡

1 Introduction

We address a multi-activity tour scheduling problem with time varying demand. The planning horizon is discretised in a set $\mathcal{T}$ of consecutive time periods of equal length (15 minutes) over a week of 7 days. The objective is to compute a valid team schedule for a fixed roster of heterogenous employees $\mathcal{E}$ in order to minimize the costs of over and under coverage of the set $\mathcal{A}$ of different production activity demands.

A feasible solution has a hierarchical structure (Figure 1): A *team schedule* consists of a set of $|\mathcal{E}|$ valid individual plannings. An *individual planning* for an employee is a set of successive day-shifts and days-off. Two consecutive day-shifts are separated by a *rest break*. A *day-shift* consists of one *timeslot* or two timeslots separated by a *lunch break*. A *day-off* represents a special day when an employee does not work. A *timeslot* is a non-empty sequence of *tasks* where different activities are carried out without interruption. Two consecutive tasks cannot be related to the same activity. A *task* is the performance of an activity over contiguous time periods. An *activity* can be either a production activity $\in \mathcal{A}$ (related to work demands) or a pause $\in \mathcal{P}$ (related to a pause policy).

We take into account constraints that we have encountered in retail real-life customer contexts. To the best of our knowledge, this work is the first attempt to combine days-off scheduling, shift scheduling, shift assignment, activity assignment, pause and lunch break assignment.

Each employee has his own set of planning constraints and each constraint has its own parameters. Different activities can be performed successively during the same timeslot depending on the employee's skills. Lunch breaks and pauses are flexible. At each level of the team schedule hierarchy, different bounded constraints have to be satisfied: the duration, the beginning and finishing time, the number of entities.

The work demand $\mathrm{DE}_{a,t}$ $(\forall a \in \mathcal{A}, \forall t \in \mathcal{T})$ is given for each 15 minutes period. Satisfying exactly the demand is not possible in most cases : thus, under/over coverages costs arise. The piecewise linear cost function distinguishes over-coverage (resp. under-coverage) from *critical* over-coverage (resp. under-coverage).

*matthieu.gerard@laposte.net. Company ASYS, 147 rue de Rennes, 75006 Paris, France — Team Dolphin, INRIA Lille Nord Europe, Université de Lille 1, France

†francois.clautiaux@math.u-bordeaux1.fr. Institut de Mathématiques de Bordeaux (UMR CNRS 5251), Université de Bordeaux, France — Team Realopt, Inria Bordeaux Sud Ouest

‡Ruslan.Sadykov@inria.fr. Institut de Mathématiques de Bordeaux (UMR CNRS 5251), Université de Bordeaux, France — Team Realopt, INRIA Bordeaux Sud Ouest

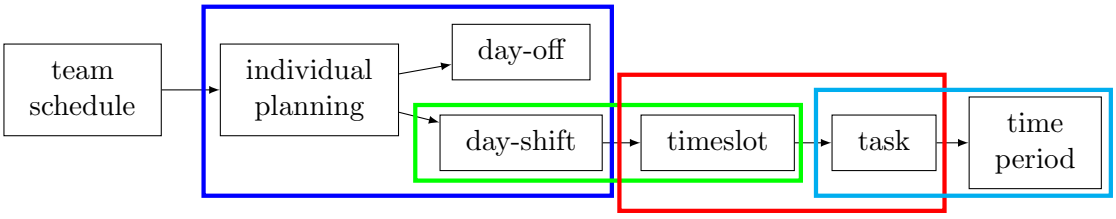


Figure 1: Hierarchical structure of a team schedule - Nested dynamic programming segmentation.

2 Solving methods

To solve this problem, four methods have been developed : a compact linear Mixed Integer Programming model, a branch-and-price like approach with a nested dynamic program to solve heuristically the subproblems, a diving heuristic and a greedy heuristic based on our subproblem solver.

The Dantzig-Wolfe decomposition is well adapted to our scheduling problem, since it consists of disjoint subproblems (one per employee) that are linked by demand constraints. In the resulting column generation method, the *subproblem* for given employee consists in designing a *valid individual planning* respecting the specific set of constraints of the employee, but disregarding the requirements dealing with the others plannings. The *master problem* combines the employee's plannings (columns) to minimize the total cost of over-coverage and under-coverage.

The subproblem for each employee can be formulated as a resource-constrained shortest path problem (RCSPP). However, the applications of state-of-the-art methods to this problem (see recent works of [1, 2, 3]) are not able to compute feasible solutions in admissible time. In fact, the specific structure of our problem leads to the following observations. There are a large number of resource constraints, but only a subset of them are active at a given node. Due to the hierarchical structure of the planning, many paths share the identical subpaths: for instance, the best day-shift for a given day is likely to be used in many non-dominated partial solutions. This led us to design an alternative more efficient heuristical approach based on a nested dynamic program (Figure 1). The column generation method might results in non-integer solution. To get a integer solution, we use a branch-and-price like approach where subproblem is solved heuristically.

For an industrial use, a fast diving heuristic has been designed based on the branch-and-price like approach (see [4]) to find a good solution with a given time limitation in minutes. However, when the time limit is set to a handful of seconds, the diving heuristic may not be able to terminate. In this case, a simple greedy heuristic based on the pricing subproblem is used to fastly find good solutions.

3 Results

The computational results, based on both real and generated tests instances, demonstrate that our methods are able to provide good quality solutions in a short computing time (Table 2). The greedy heuristic is able to find a good solution in few second. The diving heuristic is nearly optimal in few minutes of calculation (tested with time limita-

Value	$\|\mathcal{E}\|$	$\|\mathcal{A}\|$	greedy	dive120	dive1800	B&P	compact	LB_{Larg}
dst3	10	1	260	260	260	260	260	260
dst6	10	1	497	440	440	440	440	440
dst10	25	3	593	373	286	286 T	8395 T	220
dst11	25	3	540	540	540	540	540	540
dst15	40	5	709	586	580	580	589 T	580
dst18	40	5	1515	1350	1227	1215 M	-	1206

Time	$\|\mathcal{E}\|$	$\|\mathcal{A}\|$	greedy	dive120	dive1800	B&P	compact	col. gen.
dst3	10	1	0.3	0.7	0.3	0.6	4.3	0.6
dst6	10	1	0.3	5	4.1	1.2	35	1.2
dst10	25	3	0.9	126	1674	T	T	103
dst11	25	3	0.5	1.9	1.2	1.8	2493	1.7
dst15	40	5	0.9	117.3	119	9.3	T	9.3
dst18	40	5	1.5	136	1385	M	T	51.3

Figure 2: Solution values and running time (in seconds) with the different methods. "T": branch-and-price heuristic and MIP solver did not terminate after 24 hours. In this case, the solution value is the best one found; "-": MIP solver did not find any feasible solution within the time limit ; "M": branch-and-price algorithm failed due to memory issues.

tions of 120 and 1800 seconds): better results may be obtained if the time limitation is longer. Due to time limitation (24h) or memory limitations (4GB), the branch-and-price like approach may not converge and sometimes does not improve the initial solution. Similarly, the compact method has difficulties even in finding a feasible solution. Our algorithms are now embedded in a commercial software, which is already in use in a mini-mart company.

References

[1] V. Boyer, B. Gendron and L. Rousseau (2014). *: A branch-and-price algorithm for the multi-activity multi-task shift scheduling problem.* Journal of Scheduling, Volume 17, Pages 185-197.

[2] J. O. Brunner and R. Stolletz (2014). *: Stabilized branch and price with dynamic parameter updating for discontinuous tour scheduling.* Computers and Operations Research, Volume 44, Pages 137-145.

[3] M.-C. Côté, B. Gendron and L.-M. Rousseau (2013). *: Grammar-Based Column Generation for Personalized Multi-Activity Shift Scheduling.* INFORMS Journal on Computing, Number 3, Pages 461-474.

[4] C.Joncour, S. Michel, R. Sadykov, D. Sverdlov and F. Vanderbeck (2010). *: Column generation based primal heuristics.* Electronic Notes in Discrete Mathematics, Elsevier, Number 36, Pages 695-702.

Energy optimization in speed scaling models via submodular optimization

Akiyoshi Shioura * Natalia V. Shakhlevich †
Vitaly A. Strusevich (Speaker) ‡

1 Introduction

This paper is a continuation of the studies that our team has been conducting on the links between various scheduling problems and submodular optimization; see, e.g., [8].

Here we address the following speed scaling problem (SSP). The jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed either on a single machine M_1 or on parallel machines $M_1, M_2, \ldots, M_m$, where $m \geq 2$. Each job $j \in N$ is given a *release date* $r(j)$, before which it is not available, and a *deadline* $d(j)$, by which its processing must be completed, and its processing volume or size $w(j)$. The value of $w\,(j)$ can be understood as the actual processing time of job j, provided that the speed $s(j)$ of its processing is set equal to 1. In the processing of any job, *preemption* is allowed, so that the processing can be interrupted on any machine at any time and resumed later, possibly on another machine (in the case of parallel machines). The jobs are to be allocated to the machines in such a way that no job is processed on more than one machine at a time, and a machine processes at most one job at a time.

The actual processing time $p(j)$ of a job $j \in N$ depends on the speed of the processor which may change over time. In the SSP literature, the power consumption of a machine operating at speed s is proportional to s^3, or in general is described by a convex non-decreasing function $f(s)$. Notice that practically all prior publications on SSP focus on the situations when the function $f\,(s)$ is the same for all jobs. Practitioners have recently started to include in their models more realistic features of the speed scaling effects, which call for job-dependent cost functions $f_j\,(s)$; see, e.g., [2].

Given a schedule with a specified allocation of jobs to machines and fixed time intervals for processing jobs or their parts, the energy is calculated as power integrated over time. If the function f is job-independent, then due to its convexity the power is minimized if each job j is processed with a fixed speed $s\,(j) \geq 1$, which does not change during the whole processing; see, e.g., [1]. This property also holds if energy consumption functions are different for different jobs. Thus, the actual processing time of job j is equal to $p(j) = w\,(j)\,/s\,(j)$ and the total cost of processing job j is equal to

*shioura@dais.is.tohoku.ac.jp. Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan
†N.Shakhlevich@leeds.ac.uk. School of Computing, University of Leeds, Leeds LS2 9JT, U.K.
‡V.Strusevich@greenwich.ac.uk. Department of Mathematical Sciences, University of Greenwich, London SE10 9LS, U.K.

$w(j)f_j(s(j))/s(j)$, where $f_j(s(j))$ is the cost of keeping the processing speed of job j to be equal to $s(j)$ for one time unit; each function is convex non-decreasing.

In the SSP, the goal is find an assignment of speeds to jobs such that (i) the energy consumption is minimized, and (ii) a feasible schedule (with no job j processed outside the time interval $[r(j), d(j)]$) exists.

The corresponding cost function is defined either as

$$F = \sum_{j=1}^{n} w(j) f_j(s(j))/s(j), \tag{1}$$

provided that each job j is associated with an individual convex function f_j, or as

$$\Phi = \sum_{j=1}^{n} w(j) f(s(j))/s(j), \tag{2}$$

where the speed cost function f is a convex function, common to all jobs.

The paper [9] by Yao et al. provides a fundamental algorithmic technique for speed scaling for the most basic model of minimizing (2) on a single processor. It is running time, initially $O(n^3)$ has recently been reduced to $O(n^2)$; see [7]. For parallel machines, the best strongly polynomial-time algorithm to minimize (2) is due to [1]; it requires $O(n^2 h(n))$ time, where $h(n)$ is the time complexity for computing the maximum flow in a layered graph with $O(n)$ vertices.

In this work we report on the development of a unified framework that allows handling problems of minimizing a more general function (1) by submodular methods. This results into several algorithms that are faster than those previously known for special cases.

2 Approaches to Solving SSP

Rewrite the problem to minimize function F of the form (1) with the decision variables $p(j) = w(j)/s(j)$, where $p(j)$ is understood as an actual duration of job $j \in N$. The objective function F becomes

$$\hat{F} = \sum_{j=1}^{n} p(j) f_j\left(\frac{w(j)}{p(j)}\right). \tag{3}$$

We reduce the problem to a min-cost max-flow problem with a separable convex nonlinear objective function in a special bipartite network G with a single source and a single sink.

It is known (see, e.g., [3, Section 2.2]) that the values of a feasible flow on the arcs of G that leave the source form a polymatroid polyhedron, and this allows us to handle the range of problems by submodular optimization techniques.

If functions f_j are job-independent, the objective function can be written as

$$\hat{\Phi} = \sum_{j=1}^{n} p(j) f(w(j)/p(j)), \tag{4}$$

where f is a convex function, common to all jobs. A non-trivial result proved in [6] reduces the problem of minimizing (4) to minimizing a separable quadratic function

$\sum_{j=1}^{n} \frac{p(j)^2}{w(j)}$. This allows us to reduce the original problem to the parametric flow problem [4] and solve it in $O\left(n^3\right)$ time.

If the functions f_j are job-dependent, we adopt a decomposition algorithm by Fujishige [3] that minimizes a non-linear separable convex function over a base polyhedron. The most time consuming parts of any of n iterations of the decomposition process are

(i) solving a relaxed problem (a so-called simple resource allocation problem) of minimizing function (3) over a simplified base polyhedron, followed by

(ii) finding a so-called tight set, which defines how the problem at hand is decomposed into two subproblems.

In the case of parallel machines, we use network flow techniques to show that each iteration can be implemented in $O\left(n^3\right)$ time, which results in the overall time complexity of $O\left(n^4\right)$.

In the case of a single machine, the implementation of an iteration relies on solving a special scheduling problem by a modification of an algorithm developed in [5]. The overall time complexity of the resulting algorithm is $O\left(n^2\right)$.

References

[1] S. Albers, A. Antoniadis and G. Geiner, On multiprocessor speed scaling with migration, in *Proceedings of SPAA*, 2011, pp. 279–288.

[2] M. Bambagini, M. Bertogna and G. Buttazzo, On the effectiveness of energy-aware real-time scheduling algorithms on single-core platforms, in *Proceedings of ETFA*, 2014.

[3] S. Fujishige, *Submodular Functions and Optimization*, 2nd ed., Ann. Discrete Math. 58, Elsevier, Amsterdam, 2005.

[4] G. Gallo, M. D. Grigoriadis and R. E. Tarjan, A fast parametric maximum flow algorithm and applications, SIAM J. Comput., 18 (1989), pp. 30–55.

[5] D.S. Hochbaum and R. Shamir, Minimizing the number of tardy job unit under release time constraints. *Discrete Appl. Math.,* 28 (1990), pp. 45–57.

[6] K. Nagano and K. Aihara, Equivalence of convex minimization problems over base polytopes, *Japan J. Indust. Appl. Math.* 29 (2012), pp. 519–534.

[7] M. Li, F. F. Yao and H. Yuan, An $O(n^2)$ algorithm for computing optimal continuous voltage schedules, `arxiv:1408.5995v1` (2014).

[8] A. Shioura, N.V. Shakhlevich and V.A. Strusevich, A submodular optimization approach to bicriteria scheduling problems with controllable processing times on parallel machines, *SIAM J. Discrete. Math.,* 27 (2013), pp. 186–204.

[9] F.F. Yao, A.J. Demers and S. Shenker, A scheduling model for reduced CPU energy, in: *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, 1995, pp. 374–382.

Stochastic and Robust Scheduling in the Cloud*

Lin Chen † Nicole Megow ‡ Roman Rischke § Leen Stougie ¶

February 4, 2015

1 Introduction

Users of cloud computing services are offered rapid access to computing resources such as processing power, storage capacity, or network bandwidth via the Internet. Cloud providers, e.g. Amazon EC2, use different pricing options such as *on-demand* and *reserved instances*. In the reservation option, a user pays a priori a fixed amount to reserve resources in advance, whereas on-demand instances are charged on a (e.g. hourly) pay-as-used basis. Users of cloud computing services face the challenging task of choosing the best combination of pricing options when provisioning resources – in particular, if instances of computing jobs underlie uncertainty.

We propose a natural model for two-stage scheduling under uncertainty that captures such resource provisioning and scheduling problem in the cloud. We investigate the following general model for *two-stage scheduling with reservation cost*: In the first stage, we are given distributional information about scheduling scenarios, and in the second stage the actual scenario is revealed. The task is to construct a schedule for the realized scenario. Using a time unit of processing in the schedule incurs some fixed cost, independently of the used capacity (number of machines). This cost depends on the stage in which the time unit is reserved. It is low if a time slot is reserved in the first stage, not knowing the actual scenario, and significantly larger if reserved in the second stage, given full information. In the *stochastic setting*, the overall goal is to decide on reservation and scheduling such as to minimize the total expected payment (in both stages) plus scheduling cost. In *the robust setting*, the overall goal is to minimize the maximum, over all specified scenarios, of payment (in both stages) plus scheduling cost.

We focus on scheduling preemptive jobs with release dates on unrelated machines such as to minimize the total weighted completion time $\sum_j w_j C_j$ and the makespan $\max_j C_j$. The corresponding single-stage, single-scenario versions of these problems, denoted as $R\,|\,pmtn, r_j\,|\,\sum_j w_j C_j$ and $R\,|\,pmtn, r_j\,|\,C_{\max}$, are fundamental classical scheduling problems. We notice that for any scheduling problem there is a two-stage version with reservation cost. We give constant-factor approximation algorithms for the min-sum and the makespan objective in the stochastic and the robust model.

*Supported by the German Science Foundation (DFG) under contract ME 3825/1.

†lchen@math.tu-berlin.de Department of Mathematics, Technische Universität Berlin, Germany.

‡nmegow@math.tu-berlin.de Department of Mathematics, Technische Universität Berlin, Germany.

§rischke@math.tu-berlin.de Department of Mathematics, Technische Universität Berlin, Germany.

¶stougie@cwi.nl Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam & CWI, The Netherlands.

2 Stochastic Model

In the stochastic setting, we consider two models with respect to the randomness. In the *polynomial-scenario model*, the distribution over the scenario set $\mathcal{S}$ is given explicitly, i.e., each scenario $k \in \mathcal{S}$ is associated with a probability $\pi_k \in [0, 1]$ with $\sum_{k\in\mathcal{S}} \pi_k = 1$. In the *black-box model*, we have efficient access to an oracle that provides samples according to the unknown probability distribution with possibly exponentially many and dependent scenarios. Our main result is as follows.

Theorem 1 *There is an $(8 + \epsilon)$-approximation algorithm for the two-stage polynomial-scenario stochastic version of $R \,|\, pmtn, r_j \,|\, \sum_j w_j C_j$.*

The approximation result relies on a new scheduling-tailored *time-slot and job-set separation* procedure, which separates jobs into those processing exclusively on either first-stage reserved slots or second-stage reserved slots. It is inspired by [1] in which the idea of separating clients was introduced in the context of covering and network design problems. The separation in our setting is achieved through solving a generalization of the time-indexed unrelated machine scheduling LP [2] followed by an application of the slow-motion technique, proposed in [3] for min-sum single machine scheduling and extended later to open shop scheduling [4]. After separating, our rounding is applied independently to both separated instances. The two (possibly overlapping) solutions are merged to a feasible joint solution. Carefully balancing the change in reservation and scheduling cost by exploiting properties of slow-motion and α-points, the resulting procedure is proven to give the main result.

Our time-slot and job-set separation procedure is based on a general result, which is interesting on its own in the polynomial-scenario model: Given a ρ-approximation for the special case in which time-slots are reserved only in the first stage, there is an 8ρ-approximation for the two-stage model. For this special case, we give a $\rho = (3 + \epsilon)$-approximation.

Using our techniques (in a slightly simplified form) we derive the following result for the makespan objective.

Theorem 2 *There is a $7.11 + \epsilon$-approximation algorithm for the two-stage polynomial-scenario stochastic version of the makespan minimization problem $R \,|\, r_j, pmtn \,|\, C_{\max}$.*

In our most general stochastic model, the black-box model, we adopt the *Sample Average Approximation (SAA) method* proposed in [5]. It replaces the distribution on the random second-stage parameters by its empirical distribution defined by samples from it. This scenario-reduction method is known to give, under certain conditions, good approximate solutions by drawing only a polynomial number of samples and solving the resulting SAA problem instead [6, 7]. Following this framework, we can apply our algorithms to a sample average version of our problem and obtain the following results.

Theorem 3 *In the black-box model, there is a $(\rho + \epsilon)$-approximation algorithm for the two-stage stochastic variant of $R \,|\, r_j, pmtn \,|\, \sum w_j C_j$ ($\rho = 8$) and $R \,|\, r_j, pmtn \,|\, C_{\max}$ ($\rho = 7.11$), respectively.*

We notice that the work of [5, 6, 7] leads only to a first-stage reservation decision. It is not obvious in our model how to construct a good second-stage solution given

a set of slots for free from the first-stage solution. In fact, considering this problem independently from the first-stage, it is unclear if a constant approximation exists. But even when considering the two stages jointly, the difficult part is to show how a second-stage solution for some scenario (not necessarily in the sample set) can be found and bounded by the SAA solution for the sample set.

3 Robust Model

In the robust setting, we restrict to the model with an explicit description of the scenario set $\mathcal{S}$, called *discrete-scenario model*. The objective is now to minimize the worst-case total cost instead of the expected total cost.

Our approximation algorithms for the stochastic model are risk-averse, i.e., the performance guarantee holds for every scenario. Therefore, the techniques used for the stochastic model also apply to the discrete-scenario robust model. For the min-$\sum w_j C_j$ problem, certain randomized steps of our algorithm must be replaced by deterministic ones losing a factor 2 in the approximation guarantee. Such an adaptation is not needed for the robust makespan problem and we directly obtain again a $(7.11+\epsilon)$-approximation algorithm. However, the makespan problem is much easier and we provide a simple 2-approximation algorithm.

Theorem 4 *For two-stage robust scheduling with reservation cost with a discrete set of scenarios, there is a ρ-approximation algorithm for the scheduling problems $R\,|\,r_j, pmtn\,|\,\sum w_j C_j$ ($\rho = 16+\epsilon$) and $R\,|\,r_j, pmtn\,|\,C_{\max}$ ($\rho = 2$), respectively.*

References

[1] D. B. Shmoys and C. Swamy. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. J. ACM, 53(6): 978–1012, 2006.

[2] A. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. SIAM J. Discrete Math., 15(4): 450–469, 2002.

[3] A. S. Schulz and M. Skutella. Random-based scheduling: New approximations and LP lower bounds. In Proc. of APPROX and RANDOM 1997, pp. 119–133, 1997.

[4] M. Queyranne and M. Sviridenko. A $(2+\epsilon)$-approximation algorithm for the generalized preemptive open shop problem with minsum objective. J. Algorithms, 45(2): 202–212, 2002.

[5] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello. The sample average approximation method for stochastic discrete optimization. SIAM J. Optim., 12(2): 479–502, 2001.

[6] M. Charikar, C. Chekuri, and M. Pál. Sampling bounds for stochastic optimization. In Proc. of APPROX and RANDOM 2005, pp. 257–269, 2005.

[7] C. Swamy and D. B. Shmoys. Sampling-based approximation algorithms for multi-stage stochastic optimization. SIAM J. Comput., 41(4): 975–1004, 2012.

Packing While Traveling*

Sergey Polyakovskiy (Speaker) † Frank Neumann ‡

1 Introduction

In this paper, we introduce a non-linear knapsack problem that occurs when packing items along a fixed route and taking into account travel time. It is inspired by the recently introduced Traveling Thief Problem (TTP) [1] which combines the classical Traveling Salesman Problem (TSP) with the 0-1 Knapsack Problem (KP). In TTP, each city has a set of available items with weights and profits and a decision has to be made which items to pick. A selected item contributes its profit to the overall profit. However, the weight of an item leads to a higher transportation cost, and therefore has a negative impact on the overall profit. The TTP involves searching for a permutation of the cities such that each given city is visited and a packing such that the resulting profit is maximal.

Our non-linear knapsack problem uses the same cost function as the TTP, but assumes a fixed route. It deals with the problem which items to select when giving a fixed route from an origin to a destination. The precise setting is as follows. Given is a route $N = (1, 2, \ldots, n+1)$ as a sequence of $n+1$ cities where all cities are unique and distances $d_i > 0$ between two consecutive cities $(i, i+1)$, $1 \leq i \leq n$. There is a vehicle which travels through the cities of N in the order of this sequence starting its trip in the first city and ending it in the city $n+1$ as a destination point. Every city i, $1 \leq i \leq n$, contains a set of distinct items $M_i = \{e_{i1}, \ldots, e_{im_i}\}$ and we denote by $M = \bigcup_{1 \leq i \leq n} M_i$ a set of all items available at all cities. Each item $e_{ik} \in M$ has a positive integer profit p_{ik} and a weight w_{ik}. The vehicle may collect a set of items on the route such that the total weight of collected items does not exceed its capacity W. Collecting an item e_{ik} leads to a profit contribution p_{ik}, but increases the transportation cost as the weight w_{ik} slows down the vehicle. The vehicle travels along $(i, i+1)$, $1 \leq i \leq n$, with velocity $v_i \in [v_{\min}, v_{\max}]$ which depends on the weight of the items collected in the first i cities. The goal is to find a subset of M such that the difference between the profit of the selected items and the transportation cost is maximized.

*The full version of this work has appeared in the Proceedings of the 12th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming (CPAIOR 2015). This research has been supported by the ARC Discovery Project DP130104395.

†Sergey.Polyakovskiy@adelaide.edu.au. Optimisation and Logistics, School of Computer Science, The University of Adelaide, SA 5005 Australia.

‡Frank.Neumann@adelaide.edu.au. Optimisation and Logistics, School of Computer Science, The University of Adelaide, SA 5005 Australia.

2 Problem Statement

To make the problem precise, we give a nonlinear binary program formulation. The program consists of one variable x_{ik} for each item $e_{ik} \in M$ where e_{ik} is chosen iff $x_{ik} = 1$. A decision vector $X = (x_{11}, \ldots, x_{nm_n})$ defines the packing plan as a solution. If no item has been selected, the vehicle travels with its maximal velocity υ_{max}. Reaching its capacity W, it travels with minimal velocity $\upsilon_{min} > 0$. The velocity depends on the weight of the chosen items in a linear way. The travel time $t_i = \frac{d_i}{v_i}$ along $(i, i+1)$ is the ratio of the distance d_i and the current velocity

$$\upsilon_i = \upsilon_{max} - \nu \sum_{j=1}^{i} \sum_{k=1}^{m_j} w_{jk} x_{jk}$$

which is determined by the weight of the items collected in cities $1, \ldots, i$. Here, $\nu = \frac{\upsilon_{max} - \upsilon_{min}}{W}$ is a constant value defined by the input parameters. The overall transportation cost is given by the sum of the travel costs along $(i, i+1)$, $1 \leq i \leq n$, multiplied by a given rent rate $R > 0$. In summary, the problem is given by the following nonlinear binary program (NKP^c).

$$\max \sum_{i=1}^{n} \left(\sum_{k=1}^{m_i} p_{ik} x_{ik} - \frac{R d_i}{\upsilon_{max} - \nu \sum_{j=1}^{i} \sum_{k=1}^{m_j} w_{jk} x_{jk}} \right) \tag{1}$$

$$\text{s.t.} \sum_{i=1}^{n} \sum_{k=1}^{m_i} w_{ik} x_{ik} \leq W \tag{2}$$

$$x_{ik} \in \{0, 1\}, \; e_{ik} \in M$$

We also consider the unconstrained version NKP^u of NKP^c where we set $W \geq \sum_{e_{ik} \in M} w_{ik}$ such that every selection of items yields a feasible solution. Given a real value B, the decision variant of NKP^c and NKP^u has to answer the question whether the value of (1) is at least B.

3 Our Results

Firstly, we investigate constrained and unconstrained versions of the problem and show that both are $\mathcal{NP}$-hard. NKP^c is NP-hard as it is a generalization of the classical NP-hard 0-1 knapsack problem [2]. In fact, assigning zero either to the rate R or to every distance value d_i in NKP^c, we obtain KP. Our contribution is the proof that the unconstrained version NKP^u of the problem remains $\mathcal{NP}$-hard. We show this by reducing the $\mathcal{NP}$-complete *subset sum problem* (SSP) to the decision variant of NKP^u which asks whether there is a solution with objective value at least B.

Secondly, we provide a pre-processing scheme to identify items of a given instance I that can be either directly included or discarded. We distinguish between two kinds of items: *compulsory* and *unprofitable* items. We call an item *compulsory* if its inclusion in any packing plan increases the value of the objective function, and call an item *unprofitable* if its inclusion in any packing plan does not increase the value of the objective function. Therefore, an optimal solution has to include all compulsory items while all

unprofitable items can be discarded. Removing such items from the optimization process can significantly speed up the algorithms. Our pre-processing allows to decrease the number of decision variables for mixed integer programming approaches we propose next.

Thirdly, in order to solve the problems, we develop exact and approximate mixed integer programming (MIP) based approaches. Both NKP^c and NKP^u contain nonlinear terms in the objective function, and therefore are nonlinear binary programs. They belong to the specific class of fractional binary programming problems for which several efficient reformulation techniques exist to handle nonlinear terms. We follow the approach of [3] and [4] to reformulate NKP^c and NKP^u as a linear mixed 0-1 program which produces exact solutions to the problems. In addition, we introduce a set of valid inequalities in order to obtain tighter relaxations. Obtaining the optimal solution is costly in regard to running time. As an alternative, the piecewise linear approximation technique can be used. In this case, each of the non-linear terms is represented by the function $t(v) = 1/v$ which is subsequently approximated by a set of straight line segments.

Our experimental investigations performed on the benchmark set of [5] show that the proposed pre-processing scheme can significantly decrease the size of the instances making them easier for computation. Furthermore, they demonstrate the effectiveness of the MIP solutions and in particular point out that the approximate MIP approach often leads to near optimal results within far less computation time than the exact approach. Small sized instances can be solved to optimality in a reasonable time by the proposed exact approach. Larger instances can be efficiently handled by our approximate approach producing near-optimal solutions.

References

[1] M. Bonyadi, Z. Michalewicz, and L. Barone. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2013), 1037–1044, 2013.

[2] S. Martello, and P. Toth. Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, 1990.

[3] H-L. Li. A global approach for general 0-1 fractional programming. European Journal of Operational Research, 73(3):590–596, 1994.

[4] M. Tawarmalani, S. Ahmed, and N. Sahinidis. A global approach for general 0-1 fractional programming. Journal of Global Optimization, 24(4):385–416, 2002.

[5] S. Polyakovskiy, M. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. A comprehensive benchmark set and heuristics for the traveling thief problem. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014), 477–484, 2014.

Nearly tight approximability results for minimum biclique cover and partition

Parinya Chalermsook * Sandy Heydrich (Speaker) † Eugenia Holm ‡
Andreas Karrenbauer §

1 Introduction

We consider the minimum biclique cover and minimum biclique partition problems on bipartite graphs. In these problems, we are given an input graph $G = (V, E)$ and we want to compute the minimum number of complete bipartite subgraphs (bicliques) that cover all edges of G. We denote this problem by BicliqueCover. If we additionally require the bicliques to be edge disjoint, we speak of BicliquePartition.

These problems, as well as the closely connected problem of finding the biclique with the largest number of edges, has applications and connections to many other areas of computer science, such as automata and language theory [6] , security [3] , bioinformatics [9] , graph drawing [4] , single machine scheduling with precedence constraints [1] , and display optimization [7] . In most of these applications, the input graph is bipartite. The problem is NP-hard even on bipartite graphs [10], and Gruber and Holzer obtained an inapproximability result of $n^{1/3-\epsilon}$ and $m^{1/5-\epsilon}$ [6]. The problem can be solved efficiently in several graph classes like bipartite domino-free graphs and bipartite convex graphs [2]. However, no non-trivial approximation algorithms for general graphs were given so far.

In this paper, we improve the lower bound and give the first algorithms for minimum biclique cover and partition.

2 Algorithmic results

Trivially, BicliqueCover and BicliquePartition can be approximated with a factor of at most $n/2$, namely if we choose the bicliques that consist of one vertex of the smaller side of the graph together with all of its neighbors (also called *stars*).

For a better guarantee for BicliqueCover, we can split the set of (w.l.o.g.) left vertices in parts of equal size and then run an exact, super-polynomial algorithm to find

*parinya.chalermsook@mpi-inf.mpg.de. Max Planck Institute for Informatics, Campus E1 4, 66123 Saarbrücken, Germany.

†sandy.heydrich@mpi-inf.mpg.de. Max Planck Institute for Informatics, Campus E1 4, and Saarbrücken Graduate School of Computer Science, Campus E1 3, 66123 Saarbrücken, Germany.

‡eugenia.holm@uni-konstanz.de. Dept. for Computer and Information Science, University of Konstanz, Box 216, 78457 Konstanz, Germany.

§andreas.karrenbauer@mpi-inf.mpg.de. Max Planck Institute for Informatics, Campus E1 4, 66123 Saarbrücken, Germany.

the minimum biclique cover on each of the induced subgraphs. Combining the biclique covers for the subgraphs gives a biclique cover for the whole graph. If we choose the size of the subgraphs small enough, the running time of the exact algorithm becomes polynomial (w.r.t. the size of the whole graph). The approximation guarantee, on the other hand, depends inversely on this size of the subgraphs. The described approach leads to the following result:

Theorem 1 *There is an* $O\left(n/\sqrt{\log(n)}\right)$ *approximation algorithm for* BicliqueCover *that runs in polynomial time. For every constant* $c > 0$*, there is an* $O\left(n/\log^c(n)\right)$ *approximation algorithm for* BicliqueCover *that runs in quasi-polynomial time.*

For BicliquePartition, a similar approach works, giving a slightly worse approximation guarantee:

Theorem 2 *There is an* $O\left(n/\log\log(n)\right)$ *approximation algorithm for* BicliquePartition.

3 Lower bound

The main result of this paper can be stated informally as follows:

Theorem 3 (Informal) BicliqueCover *and* BicliquePartition *on bipartite graphs are (almost) as hard to approximate as graph coloring.*

The proof follows the framework of graph product techniques. Roughly speaking, this framework reduces the task of proving hardness of approximation to that of proving graph product inequalities. In our case, this amounts to bounding the quantity $bc(K_2 \times (G \cdot H))$ by some slowly growing function of $bc(K_2 \times H)$ and $bc(K_2 \times G)$, where $bc(G)$ denotes the size of the minimum biclique cover of G, $\times$ and $\cdot$ are strong and lexicographic products of graphs, respectively, and K_2 is the graph consisting of one edge. The main idea of the proof is to use an optimal vertex coloring of $\overline{G}$ together with a biclique covering of $K_2 \times H$ to suggest the biclique cover of $K_2 \times (G \cdot H)$. We note that, while we give lower bound results, the flavor of our proofs is rather algorithmic: It illustrates how one can algorithmically utilize the coloring of graph $\overline{G}$ in minimizing the biclique cover in $K_2 \times G^k$.

If we combine Theorem 3 with hardness results for graph coloring [5, 8, 11], we get the following inapproximability result for BicliqueCover and BicliquePartition:

Theorem 4 BicliqueCover *and* BicliquePartition *do not admit* $n^{1-\epsilon}$ *and* $m^{1/2-\epsilon}$ *approximation algorithms unless* P=NP*. With the stronger complexity assumption* $\mathsf{NP} \not\subseteq \mathsf{BPTIME}(2^{\mathsf{polylog}(n)})$*, we get a stronger hardness result of* $\frac{n}{2^{\log^{7/8+\epsilon}(n)}}$ *and* $\frac{\sqrt{m}}{2^{\log^{7/8+\epsilon}(m)}}$ *for any* $\epsilon > 0$*.*

For the related weighted maximum biclique problem, we find the following.

Theorem 5 *There is no polynomial time algorithm to approximate the weighted maximum biclique problem within factors of* $n^{1-\epsilon}$ *and* $m^{1/2-\epsilon}$ *for all* $\epsilon > 0$ *unless* P=NP*, or within a factor* $O\left(\frac{\min\{n,\sqrt{m}\}}{2^{\log^{7/8+\epsilon}(n)}}\right)$ *unless* $\mathsf{NP} \subseteq \mathsf{BPTIME}(2^{\mathsf{polylog}(n)})$*. This even holds for edge weights in* $\{0, 1\}$*.*

References

[1] Christoph Ambühl, Monaldo Mastrolilli, and Ola Svensson. Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 329–337. IEEE Computer Society, 2007.

[2] Jérôme Amilhastre, Marie-Catherine Vilarem, and Philippe Janssen. Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. *Discrete Applied Mathematics*, 86(23):125 – 144, 1998.

[3] Alina Ene, William G. Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert Endre Tarjan. Fast exact and heuristic methods for role minimization problems. In Indrakshi Ray and Ninghui Li, editors, *SACMAT 2008, 13th ACM Symposium on Access Control Models and Technologies, Estes Park, CO, USA, June 11-13, 2008, Proceedings*, pages 1–10. ACM, 2008.

[4] David Eppstein, Michael T. Goodrich, and Jeremy Yu Meng. Confluent layered drawings. *Algorithmica*, 47(4):439–452, 2007.

[5] Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci.*, 57(2):187–199, 1998.

[6] Hermann Gruber and Markus Holzer. Inapproximability of nondeterministic state and transition complexity assuming P≠NP. In Tero Harju, Juhani Karhumäki, and Arto Lepistö, editors, *Developments in Language Theory*, volume 4588 of *Lecture Notes in Computer Science*, pages 205–216. Springer Berlin Heidelberg, 2007.

[7] Sandy Heydrich. Approximating the minimum biclique cover problem on bipartite graphs. Master's thesis, Universität des Saarlandes, 2015.

[8] Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for MaxClique, chromatic number and Min-3Lin-Deletion. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 226–237. Springer, 2006.

[9] Dana S. Nau, George Markowsky, Max A. Woodbury, and D. Bernard Amos. A mathematical analysis of human leukocyte antigen serology. *Mathematical Biosciences*, 40(3-4):243 – 270, 1978.

[10] James Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406 – 424, 1977.

[11] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

Online Multilevel Acknowledgment with Bounded Depth

Marcin Bienkowski * Martin Böhm † Jarosław Byrka *
Marek Chrobak ‡ Christoph Dürr § Lukáš Folwarczný † Łukasz Jeż *
Nguyen Kim Thang ¶ Jiří Sgall (Speaker) § Pavel Veselý §

1 Introduction

Certain optimization problems can be formulated as aggregation problems. They typically arise when expensive resources are shared by multiple agents. Perhaps the most prominent example is the TCP Acknowledgment Problem [6]. In the area of scheduling, problems of this type are various models of broadcast scheduling and batch scheduling.

We study optimization problems where aggregation can occur at multiple levels. In our model we are given an underlying rooted tree $\mathcal{T} = (V, E, s, c)$ with vertices V, directed edges E with all edges directed toward $s \in V$, and a cost function c. Each node v except for s has a cost $c(v)$ which is the cost of transmitting from v to its parent. The cost of a set of nodes U is defined as $c(U) = \sum_{v \in U \setminus \{s\}} c(v)$.

Requests arrive at arbitrary release times at arbitrary nodes of the tree, each request has a known deadline by which it has to be transmitted from its node to s. At each time, the algorithm may schedule a transmission from some subset of vertices T such that T induces a subtree of $\mathcal{T}$ containing s; this costs $c(T)$, and the effect is that all the requests currently available at the nodes of T are served. (Note that the cost has no relation to the number of requests served.) The objective is to minimize the total cost, while meeting all the deadlines. We call this problem *Multi-Level Aggregation Problem with Deadlines* (MLAP-D).

We give an algorithm for trees of height H which is $H^2 2^H$-competitive.

Before our result, competitive algorithms were known only for very special classes of trees such as trees of depth two or paths. For trees of depth two, MLAP-D becomes equivalent to the so-called Joint Replenishment Problem (with deadlines), studied in operations research as a model of two-level delivery network with warehouses and retailers; the competitive ratio for the deadline variant of the problem is 2, see [2], and this is tight. For paths, the best published upper bound is 5, see [3]; we have an upper bound of 4 and a matching lower bound (work in progress). The offline variant is NP-hard already for two levels, the best approximation algorithm achieves ratio 2 [1].

*mbi,jby,lje@ii.uni.wroc.pl, Institute of Computer Science, University of Wrocław, Poland.
†bohm,folwar,sgall,vesely@iuuk.mff.cuni.cz, Computer Science Institute of Charles University, Prague, Czech Republic.
‡marek@cs.ucr.edu, Department of Computer Science, University of California at Riverside, USA.
§Christoph.Durr@lip6.fr, Sorbonnes Univ., LIP6, Univ. Pierre et Marie Curie, Paris, France.
¶thang@ibisc.fr, IBISC, University Evry Val d'Essonne, France.

Much of related work was done in a model with no deadlines but with waiting costs, which are either linear or arbitrary (request-dependent) functions. The objective is then to minimize the cost of shipments plus the waiting cost. The special cases studied include a single link, which is the TCP acknowledgment problem, the Joint Replenishment Problem [5, 2] and paths [4, 3]. An equivalent model for general trees was previously considered in [7, 4], but their competitive ratio depends (logarithmically) on the total costs of vertices.

Recently we have been able to generalize our result from MLAP-D to general waiting costs with a similar competitive ratio of $O(H^4 2^H)$.

2 The Algorithm for MLAP-D

Let $\mathcal{T}$ be the underlying tree of height H. We start by three simplifying assumptions:

(i) We may assume that s has a single child r. More precisely, if we have an algorithm (online or offline) for such trees, we can apply it independently to each child of s and its subtree and obtain an algorithm for a general tree, with the same performance. Under this assumption, we call s the superroot and r the root; note that r is included in every (non-trivial) transmission.

(ii) We may assume that all the deadlines are distinct (via small perturbations); this will simplify the presentation.

(iii) Given $L \geq 1$, we say that the tree is L-decreasing if for each node u except for the root and the superroot, the cost of its parent is at least $L \cdot c(u)$. We may assume that the tree is L-decreasing. This is shown by a transformation which modifies the tree so that each edge (u, v) violating the condition is changed to (u, w) for some suitable ancestor w of v; this transformation loses a factor of LH in the competitive ratio.

Intuitions. Let us first look at the optimal 2-competitive algorithm for trees of height 2. Whenever a request reaches its deadline, it transmits r and the most urgent leaves so that the total cost of leaves is about $c(r)$. This is a fairly natural strategy inspired by rent-or-buy or ski-rental problems: We have to transmit r and the leaves of cost $c(r)$ only double the cost. However, it may save us a lot, as for transmitting of a leaf v we pay only $c(v)$, while in a separate transmission we would have to pay $c(v) + c(r)$.

For three levels, we may try to iterate the algorithm. We take a set B of children of r so that $c(B)$ is about $c(r)$, preferring those more urgent (i.e., with the smallest deadlines in the subtree). Now we have two possibilities: Either take for each $v \in B$ its most urgent children of total cost $c(v)$, or take the globally most urgent children of the nodes in B of total cost $c(r)$. It is easy to see that the first possibility is not sufficient: If the most urgent leaves are under a single $v \in B$, we need to take those with cost $c(r)$, as otherwise we are not making sufficient progress. A more complicated example shows that the second possibility is not good either. However, it turns out that it is sufficient (and also within a constant factor) to take both types of leaves.

Let V_i be the nodes of $\mathcal{T}$ of depth i (the superroot has depth 0, the root r has depth 1, leaves have depth at most H). Let $V_{\leq i} = \bigcup_{j=0}^{i} V_j$. For a set of nodes X, let $X_i = X \cap V_i$ and $X_{\leq i} = X \cap V_{\leq i}$, i.e., nodes in X of depth i and of depth at most i.

Our algorithm. The algorithm is now as follows. Whenever some pending packet reaches its deadline, we transmit T constructed top-down as follows. Set $T = \{r\}$ at the beginning. Then for $i := 2, \ldots, H$ we add sufficiently many nodes of V_i. More precisely,

for every $v \in T_{\leq(i-1)}$, let Y be the set of all sons x of all nodes in $T_{i-1}(v)$ that have a pending request in their subtree (i.e., there is a request that needs to use x). Construct the set X by adding the nodes of Y one by one from the most urgent nodes until $X = Y$ or $c(Y) \geq c(v)$; add X to T. Note that since the tree is L-decreasing, the cost of each node of Y is at most $c(v)/L$ and thus $c(X) \leq (1+1/L)c(v)$. This implies that at the end $c(T_i) \leq (1+1/L)c(T_{\leq(i-1)})$ and by induction we get a bound $c(T_{\leq i}) \leq (2+1/L)^{i-1}c(r)$. In particular, $c(T) \leq (2+1/L)^{H-1}c(r)$.

Intuitively, it is clear that the algorithm cannot have a better competitive ratio than $c(T)/c(r)$: It is possible that the optimum transmits only r where the most urgent request is, while our algorithm transmits T with many nodes that turn out to be useless. A recursive charging argument shows that the ratio $c(T)/c(r)$ is achieved by the algorithm. When we set $L = H/2$, use the previous bound on $c(T)$ and the fact that the reduction to L-decreasing trees loses a factor of HL, we obtain the claimed ratio $H^2 2^H$.

Final remarks. The best lower bound for a small H is 2, which is valid already for 2 levels. The lower bound of 4 for paths implies that if H goes to infinity, the lower bound approaches 4. It remains an open question whether there exists an algorithm with a better competitive ratio, polynomial in H, or even a constant independent of H.

References

[1] L. Becchetti, A. Marchetti-Spaccamela, A. Vitaletti, P. Korteweg, M. Skutella, and L. Stougie. Latency-constrained aggregation in sensor networks. *ACM Transactions on Algorithms*, 6(1):13:1–13:20, 2009.

[2] M. Bienkowski, J. Byrka, M. Chrobak, Ł. Jeż, D. Nogneng, and J. Sgall. Better approximation bounds for the joint replenishment problem. In *Proc. of the 25th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 42–54, 2014.

[3] M. Bienkowski, J. Byrka, M. Chrobak, Ł. Jeż, J. Sgall, and G. Stachowiak. Online control message aggregation in chain networks. In *Proc. of the 13th Int. Workshop on Algorithms and Data Structures (WADS)*, volume 8037 of *Lecture Notes in Computer Science*, pages 133–145. Springer, 2013.

[4] C. Brito, E. Koutsoupias, and S. Vaya. Competitive analysis of organization networks or multicast acknowledgement: How much to wait? *Algorithmica*, 64(4):584–605, 2012.

[5] N. Buchbinder, T. Kimbrel, R. Levi, K. Makarychev, and M. Sviridenko. Online make-to-order joint replenishment model: Primal dual competitive algorithms. In *Proc. of the 19th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 952–961, 2008.

[6] D. R. Dooly, S. A. Goldman, and S. D. Scott. On-line analysis of the TCP acknowledgment delay problem. *J. of the ACM*, 48(2):243–273, 2001.

[7] S. Khanna, J. Naor, and D. Raz. Control message aggregation in group communication protocols. In *Proc. of the 29th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 135–146, 2002.

Scheduling Multipacket Frames With Frame Deadlines

Łukasz Jeż (Speaker) * Yishay Mansour † Boaz Patt-Shamir ‡

In many networking settings the ingress flows to the network has a nice (almost) periodic structure. The network should guarantee a pre-specified Quality of Service (QoS) for the flows, where one basic QoS guarantee is a deadline by which a transfer would complete. Clearly, there is uncertainty regarding the future flow arrivals. We capture this in online scheduling of information units called frames, each with a delivery deadline. Frames consist of packets, which arrive in a roughly-periodic fashion, and compete on allocation of transmission slots. A frame is deemed useful only if all its packets are delivered before its deadline. We assume the competitive analysis viewpoint.

Formally, we consider a standard scheduling model at the ingress of a link. Time is slotted, packets arrive on-line, and in each time slot at most one packet can be transmitted (meaning implicitly that we assume that all packets have the same length). The idiosyncrasies of our model are our assumptions about the arrival pattern and about the way the algorithm is rewarded for delivering packets.

Input: packets and frames. The basic entities in our model are *frames* and *packets*. Each frame f consists of $k_f \in \mathbb{N}$ packets, and has a *value* $v_f \in \mathbb{N}$. We assume that packets of frame f arrive with *periodicity* d_f and *jitter* Δ_f, namely if packet 1 of f arrives at time t, then packet $i \in \{2, \ldots, k_f\}$ arrives in the time interval $t+(i-1)d_f \pm \Delta_f$. A frame is called *perfectly periodic* if $\Delta_f = 0$. Each frame f has a *slack* $s_f \geq 1$, which determines the *deadline* of f (see "output" paragraph below). The parameters of a frame f (i.e., size k_f, value v_f, period d_f, jitter Δ_f and slack s_f) are made known to the algorithm when the first packet of f arrives. We denote the actual arrival time of the i-th packet of frame f, for $i \in \{1, \ldots, k_f\}$, by $t_i(f) \in \mathbb{N}$. The arrival time of the first packet of frame f, $t_1(f)$, is also called the arrival time of f.

We assume that the algorithm knows nothing about a frame f before its arrival, and even then, it does not know the exact arrival times of the remaining packets: let $\tau_i(f) := t_1(f) + (i-1)d_f$. Then the guarantee is that the actual arrival time satisfies that $t_i(f) \in [\tau_i(f) - \Delta_f, \tau_i(f) + \Delta_f]$ for $i > 1$.

For a given instance, we let $\Delta_{\max} = \max_f(\Delta_f)$, $s_{\max} = \max_f(s_f)$, $k_{\max} = \max_f(k_f)$, $d_{\max} = \max_f(d_f)$, and $v_{\max} = \max_f(v_f)$; we also define $k_{\min}$, $d_{\min}$, and $v_{\min}$ analogously. We assume that there is a constant $c \geq 0$ such that $\Delta_f \leq c \cdot s_f$ holds for all frames f. This is due to a simple observation that, even if all frames have identical parameters, if $s \ll \Delta$, then every on-line algorithm has competitive ratio at most $O(1/k)$.

Output: delivered frames. A *schedule* says which packet is transmitted in each time step. The *deadline* of frame f is $D_f := \tau_{k_f}(f) + \Delta_f + s_f$, and a frame f is said to be *delivered* in a given schedule if all its packets are transmitted before the frame deadline

*lje@cs.uni.wroc.pl. Eindhoven University of Technology (NL) and University of Wrocław (PL).
†mansour@tau.ac.il. Tel Aviv University (IL) and Microsoft Research Herzliya (IL).
‡boaz@eng.tau.ac.il. Tel Aviv University (IL).

(we use s_f instead of $s_f - 1$ to reduce clutter later.) Given a schedule, the value delivered by that schedule is the sum of values of frames delivered by that schedule. A schedule is called *work conserving* if it always transmits a packet if some packet is pending .

Algorithms. The duty of an algorithm is to produce a schedule for any given arrival sequence, and the goal is to maximize the sum of values of delivered frames. An algorithm is called *on-line* if its decision at any time t depends only on the arrivals and transmissions before time t. We assume that the buffer space is unbounded, which means that the only contention is for the transmission slots.

The *competitive ratio* of an algorithm A is the worst-case ratio, over all allowed arrival sequences σ, between the value delivered by A from σ and the value delivered by any (i.e., optimal off-line) schedule for σ. Note that $\rho(A) \in [0, 1]$ by definition.

Motivating examples. Consider a switch with multiple incoming video streaming flows competing for the same output link. Each flow consists of *frames*, and each frame consists of a variable number of *packets*. The video source is completely periodic, but due to compression, different frames may consist of a different number of packets. On top of that, asynchronous network transfer typically adds some jitter, so the input at the switch is only approximately periodic. In order for a frame to be useful, all its packets must be delivered before the frame's deadline (which is a short while after its last packet arrives). A frame is considered completed if all its packets are delivered before the frame's deadline, and the goal of a scheduling algorithm is to maximize the number of completed frames. Partially completed frames are considered worthless.

As another example, consider a Voice over IP (VoIP) setting. Voice calls generate samples at a relatively fast rate. Samples are wrapped in packets which are aggregated in logical frames with lower-granularity deadlines. Frame deadlines are laxer due to the tolerance of the human ear. Completed frames are reconstructed and replayed at the receiver's side; incomplete frames are discarded, resulting in audible interruption (click) of the call. Our focus is on an oversubscribed link on the path of many such calls.

As a last example, consider a database (or data center) engaged in transferring truly huge files (e.g., petabytes of data) for replication purposes. It is common in such a scenario that the transfer must be completed by a certain given deadline. Typically, the transmission of such files is done piecemeal by breaking the file into smaller units, which are transmitted periodically so as to avoid overwhelming the network resources. We are interested in scenarios where mutiple such transfers cross a common congested link.

Our Approach and Results. Our model departs from the common assumption in much of the competitive analysis literature by assuming that the arrival sequence is *not arbitrary*. Specifically, we assume that once the first packet of a frame arrives, the arrival times of the remaining packets are predictable within a given bounded jitter.

We note that if an instance is perfectly periodic and *nearly uniform*, i.e., all frames have roughly the same sizes, periods, and values, then there is a simple $\Omega(1)$-competitive algorithm. Combined with the *classify and select* technique this yields poly-logarithmic (in $k_{\max}, d_{\max}, v_{\max}$) randomized algorithms for perfectly periodic instances. The conceptual contribution of this work is identifying some interesting, important, and broader classes of instances where a *constant* competitive ratio can still be achieved, as well as relaxing the perfect periodicity assumption, i.e., handling both jitter and more stringent constraints on last packets. Technically, the main results in this paper are constant-competitive, deterministic algorithms for the following cases.

- All frames have (roughly) the same period but possibly different sizes, where the

size of the frame is the number of its packets. The frame value is its size in this case. This result can combined with the classify and select technique to yield improved polylog ratios for the general case.

- All frames have the same size but possibly different periods, and they are perfectly periodic. The value of all frames in this case is identical (say, 1).

We also consider a case similar to the former: of frames with common period, different sizes, and *unit value per frame*. We note that in this case, there is a simple randomized algorithm whose competitive ratio is logarithmic in $k_{\max}$. We show that in this case a few natural algorithms, such as Earliest Deadline First (EDF) or Shortest Remaining Processing Time (SRPT), cannot guarantee significantly better competitive ratio.

Related work. The first multipacket-frame on-line model was introduced in [4], and has since been extended by considering redundancy and hierarchically structured frames. In all these models the main difficulty is not deadlines but rather limited buffer space.

A work closest to ours is [5], which differs only in that each *packet* has its own deadline and the packet arrivals are arbitrary (i.e., not roughly periodic). It shows both lower and upper bounds on the competitive ratio that are exponential in $k_{\max}$. One can view our results as showing that adding the extra assumptions of approximately periodic packet arrivals and only whole frames having deadlines allows for significantly better competitive ratio(s).

We note that the classic preemptive job scheduling problem of maximizing (weighted) throughput on a single machine [2, 3, 1] corresponds to a special case of the problem we study in which the only restrictions are that all frames have period 1 and there is no jitter. Thus strong upper bounds (almost tight in the job scheduling problem) follow for the general setting of our problem if frame values are either unit or arbitrary [1]. However, none of the known results, neither upper nor lower bounds, apply or easily extend to special cases of our problem motivated by network applications.

References

[1] C. Dürr, Ł. Jeż, and N. K. Thang. Online scheduling of bounded length jobs to maximize throughput. *J. Scheduling*, 15(5):653–664, 2012. Also appeared in *Proc. of the 7th Workshop on Approx. and Online Algorithms* (WAOA), pp. 116–127 (2009).

[2] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000. Also appeared in *Proc. of the 36th Symp. on Foundations of Comp. Sci.* (FOCS), pp. 214–221 (1995).

[3] B. Kalyanasundaram and K. Pruhs. Maximizing job completions online. *J. Algorithms*, 49(1):63–85, 2003. Also appeared in *Proc. of the 6th European Symp. on Algorithms* (ESA), pp. 235–246 (1998).

[4] A. Kesselman, B. Patt-Shamir, and G. Scalosub. Competitive buffer management with packet dependencies. *Theor. Comput. Sci.*, 489-490:75–87, 2013. Also appeared in *23rd IEEE Int. Parallel and Distributed Processing Symp.* (IPDPS), pp. 1–12 (2009).

[5] M. Markovitch and G. Scalosub. Bounded delay scheduling with packet dependencies. In *Proc. of the IEEE INFOCOM Workshops*, pages 257–262, 2014.

Online Deadline Scheduling: Speed Augmentation Revisited*

Nicole Megow Kevin Schewior (Speaker) †

1 Introduction

We consider scheduling an instance of n jobs on m parallel machines. Each job j has a release date r_j, a processing time p_j, and a deadline d_j. A *feasible* schedule is a preemptive schedule in which each job j is processed for p_j time units within its feasible time window $[r_j, d_j]$. Moreover, an instance is called *feasible* if there exists a feasible schedule for it. It is known that there is no online algorithm, that is, an algorithm that learns about the jobs only at their respective release dates, that can guarantee to produce feasible schedules for all feasible instances [2].

As suggested by Phillips et al. [4] we perform competitive analysis using resource augmentation to evaluate online algorithms for this problem. Specifically, we consider *speed augmentation* and call an online algorithm a *speed-s algorithm* if, for all feasible instances, it outputs a feasible schedule given speed-s machines. The currently best general lower bound is due to Lam and To [3] and it states that there does not exist a speed-s algorithm for any $s < (1+\sqrt{2})/2 \approx 1.21$. They also show that an algorithm that only has access to the relative order of deadlines instead of their actual values (called deadline-ordered algorithm) cannot be a speed-s algorithm for any $s < e/(e-1) \approx 1.58$.

A very simple online algorithm is *Earliest Deadline First* (EDF) which, at any moment in time, schedules m unfinished jobs with minimum deadline. It is known [4] that EDF requires a speed of exactly $2-1/m$. The more sophisticated algorithm *Least Laxity First* (LLF) prioritizes jobs by their *laxities* $\ell_j(t) := d_j - t - p_j(t)$, where $p_j(t)$ is the remanining processing time of job j at time t, instead of their deadlines. The upper bound of $2 - 1/m$ holds also for LLF [4], and a lower bound of $\varphi = (1+\sqrt{5})/2 \approx 1.62$ was shown [1].

The currently best online algorithm in terms of speed requirement is *Full-Reduced* (FR), introduced in [3]. It requires a speed-up factor of $2 - 2/(m+1)$. An improved guarantee of $e/(e-1) \approx 1.58$ was claimed in [1], but it turned out to be false. Both algorithms are deadline-ordered algorithms and both rely on the same concept in proving the guarantee based on the so-called *Yardstick* schedule.

*Technische Universität Berlin, Institut für Mathematik, Berlin, Germany. Email: {nmegow, schewior}@math.tu-berlin.de. Supported by the German Science Foundation (DFG) under contract ME 3825/1.

†This research was supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

2 Our Contribution

We settle the open question on the exact speed-up required by LLF. Then we suggest to study the performance of online algorithms in dependence of the number of distinct release dates. Finally, we revisit the lower-bound concept of the *Yardstick* schedule which, we believe, may be the key to improved algorithms.

2.1 A new lower bound for LLF

The speed-up factor required by LLF is known to lie between the golden ratio (i.e., $(1+\sqrt{5})/2 \approx 1.62$) [1] and $2-1/m$ [4]. It seemed widely assumed that the highly dynamic algorithm LLF outperforms the fixed-priority algorithm EDF and that LLF's exact speed-up is strictly less than $2-1/m$. Somewhat surprisingly, we disprove this and show that the speed requirement in the worst case is equal for both algorithms.

Theorem 1 *For any $s < 2-1/m$, LLF is not a speed-s algorithm.*

Thus, the exact speed requirement for LLF to guarantee a feasible solution is now settled to be $2-1/m$. The key to show this is to, one after the other, almost completely block machines in LLF within a certain interval while an optimum schedule could be finished with all available jobs. More specifically, for some given $s < 2-1/m$ and $\varepsilon > 0$, we give an instance J, a time t, and a job $j \in J$ such that $\ell_j(t) = \varepsilon \cdot (d_j - t)$ holds in LLF whereas, by t, all the jobs could be finished on m unit-speed machines. The result then follows by nesting those instances and carefully choosing ε for each of them.

2.2 Relation to the number of distinct release dates

Revisiting the lower bounds of aformentioned algorithms, such as EDF [4], or the currently best known algorithm FR [3], it turns out that their worst case behavior is already achieved for a single release date. In contrast, for LLF we need a huge number of distinct release dates to achieve the new, tight lower bound. Indeed, the number has to tend towards infinity to get arbitrarily close to $2-1/m$. This raises the question if LLF outperforms all known algorithms for a sufficiently small number of release dates. In fact, for the extreme case of just one release date, it is quite easy to observe, that LLF performs best possible.

Lemma 2 *On instances with a single release date, LLF is a unit-speed algorithm.*

We strongly conjecture that, for a fixed number of release dates, LLF requires a speed strictly less than $2-1/m$.

We complement this discussion with a lower bound for any online algorithm as a function of the number of distinct release dates. Here we generalize the lower bound of 1.21 by Lam and To [3].

Theorem 3 *On instances with at most k release dates and for any*

$$s < \frac{1+\sqrt{2-\frac{1}{k}}}{2},$$

there does not exist a speed-s algorithm.

2.3 Yardstick-Based Algorithms

Lam and To [3] proposed the *Yardstick* (YS) schedule, which is a relaxed online schedule on m unit-speed machines in which all jobs (of a feasible instance) meet their deadlines, but in which jobs may run to some extent simultaneously on multiple machines (which is a relaxation and not feasible for our problem). The key is to restrict the parallelization of a job in YS to times at which it is *underworked*, which means that it has not received as much processing in YS as it could have when running continuously on a unit-speed machine from its release date on. More formally, a job j is underworked at time t if $p_j - p_j(t) < t - r_j$. The general idea of Yardstick-based algorithms is to simulate the (infeasible) YS schedule and construct online a feasible schedule meeting the finishing times in Yardstick and using machines with increased speed.

The algorithm Full-Reduced (FR) [3] uses the speed-up to outdistance YS on each job (in "full mode") to such an extent that it can, from then on (in "reduced mode"), mimic the behavior of YS on this job, even if YS parallelizes it on all machines. Its speed requirement is exactly $2 - 2/(m+1)$, improving marginally upon EDF and LLF.

The idea in YS-STR [1] is as follows: While a job is not underworked in YS, it receives the same processing in YS-STR as in YS (i.e., unit speed). The processing volume of underworked jobs which is scheduled on multiple machines in YS, needs to be stretched across a larger time interval in YS-STR but uses the full available speed. It was claimed in [1] that the particular stretching of YS-STR is a speed-$(e/(e-1))$ algorithm. However, there is an incorrect claim in the analysis of the algorithm. It was falsely assumed that the worst case happens when all jobs arrive at the same time. We show the following.

Theorem 4 *For $m = 2, 3$, YS-STR requires a speed of at least $3/2$ and $5/3$, respectively.*

We conjecture that the speed-up required by YS-STR is exactly $2 - 1/m$ for any m.

We still believe that the core idea of this algorithm is most promising but needs to be enhanced with a better stretching routine. We propose the new algorithm YS-LLF which, like YS-STR, schedules non-underworked jobs on unit-speed. In contrast to YS-STR, however, a variant of LLF handles the volume of underworked jobs on the using the remaining capacity. As for YS-STR, we can prove that the speed requirement of this algorithm for a single release date is exactly $e/(e-1)$. However, whereas YS-STR sometimes leaves machine capacity unused in spite of present jobs, this never happens in YS-LLF. In fact, we conjecture that YS-LLF is a $(e/(e-1))$-speed algorithm.

References

[1] S. Anand, N. Garg, and N. Megow, *Meeting deadlines: How much speed suffices?*, in Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP 2011), vol. 6755 of LNCS, Springer, 2011, pp. 232–243.

[2] M. Dertouzos and A. Mok, *Multiprocessor online scheduling of hard-real-time tasks*, IEEE Transactions on Software Engineering, 15 (1989), pp. 1497–1506.

[3] T. W. Lam and K.-K. To, *Trade-offs between speed and processor in hard-deadline scheduling*, in Proc. of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999, pp. 623–632.

[4] C. A. Phillips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, Algorithmica, 32 (2002), pp. 163–200.

Online Non-clairvoyant Scheduling to Simultaneously Minimize All Convex Functions

Kyle Fox [*] Janardhan Kulkarni [†] Sungjin Im [‡]
Benjamin Moseley (Speaker) [§]

1 Introduction

Scheduling a set of jobs that arrive over time on a single machine is perhaps the most basic setting considered in scheduling theory. A considerable amount of work has focused on this fundamental problem. In this setting, there are n jobs that arrive over time, and each job i requires some processing time p_i to be completed on the machine. In the *online* setting, the scheduler becomes first aware of job i at time r_i when job i is released. Note that in the online setting, it is standard to assume jobs can be *preempted*.

Generally, a client that submits a job i would like to minimize the *flow time* of the job defined as $F_i := C_i - r_i$, where C_i denotes the completion time of job i. The flow time of a job measures the amount of time the job waits to be satisfied in the system. Additionally jobs could have priorities. Here each job i is associated with a weight w_i denoting its priority. When there are multiple jobs competing for service, the scheduler needs to make scheduling decisions to optimize a global objective. A popular objectives is to minimize the total (or equivalently average) weighted flow time of all the jobs, i.e., $\sum_{i\in[n]} w_i F_i$. Another is minimizing the maximum weighted flow time of the jobs. For both objectives, it is know that no algorithm can be $O(1)$-competitive.

Due to these strong lower bounds, previous work for these objectives has appealed to the relaxed analysis model called resource augmentation [9]. For the total weighted flow time objective, it is known that the algorithm Highest-Density-First (HDF) is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive for any fixed $\epsilon > 0$ [10, 1]. For the maximum weighted flow objective, the algorithm Biggest-Weight-First (BFW) is known to be $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive [5].

These objectives have also been considered in the identical machine scheduling setting. In this setting, there are m machines that the jobs can be scheduled on. Each job can be scheduled on any machine and job i requires processing time p_i no matter which machine it is assigned to. HDF as well as several other algorithms are known to be scalable for weighted flow time [11]. For the maximum unweighted flow it is known that FIFO is $(3-2/m)$-competitive, and for weighted maximum flow time a scalable algorithm is known [2, 5].

[*] kylefox@cs.duke.edu. Duke University
[†] kulkarni@cs.duke.edu. Duke University
[‡] sim3@ucmerced.edu. University of California Merced
[§] bmoseley@wustl.edu. Washington University in St. Louis

It is common in scheduling theory that algorithms are tailored for specific scheduling settings and objective functions. For instance, FIFO is considered the best algorithm for non-clairvoyantly minimizing the maximum flow time. An algorithm that does not know the processing time of a job before completing the job is said to be *non-clairvoyant*, a highly desirable property of a scheduling in practice. The algorithm HDF is considered one of the best algorithms for minimizing total weighted flow time. One natural question that arises is what to do if a system designer wants to minimize several objective functions simultaneously. For instance, a system designer may want to optimize average quality of service, while minimizing the maximum waiting time of a job. Different algorithms have been considered for minimizing average flow time and maximum flow time, but the system designer would like to have a single algorithm that performs well for both objectives.

Motivated by this question, the general cost function objective was considered in [8]. In the general cost function problem, there is a function $g : \mathbb{R}^+ \to \mathbb{R}^+$ given, and the goal of the scheduler is to minimize $\sum_{i\in[n]} w_i g(F_i)$. One can think of $g(F_i)$ as the penalty of making job i wait F_i time steps, scaled by job i's priority (its weight w_i). This objective captures most scheduling metrics. For example, this objective function captures total weighted flow time by setting $g(x) = x$. By making g grow very quickly the objective can be designed to capture minimizing the maximum weighted flow time. As stated, one of the reasons this objective was introduced was to find an algorithm that can optimize several objectives simultaneously. If one were to design an algorithm that optimizes the general cost function g while being oblivious to g, then this algorithm would optimize *all* objective functions in this framework *simultaneously*.

In [8], the general cost function objective was considered only assuming that g is non-decreasing. This is a natural assumption since there should be no incentive for a job to wait longer. It was shown that in this case, no algorithm that is oblivious to the cost function g can be $O(1)$-competitive with speed $2 - \epsilon$ for any fixed $\epsilon > 0$. Surprisingly, it was also shown that HDF, an algorithm that is oblivious to g, is $(2+\epsilon)$-speed $O(1/\epsilon)$-competitive. This result shows that it is indeed possible to design an algorithm that optimizes most of the reasonable scheduling objectives simultaneously on a single machine. The algorithm HDF is clairvoyant. Ideally, we would like to have a non-clairvoyant algorithm for general cost functions. Further, before our work there was no known similar result in the multiple identical machines setting.

Results: In our work, we consider non-clairvoyant online scheduling to minimize the general cost function on a single machine as well as on multiple identical machines. In both the settings, we give the *first* nontrivial positive results when the online scheduler is required to be non-clairvoyant. We concentrate on cost functions g which are differentiable, non-decreasing, and convex. We assume without loss of generality that $g(0) = 0$. Note that all of the objectives discussed previously have these properties. We show the following somewhat surprising result.

Theorem 1.1 *There exists a non-clairvoyant algorithm that is $(2+\epsilon)$-speed $O(1/\epsilon)$-competitive for minimizing $\sum_{i\in[n]} w_i g(C_i - r_i)$ on a single machine for any $\epsilon > 0$, when the given cost function $g : \mathbb{R}^+ \to \mathbb{R}^+$ is differentiable, non-decreasing, and convex (g' is non-decreasing). Further, this algorithm is oblivious to g.*

We note that this result implies there is a non-clairvoyant algorithm that simultaneously minimizes all functions g. We then consider the general cost function objective on

multiple machines for the first time, and give a positive result. This algorithm is also non-clairvoyant.

Theorem 1.2 *There exists a non-clairvoyant algorithm that is $(3+\epsilon)$-speed $O(1/\epsilon)$-competitive for minimizing $\sum_{i\in[n]} w_i g(C_i - r_i)$ on multiple identical machines for any $\epsilon > 0$, when the given cost function $g : \mathbb{R}^+ \to \mathbb{R}^+$ is differentiable, non-decreasing, and convex (g' is non-decreasing). Further, this algorithm is oblivious to g.*

A conference version of these results can be found here [6].

References

[1] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *Journal of Discrete Algorithms*, 4(3):339–352, 2006.

[2] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *SODA*, pages 270–279, 1998.

[3] Carl Bussema and Eric Torng. Greedy multiprocessor server scheduling. *Oper. Res. Lett.*, 34(4):451–458, 2006.

[4] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *STOC*, pages 363–372, 2004.

[5] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Online scheduling to minimize maximum response time and maximum delay factor. *Theory of Computing*, 8(1):165–195, 2012.

[6] Kyle Fox, Sungjin Im, Janardhan Kulkarni, and Benjamin Moseley. Online non-clairvoyant scheduling to simultaneously minimize all convex functions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, pages 142–157, 2013.

[7] Kyle Fox and Benjamin Moseley. Online scheduling on identical machines using srpt. In *SODA*, pages 120–128, 2011.

[8] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Online scheduling with general cost functions. In *SODA*, pages 1254–1265, 2012.

[9] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[10] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[11] Kirk Pruhs, Jiri Sgall, and Eric Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. 2004.

Hallucination Helps: Energy Efficient Virtual Circuit Routing

Antonios Antoniadis* Sungjin Im† Ravishankar Krishnaswamy‡
Benjamin Moseley§ Viswanath Nagarajan¶ Kirk Pruhs‖ Cliff Stein**

1 Introduction

According to the US Department of Energy [1], data networks consume more than 50 billion kWH of energy per year, and a 40% reduction in wide-area network energy is plausibly achievable if network components could dynamically adjust their speed to be proportional to demand. Virtual circuit routing, in which each connection is assigned a reserved route in the network with a guaranteed bandwidth, is used by several network protocols to achieve reliable communication. In this paper we consider virtual circuit routing protocols, with an objective of minimizing energy, in a network of components that are speed scalable, and that may be shutdown when idle.

We adopt the standard models for virtual circuit routing and component energy, in particular these are the same as used in [3, 2, 4]. In the Energy Efficient Routing Problem (EERP), the input consists of an undirected multi-graph $G = (V, E)$, with $|V| = n$, $|E| = m$, and a collection of k request-pairs $\{(s_i, t_i) \mid s_i, t_i \in V \text{ and } i \in [k]\}$. The output is a path P_i, representing the virtual circuit for a unit bandwidth demand, between vertices s_i and t_i, for each request-pair $i \in [k]$. In the online version of the problem, the path P_i must be specified before later request-pairs become known to the algorithm. We assume that the speed of an edge is proportional to its flow, which is the number of paths that use that edge. We further assume that the power used by an edge with flow f is $\sigma + f^\alpha$ if $f > 0$, and that the edge is shutdown and consumes no power if it supports no flow. The objective is to minimize the aggregate power used over all the edges.

The term f^α is the dynamic power of the component as it varies with the speed, or equivalently load, of the component. Here $\alpha > 1$ is a parameter specifying the energy inefficiency of the components, as speeding up by a factor of s increases the energy used per unit computation/communication by a factor of $s^{\alpha-1}$. The parameter σ is the static power, that is the power used when the component is idle, and that can only be saved

*antoniosantoniadis@gmail.com Max-Planck-Institut fur Informatik, 66123 Saarbrcken, Germany
†sim3@ucmerced.dot.edu Electrical Engineering and Computer Science, University of California at Merced, Merced, CA 95344, USA
‡ravishankar.k@gmail.com Microsoft Reseaarch India, Bangalore, India
§bmoseley@wustl.edu Washington University in St. Louis, St. Louis, MO 63130, USA
¶viswa@umich.edu Industrial & Operations Engineering, Ann Arbor, MI 48109-2117 USA
‖kirk@cs.pitt.edu Dept. of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA.
**cliff@ieor.columbia.edu Dept. of IEOR, Columbia University, New York, NY, 10027, USA

by turning the component off. The static power is really only relevant/interesting if it is large relative to the dynamic power of routing one unit of flow, thus we will assume that $\sigma \gg 1$.

1.1 Previous Work

If the static power σ is zero, Aspens et al. [**?**] showed that the natural online greedy algorithm is $O(1)$-competitive. Gupta et al. [7] showed how to use convex duality to attain the same result. On the other hand, if the static power is very large ($\sigma \gg k^\alpha$), then the optimal solution is essentially to route all flow over a minimum cardinality Steiner forest that connects corresponding request-pairs (since this minimizes static power). The difficulty, in the general case, comes from the competing goals of minimizing static power, where it's best that flows are concentrated, and minimizing dynamic power, where it's best that the flows are spread out. Andrews et al. [3] showed that there is a limit to how well these competing demands can be balanced by showing that there is no polynomial-time algorithm with approximation ratio $o(\log^{1/4} n)$, under standard complexity theoretic assumptions. In contrast, Andrews et al. [2] showed that these competing forces can be "poly-log-balanced" by giving a polynomial-time poly-log-approximation algorithm. We think it is fair to say that the algorithm design and analysis in [2] are complicated and rely on big "hammers", namely the well-linked decomposition of Chekuri-Khanna-Shepherd [6], the construction of expanders via matchings of Khandekar-Rao-Vazirani [8], and edge-disjoint routings in well-connected graphs due to Rao-Zhou [10]. Moreover, the "poly" in the poly-log approximation is sufficiently large that it was not explicitly calculated in [2]. A critical parameter in [2] is $q = \sigma^{1/\alpha}$. If the flow on an edge is at least q, then one knows that the dynamic power on that edge is at least the static power, and thus static power can be charged to the dynamic power in the analysis. Roughly speaking, the algorithmic strategy in [2] is to aggregate the flow within groups, each containing q request-pairs, and then combining the above mentioned results [6, 8, 10] to route between groups. Bansal et al. [4] considered the case of a common source vertex s for all request-pairs, that is all $s_i = s$. In addition, they also provided hardness results for various generalizations of EERP.

1.2 The Main Results

1. A polynomial-time $O(\log^\alpha k)$-approximation algorithm for EERP. The algorithm consists of the following two stages:
Buying Stage: The first stage of the algorithm determines which edges to use (it is convenient to say that we *buy* these edges). The algorithm first buys a Steiner forest to ensure minimal connectivity. Then each request-pair, with probability $\Theta(\frac{\log k}{q})$ *hallucinates* that it wants to route q units of flow unsplittably on a path between its end-points. Any routing algorithm that is "good" for the objective of dynamic power, for example the natural greedy algorithm from [7], is then used to route this *hallucinated flow*. All edges on which hallucinated flow is routed are then bought. Note that no actual flow is routed in this stage.
Routing Stage: The second stage of the algorithm routes the flow on the edges bought in the first stage, using any algorithm that is "good" for minimizing dynamic power.

There are two main steps in the analysis. The first step is to show by randomized rounding that the dynamic power of the hallucinated flow is comparable to OPT's total

power. The second step is to show that there is a routing on the bought edges that has low dynamic power.

Overall, this improves on the results in [2] in the following ways: (a) the approximation ratio is better by many $\log^{\alpha} k$ factors, (b) the algorithm is much simpler, being the combination of simple combinatorial algorithms, and (c) the analysis is considerably simpler, with the only real "hammer" being the flow-cut gap for multicommodity flow. On the other hand, the results in [2] extend to the slightly more heterogeneous setting where the power used by each edge could include an edge-dependent constant multiplier.

2. A Randomized $O(\log^{3\alpha+1} k \cdot (\log\log k)^{2\alpha})$-competitive online algorithm for EERP. The offline algorithm rather naturally extends to an online algorithm: We buy the Steiner backbone edges using any of the known online algorithms for Steiner forest. Whether a request-pair should hallucinate is decided online by independent sampling. The online greedy algorithm from [7], can be used for routing hallucinated flow, and for routing the actual flow on the bought edges. The analysis however is considerably more involved than in the offline case.

We remark that this is the first poly-log-competitive online algorithm for EERP, and that we believe that our techniques for priority multicommodity flows and cuts will likely find further applications in the future.

References

[1] Vision and roadmap: Routing telecom and data centers toward efficient energy use, May 2009. Proceedings of Vision and Roadmap Workshop on Routing Telecom and Data Centers Toward Efficient Energy Use.

[2] Matthew Andrews, Spyridon Antonakopoulos, and Lisa Zhang. Minimum-cost network design with (dis)economies of scale. In *FOCS*, pages 585–592, 2010.

[3] Matthew Andrews, Antonio Fernández, Lisa Zhang, and Wenbo Zhao. Routing for energy minimization in the speed scaling model. In *INFOCOM*, pages 2435–2443, 2010.

[4] Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, Kirk Pruhs, and Cliff Stein. Multicast routing for energy minimization using speed scaling. In *MedAlg*, pages 37–51, 2012.

[5] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *STOC*, pages 183–192, 2005.

[6] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. *CoRR*, abs/1109.5931, 2011.

[7] Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4), 2009.

[8] Satish Rao and Shuheng Zhou. Edge disjoint paths in moderately connected graphs. *SIAM J. Comput.*, 39(5):1856–1887, 2010.

Approximation algorithms for generalized routing open shop problems

Alexander Kononov (Speaker) * Anna Melnichenko †

1 Introduction

The routing open shop problem is introduced by Averbakh et al. in [1], [2]. We are given an undirected edge-weighted complete graph $G = (V, E)$, where $V = \{v_0, v_1, \ldots, v_n\}$ is the vertex set and E is the edge set. The weight τ_{ij} of edge $e_{ij} = [v_i, v_j]$ is a nonnegative integer which represents a distance between nodes v_i and v_j. Distances satisfy the triangle inequality. We have a set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ and a set of m machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_j has to be processed by each machine M_i, and this operation takes $p_{ji} \in Z^+$ time units. Each job J_j is located at vertex v_j and all machines originally stay at vertex v_0. To process these jobs the machines have to travel between the vertices (with unit speed). Thus not only the processing times of the operations, but also the travel times between jobs have to be taken into account. Operations of each job can be processed in an arbitrary order. Preemption is not allowed. Different machines cannot work on the same job simultaneously, and a machine cannot work on more than one job at a time. The makespan of a feasible schedule is the interval between the time moment when the machines start working or moving and the time moment when the last machine returns to v_0 after finishing all its operations. The goal is to minimize the makespan $C_{\max}$.

According to the standard three-field notation of scheduling problems [6], we will denote this problem as $RO||C_{\max}$ (or $ROm||C_{\max}$ for a fixed number m of machines).

The routing open shop problem is strongly NP-hard even for a single machine case as it contains the metric travelling salesman problem as a special case. Moreover, the routing open shop problem is NP-hard in ordinary sense even on a 2-node network with only two machines [2]. For the latter case a 6/5-approximation polynomial time algorithm was presented in [1]. Recently, Kononov [4] presents an FPTAS for $RO2||C_{\max}$ on a 2-node network. A 7/4-approximation algorithm for the general 2-machine case and a simple $(m + 4)/2$-approximation algorithm for the m-machine case were given in [2]. Chernykh et al. [3] present a 13/8-approximation algorithm for $RO2||C_{\max}$. Moreover, they devised an $O(\sqrt{m})$-approximation algorithm for the m-machine case $RO||C_{\max}$ using a job-aggregation idea and the greedy algorithm for the classical open shop. Yu and Zhang [7] improve the latter result and present $O(\log m(\log\log m)^{1+\epsilon})$-approximation algorithm based on the reduction of the original problem to the classical flow shop problem. This was later improved by Kononov [5] to $O(\log m)$.

*alvenko@math.nsc.ru. Sobolev Institute of Mathematics, ak Koptyuga 4, 630090, Novosibirsk, Russia.

†annamelnich@mail.ru. Novosibirsk State University, ak Koptyuga 2, 630090, Novosibirsk, Russia.

2 Our results

We consider the following generalization of the routing open shop problem. We suppose that a given subset of jobs $\mathcal{J}_j \subseteq \mathcal{J}$ must be processed on machine M_i. Thus, machine M_i does not need to visit all the vertices of G. We call this problem the generalized routing open shop problem.

According to the standard three-field notation of scheduling problems [6], we will denote this problem as $R\bar{O}||C_{\max}$ (or $R\bar{O}m||C_{\max}$ for a fixed number m of machines).

In the introduction, we listed all approximation algorithms for different versions of the routing open shop problem. All these algorithms utilize the following property: each job has to be processed by each machine and therefore each machine must visit all the vertices of G. Thus, a hamiltonian tour in G with the minimum total weight provides a good lower bound on which all these algorithm are based. Unfortunately, it does not work for the generalized routing open shop problem.

We present new approximation algorithms for different versions of the generalized routing open shop problem.

Theorem 1 *There exists an $O(m)$-approximation algorithm for $R\bar{O}||C_{\max}$.*

The distances τ_{ij} form *a tree metric* if there exists a tree $T = (V_T, E_T)$ with nodes $V_T \supseteq V$ and lengths associated with the edges, such that τ_{ij} is exactly equal to the length of the unique path in T from v_i to v_j, for all $v_i, v_j \in V$.

Theorem 2 *Consider the restriction of $R\bar{O}||C_{\max}$ in which the distances form a tree metric. There exists an $O(\sqrt{m})$-approximation algorithm for this restricted problem.*

If the number of machines is equal to two we obtain the following result.

Theorem 3 *There exists a $9/5$-approximation algorithm for $R\bar{O}2||C_{\max}$.*

Acknowledgements

The first author was partially supported by Russian Foundation for the Humanities (grant 13- 22-10002).

References

[1] I. Averbakh, O. Berman, and I. Chernykh (2005). A $\frac{6}{5}$-approximation algorithm for the two-machine routing open shop problem on a 2-node network, *Europ. J. of Oper. Res.*, Vol. 166(1), pp. 3–24.

[2] I. Averbakh, O. Berman, and I. Chernykh (2006). "The Routing Open-Shop Problem on a Network: Complexity and Approximation", *Europ. J. of Oper. Res.*, Vol. 173(2), pp. 521–539.

[3] I. Chernykh, A. Kononov A, S. Sevastyanov (2013). Efficient approximation algorithms for the routing open shop problem, *Comp. Oper. Res.*, Vol. 40, No. 3, pp. 841–847.

[4] A. Kononov (2012). About the two-machine routing open shop problem on a 2-node network, *Discrete Analysis and Operations Research*, Vol.19, No. 2, pp. 54-74 (in Russian), translated in *Journal of Applied and Industrial Mathematics*, Vol. 6, No. 3, pp. 318331.

[5] A. Kononov (2015) $O(\log m)$-approximation for the routing open shop problem, *RAIRO Oper. Res.*, Vol. 49, No 2, pp. 383–391.

[6] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (1993) Sequencing and Scheduling: Algorithms and Complexity, *Handbooks in Operations Research and Management Science, Vol. 4, Logistics of Production and Inventory*, North Holland, Amsterdam, pp. 445–522.

[7] W. Yu and G. Zhang (2011). Improved approximation algorithms for routing shop scheduling, the Proceedings of of ISAAC 2011, in: *Lecture Notes in Comp. Sci.* 7074, pp. 30–39.

Scheduling on a single machine under time-of-use tariffs

Kan Fang (Speaker) [*] Nelson A. Uhan [†] Fu Zhao [‡]
John W. Sutherland [§]

Electricity is a primary energy source for manufacturing in most countries. As a result, improving the efficiency of electricity consumption is crucial as our world faces rising energy costs, and also provides huge opportunities to save costs for electricity-intensive manufacturing enterprises. To this end, in this work, we study scheduling using the following two strategies: (1) exploiting the *variable pricing of electricity* (e.g. time-of-use electricity tariffs), in which energy prices to customers vary hourly to reflect changes in wholesale energy prices, and (2) implementing *dynamic speed scaling*, in which jobs can be processed at an arbitrary speed, with a tradeoff between speed and power demand.

The scheduling problem we study is as follows. We are given a time-of-use tariff scheme that is represented by a set of time periods $\mathcal{P} = \{1, 2, \ldots, K\}$, along with an electricity price c_k per unit energy and a duration d_k for each period $k \in \mathcal{P}$. We are also given a set of jobs $\mathcal{J} = \{1, 2, \ldots, n\}$ that need to be scheduled on a single machine: each job $j \in \mathcal{J}$ has required workload w_j and required power demand q_j. We consider two different assumptions on the machine environment:

1. *Uniform-speed machine.* In this case, all jobs must be processed at a single uniform speed. The relationship between processing time and power demand is arbitrary.

2. *Speed-scalable machine.* In this case, each job can be processed at an arbitrary speed chosen from a continuous interval. In addition, the power demand for processing a job at speed s is s^α for some constant $\alpha > 1$.

We consider both preemptive and non-preemptive processing. The objective of the problem is to find a feasible schedule that minimizes the total electricity cost, calculated based on the energy consumed over time, taking into account that each time period has a potentially different electricity price per unit energy consumed.

Starting with the work of Yao et al. [1], a great deal of research has been done on scheduling with dynamic speed scaling, especially in single machine environments; see the surveys [2, 3] for more detail. Some examples of more recent work includes [4, 5, 6] (note that this list is by no means complete). There has also been an increasing amount of work

[*] kfang@tju.edu.cn College of Management and Economics, Tianjin University, Tianjin 300072, China

[†] uhan@usna.edu. Mathematics Department, United States Naval Academy, Annapolis, MD 21402, USA

[‡] fzhao@purdue.edu. School of Mechanical Engineering, Purdue University, West Lafayette, IN 47904, USA

[§] jwsuther@purdue.edu. Environmental and Ecological Engineering, Purdue University, West Lafayette, IN 47904, USA

on energy-aware scheduling outside of computer science, in particular, in manufacturing [7, 8]. However, most of these efforts have been aimed at minimizing energy consumption. Under time-of-use tariffs, optimizing energy consumption and optimizing energy costs can be quite different. Most related to our focus here on time-of-use tariffs is [9], which studied single machine time slot scheduling problems in which each time slot has a corresponding cost, and the objective is to minimize the total time slot costs plus some traditional scheduling objective.

Contributions of this work. We investigate different variants of the scheduling problem described above. In particular:

- *Uniform-speed machine, preemptive processing.* For this case, we give a polynomial-time algorithm that determines an optimal schedule.

- *Uniform-speed machine, non-preemptive processing.* In particular, we require that each job must be processed at a single speed from its start to completion, which is stronger than the classical definition of non-preemption. For this case, we prove that the problem is strongly NP-hard, and in fact inapproximable within a constant factor, unless P = NP. We also propose an exact polynomial-time algorithm for this problem when all the jobs have the same workload and the electricity prices follow a *pyramidal* structure; that is, $c_1 < c_2 < \ldots < c_{h-1} < c_h > c_{h+1} > \ldots > c_K$ for some $h \in \mathcal{P}$.

- *Speed-scalable machine, preemptive processing.* In this case, we assume that when a job is processed in several parts, the speeds used to process each part can be different. For this case, we give some structural results on optimal schedules and propose an exact polynomial-time algorithm for this problem.

- *Speed-scalable machine, non-preemptive processing.* Here, we require that each job must be processed at a single speed from its start to completion. In this case, we show that this problem is strongly NP-hard. In addition, we present different approximation algorithms that transform optimal preemptive schedules into non-preemptive schedules and empirically test the performance of these approximation algorithms on randomly generated instances.

References

[1] F. Yao, A. Demers and S. Shenker (1995). A scheduling model for reduced CPU energy. *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 374-382.

[2] S. Irani and K.R. Pruhs (2005). Algorithmic problems in power management. *SIGACT News*, 36(2):63-76.

[3] S. Albers (2010). Energy-efficient algorithms. *Communications of the ACM*, 53(5):86-96.

[4] E. Bampis, A. Kononov, D.Letsios, G. Lucarelli and I. Nemparis (2013). From preemptive to non-preemptive speed-scaling scheduling. *19th International*

Conference on Computing and Combinatorics (COCOON2013), volume 7936 of *Lecture Notes in Computer Science*, pages 134-146.

[5] A. Antoniadis and C. Huang (2013). Non-preemptive speed scaling. *Journal of Scheduling*, 16(4):385-394.

[6] J. Kulkarni and K. Munagala (2013). Algorithms for cost-aware scheduling. *10th International Workshop on Approximation and Online Algorithms (WAOA2012)*, volume 7846 of *Lecture Notes in Computer Science*, pages 201-214.

[7] G. Mouzon, M.B. Yildirim and J. Twomey (2007). Operational methods for minimizing of energy consumption of manufactruing equipment. *International Journal of Production Research*, 45(18-19):4247-4271.

[8] K. Fang, N.A. Uhan, F. Zhao and J.W. Sutherland (2013). Flow shop scheduling with peak power consumption constraints. *Annals of Operations Research*, 206(1):115-145.

[9] G. Wan and X. Qi (2010). Scheduling with variable time slot costs. *Naval Research Logistics*, 57(2):159-171.

Scheduling with two non-unit task lengths is NP-complete

Jan Elffers * Mathijs de Weerdt †

1 Introduction

The problem considered in this paper is the non-preemptive task scheduling problem with release times and deadlines. In the three-field notation, the problem is denoted $1|r_i|L_{\max}$. In this offline scheduling problem, there is a set of tasks, each having a release time, a deadline and a processing time, that need to be scheduled on a single machine without preemption. The goal is to schedule the tasks without preemption such that no task starts before its release time and no task completes much later than its deadline. Formally, the $L_{\max}$ optimization criterion asks to minimize the maximum lateness, that is, the maximum difference in time between a task's completion time and its deadline. The problem is NP-complete in theory by an easy reduction from bin packing [3], but branch and bound algorithms work well in practice, at least on certain distributions of randomly generated instances of up to 1000 jobs [1]. The decision problem asks whether a schedule without late jobs exists. The formal definition is as follows. We require that all release times, deadlines and task lengths are integers.

Definition 1 (Single machine scheduling with release times and deadlines) *Given a set of tasks $J = \{([r_i, d_i], p_i) \mid i = 1, \ldots, n\}$, where $r_i, d_i \in \mathbb{Z}$ are the task's release time and deadline, together forming the task's* availability interval *$[r_i, d_i]$, and $p_i \in \mathbb{N}$ is the task's processing time, does there exist a* schedule*, that is, an assignment of starting times $t : \{1, \ldots, n\} \to \mathbb{R}$ to the tasks, such that $r_i \leq t(i) \leq d_i - p_i$ for all $i = 1, \ldots, n$, and the set of execution intervals $\{[t(i), t(i) + p_i) \mid i = 1, \ldots, n\}$ is pairwise disjoint?*

We study here a parameterized version of the problem, with the parameter the set of task lengths $P(J) = \{p_i \mid ([r_i, d_i], p_i) \in J\}$. The case $p_i = 1$ (unit task lengths) is solved by the greedy Earliest Due Date (EDD) algorithm; the general case $p_i = p$ (identical task lengths) is much more difficult to solve, but it can still be solved in polynomial time. The fastest known algorithm runs in $O(n \log n)$ time [2]. For multiple task lengths, the case $P = \{1, p\}$ can be solved using a linear programming formulation [4]. This approach computes a sequence of starting times such that the length-p jobs can be assigned to these starting times, with additional constraints that guarantee sufficient idle time for the unit length jobs. This approach fails even for the case $P = \{2, 4\}$, because the length-4 tasks may require starting times such that the idle time for a length-2 task consists of

*elffers@kth.se. Department of Computer Science and Communication, KTH Royal Institute of Technology, 10044 Stockholm, Sweden. Work done while a master student at TU Delft.

†m.m.deweerdt@tudelft.nl. Department of Electrical Engineering, Mathematics and Computer Science, TU Delft, P.O. Box 5, 2600 AA Delft, The Netherlands.

two isolated time units. The complexity status of the general two-task-lengths problem has been noted as an open problem [5, 4]. In this paper we prove NP-completeness of the problem for any fixed pair of non-unit task lengths. Formally, we have this result:

Theorem 2 (NP-completeness result) *For any two fixed non-unit integer task lengths $p > q > 1$, the non-preemptive single machine scheduling problem with release times and deadlines on the set of task lengths $\{p, q\}$ is NP-complete.*

2 Overview of the reduction

Our proof of NP-completeness is via an auxiliary scheduling problem AUX(p, q) that is defined for any two integer task lengths (including the case $q = 1$). We prove this problem to be polynomial-time reducible to the original problem and, in the case both task lengths are non-unit, to be NP-complete.

Our auxiliary problem is the following extension of the original problem. Next to a set of tasks J with an ordinary availability interval $[r, d]$, we add two equal size sets of "pending" tasks J_p, J_q, one per task length, with two deadlines per task. We assume the ordinary tasks to have non-negative release times and choose $t = 0$ as the common release time of all pending tasks. Each pending task has two deadlines (d', d), where $d' \leq d$. Let $N = |J_p| = |J_q|$ be the number of tasks per length. We demand the pairs of deadlines per task length to be ordered as $d'[1] \leq d[1] \leq d'[2] \leq d[2] \leq \ldots \leq d'[N] \leq d[N]$, and the long tasks to be relatively urgent compared to the short tasks: $d_p[i] \leq d'_q[i]$ for all $i = 1, \ldots, N$. The problem is to find a feasible schedule of $J \cup J_p \cup J_q$ in which at least one of the two pending tasks with index i, that is, $J_p[i]$ or $J_q[i]$, finishes by its early deadline d'. Our hardness proof of AUX(p, q) is as follows. We choose the pending jobs' deadlines such that no job can be scheduled much before its deadline; in addition, we add a small number of ordinary tasks (with release time and deadline) to the instance to impose local constraints on the pending jobs scheduled next to them. These constraints, together with the trade-off between completing the short or the long job by its early deadline, are enough to encode an NP-complete problem.

References

[1] Jacques Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42 – 47, 1982. ISSN 0377-2217.

[2] M. R. Garey, David S. Johnson, Barbara B. Simons, and Robert Endre Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM J. Comput.*, 10(2):256–269, 1981.

[3] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008. ISBN 0387789340, 9780387789347.

[4] Jiri Sgall. Open problems in throughput scheduling. In *ESA*, pages 2–11, 2012.

[5] Barbara B. Simons and Manfred K. Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM J. Comput.*, 18(4):690–710, 1989.

Sequential diagnosis of k-out-of-n systems with imperfect tests

W. Wei [*] K. Coolen (speaker) [†] F. Talla Nobibon [‡] R. Leus [§]

1 Introduction

System health monitoring for complex systems, such as a space shuttle, aircraft or integrated circuits, is crucial for reducing the likelihood of accidents due to sudden failures, and for improving system availability. It is also imperative that systems be tested before being put into operation, in order to ascertain their functionality. At manufacturing sites, for instance, products are typically inspected at the final stage of production before being shipped to the customer. Electronic equipment (smart mobile phones, laptops, etc.) in particular, which contains components from many different suppliers, has various tests executed throughout the different stages of manufacturing.

The k-out-of-n configuration has a wide range of applications in both industrial and engineering systems, such as a multi-display system in a cockpit, the multi-radiator system in a heating system, a bridge with n cables where a minimum of k cables are necessary to support the bridge, and the power grid of a city with excess power generators [3]. Consider, for example, an airplane with four engines. Furthermore, suppose that the design of the aircraft is such that at least two engines are required to function for the aircraft to remain airborne. This means that the engines are related in a k-out-of-n configuration, with $k = 2$ and $n = 4$. This is in literature sometimes also referred to as a "2-out-of-4:G" system, where G means the system works or is "good"; a k-out-of-n:G system is equivalent to an $(n-k+1)$-out-of-n:F system, which fails ("F") if at least $(n-k+1)$ components fail. The airplane is tolerant to failures in up to two engines. More generally, the k-out-of-n system configuration represents systems with built-in redundancy. A so-called *series* system is an n-out-of-n system and a *parallel* system is a 1-out-of-n system.

In sequential testing, the procedure of diagnosing a system consists in testing the components one by one in order to learn the state of the system [3]. The same diagnosis procedure may be repeated thousands of times, and so it is important to minimize the total expected costs in the long run. Additionally, besides this cost directly attributable to the test-set hardware and manpower, field return costs can be reduced by improving output quality through appropriate testing. In this article we search for an inspection *policy*, which is a set of decision rules that decide in which order to test the components,

[*] wenchao.wei@inesctec.pt. Faculty of Engineering, INESC, University of Porto, Porto.
[†] kris.coolen@ulg.ac.be. HEC - Management School, QuantOM, Université de Liège.
[‡] ftallanobibon@fedex.com. FedEx Express, Strategic Planning & Engineering, Brussels, Belgium.
[§] roel.leus@kuleuven.be. Faculty of Economics and Business, ORSTAT, KU Leuven, Leuven.

and respects specific stopping criteria. More specifically, we develop algorithms for finding optimal policies that minimize the expected testing expenses.

We focus on the case where individual component tests are *imperfect*, which means that a test can identify a component as working when in reality it is down, and vice versa; this can have severe implications. Obviously, different costs will be incurred in different ensuing situations, but these are neglected in this article: we only focus on the expected cost to assess the state of the system with a specific confidence level. Sequencing of imperfect component tests has already been studied in a number of isolated articles. The reference closest to our work is [2], who focus only on series systems.

To the best of our knowledge, however, the more general sequencing problem of imperfect tests for k-out-of-n systems has not yet been treated in the existing literature. It is the goal of this paper to fill exactly this gap.

2 Problem statement and results

A solution to the sequential testing problem with imperfect tests is a testing policy, which decides how to proceed at each stage based on diagnosis information from the preceding test outcomes. After each test, either a new component is selected or the diagnosis procedure is halted. The stopping criterion is a pre-specified threshold for the probability of a working or failing system. We thus stop testing when this threshold is reached or all components are tested (in which case the system state is inconclusive). The uncertainty in a test outcome of a component is described by its positive (negative) predictive error, which is the probability of a working (failing) component given that the test predicts a failing (working) component. Each component test involves a cost, and the aim is to find a policy with total minimum expected cost.

We show that a globally optimal policy for the sequential testing problem with imperfect tests can be found in polynomial time when the positive (negative) predictive error is the same for each component, and its sum is not more than one. Furthermore, the optimal policy that is found, can be represented compactly. The result follows from two key observations that can be derived from the assumptions on the predictive errors. Firstly, as the predictive error is component independent, it can be seen that the probability of a working (failing) system is only depending on the number of observed working and failing components rather than the order in which tests are conducted. Secondly, when the sum of predictive errors is bounded by one, the probability of a working (failing) system for a fixed number of tests does not decrease when more working (failing) components are observed. Using these observations, the problem with imperfect tests can be transformed into an equivalent generalized (k_0, k_1)-out-of-n testing problem (with perfect tests) in which the system needs at least k_1 working components to function and k_0 failing components to malfunction. We show that the polynomial-time algorithm of Chang et al.[1] for k-out-of-n systems can also be applied to this generalized testing problem.

References

[1] M.-F. Chang, W. Shi, and W.K. Fuchs. Optimal diagnosis procedures for k-out-of-n structures. *IEEE Transactions on Computers*, 39(4):559–564, 1990.

[2] J.A. Nachlas, S.R. Loney and B.A. Binney. Diagnostic-strategy selection for series systems. *IEEE Transactions on Reliability*, 39(3):273–280, 1990.

[3] T. Ünlüyurt. Sequential testing of complex systems: A review. *Discrete Applied Mathematics*, 142:189–205, 2004.

Staff and machine shift scheduling in a German potash underground mine

Marco Schulze (Speaker) * Jürgen Zimmermann †

1 Introduction

During the last five decades, numerous publications (e.g., [3], [4], and [5]) have appeared concerned with the application of optimization methods in the mining industry. Most of them focus on long-term production scheduling for underground mining, e.g., [1] as well as open pit mining, cf. [2]. In contrast, this paper addresses a short-term underground mine production scheduling problem. In particular, we emphasize the assignment and scheduling of single mining operations to available resources (i.e., machines and staff) in single working shifts.

The extraction of the examined German potash mine is done by room-and-pillar mining. In this mining system the mined material is extracted across a horizontal plane while leaving pillars of untouched material to support the roof of the mine. Thus, open areas (rooms) emerge between the pillars. As mining advances, a grid-like pattern of rooms and pillars is formed. There are two types of room-and-pillar mining: conventional mining and continuous mining. Except for some special applications, the excavation of potash is based on the former type involving drilling and blasting. This kind of underground mining is characterized by nine consecutive sub-steps (operations), that can be defined as a production cycle. For each processing step one special mobile machine out of a set of heterogenous machines is required.

In the field of potash excavation, several scheduling problems occur that can be embedded into a hierarchical planning approach. In so doing, each planning level takes a different planning horizon into consideration. In order to generate practicable schedules for each planning level, it is necessary to tackle the single problems in a top-down analysis, but there is also feedback from lower to higher levels. On a *tactical* planning level, a block-sequencing problem has to be solved, where the sequence in which the blocks should be removed from the mine in each period has to be specified. Thereby, a common assumption is that the mine/deposit is discretized into a grid of blocks, and each block consists of a specific volume of material with a specific potash quality. The aim is to minimize deviations of the mean quality per period from a prescribed potash quality level. On an *operational* planning level, the room-and-pillar mining method is considered in more detail. Here, one block is separated into the different consecutive operations from the production cycle. The resulting problem can be formulated as a

*`marco.schulze@tu-clausthal.de`. Operations Research Group, Clausthal University of Technology, Julius-Albert-Straße 2, 38678 Clausthal-Zellerfeld, Germany.

†`juergen.zimmermann@tu-clausthal.de`. Operations Research Group, Clausthal University of Technology, Julius-Albert-Straße 2, 38678 Clausthal-Zellerfeld, Germany.

hybrid flow shop scheduling problem (e.g., [6]) where the sum of completion times of all operations has to be minimized. The problem we focus on in this paper forms the *bottom level* of the hierarchical planning approach. In order to generate reasonable shift schedules, the overlying planning levels provide input data concerning which amount has to be mined per shift and which parts of the mine should be excavated with higher priority. Within our staff and machine shift scheduling problem (SMSSP) we consider several requirements, that are explained in the subsequent section.

2 Problem description for SMSSP

The input data for the SMSSP are mainly composed of jobs, machines, and workers (staff). A *job* is denoted by two components: on the one hand it is defined by a certain operation from the production cycle and on the other hand it is characterized by the place where the operation has to be executed in the mine. Derived from the two planning levels above, we obtain a priority list that induces a precedence constraint between any two jobs belonging to the same operation. Moreover, we have to consider precedence constraints for jobs from the same block, meaning, that an operation cannot be started before the preceding operation has finished. Finally, we have to know the processing time for any job depending on the machine and worker, respectively. Concerning the mobile *machines*, we need to know the number of available machines for the upcoming shift, the parking position, as well as the travel times between any two consecutive jobs. Another important data is the duration of the so-called technical service that has to be performed before using the machine for the first job and at the end of the shift. Regarding the *workers*, the number of available persons and their corresponding skills, including the individual skill levels (that depend on the machine group) have to be provided. When solving the considered scheduling problem, the following important practical requirements and aims have to be fulfilled:

- The output or performance of a shift is mainly evaluated by the total amount of potash (measured in tons) that was excavated by the available resources. Although the potash or crude salt of each block is not available until the blasting step is completed, this expected value is an input parameter of each job, no matter what operation has to be executed there. Consequently, one requirement is characterized by reaching a *prescribed amount of potash* for every machine group in each shift as good as possible.
- Another aim is the minimization of the *travel times* for all scheduled machines.
- Since we do not generate shift schedules in an online manner (maybe due to machine breakdowns concerning preceding operations), we have to provide suitable *backup jobs* for the machines.
- Another aspect relates to the number of so-called *change-overs*, which has to be as small as possible. A change-over is identified when a worker has to use another machine in order to process the subsequent job.
- One up to 10 jobs form a set of *connected jobs*. In case that the current mining progress requires the performance of different operations for a subset of connected jobs, a steady progress for all of them has to be achieved. Hence, the same operation should be executed for all jobs in the corresponding set. Moreover, it is required that the same machine of the related machine group should process the jobs, what is principally motivated by avoiding large transfer times and blocking (in terms of blocking the way for a machine of the same group).

- If a job could not be finished in the previous shift, it has to be scheduled as the *first task* for the appropriate machine because it usually was parked near to that job.
- We have to distinguish *interruptible and non-interruptible* jobs depending on the required operation. That means, for example, if there are 50 minutes left concerning the shift length, and the next job for a drilling machine which could be chosen, lasts more than 50 minutes, this job can not be scheduled and an alternative job has to be found.
- Due to legal regulations, *breaks* for the workers have to be integrated in the schedule, where a prescribed time window has to be incorporated.
- Each time we have to schedule a machine (out of a set of heterogeneous machines) from a machine group, we have to take a *prescribed order of usage* into account.
- In addition to the prescribed order of usage concerning the machines, we also have to consider a prescribed order of assignment from workers to single machines (apart from the decision derived from the different skill levels).

3 Solution procedure

In cooperation with our industry partner, we identified three important properties a solution procedure has to exhibit. At first, it has to fulfill all the complex practical requirements just mentioned. A second important fact is a small computation time, because the schedule can only be generated within the shift breaks. Moreover, we have to reach a high acceptance of the generated schedule in order to not disturb the motivation of the workers. As a consequence, we came to the conclusion to solve this problem by using a construction heuristic. The main idea of the heuristic is to generate a schedule in a stepwise manner by applying the phases *priority-based scheduling*, *staff changes*, and *job reassignment* iteratively. A detailed description of the procedure and a corresponding case study will be given during the talk.

References

[1] L.P. Trout (1995). *Underground Mine Production Scheduling Using Mixed Integer Programming.* Proceedings of the Application of computers and operations research in the minerals industries, Brisbane, Australia, pp. 395-400.

[2] S. Ramazan and R. Dimitrakopoulos (2004). *Recent applications of operations research and efficient MIP formulations in open pit mining.* Transactions of Society for Mining, Metallurgy, and Exploration, vol. 316, pp. 73-78.

[3] A. Weintraub, C. Romero, T. Bjørndal and R. Epstein (2007). *Handbook of operations research in natural resources.* Springer, New York.

[4] A.M. Newman, E. Rubio, R. Caro, A. Weintraub and K. Eurek (2010). *A review of operations research in mine planning.* Interfaces, vol. 40, no. 3, pp. 222-245.

[5] E. Kozan and S.Q. Liu (2011). *Operations Research for Mining: A Classification and Literature Review.* ASOR Bulletin, vol. 30, no. 1, pp. 2-23.

[6] M. Schulze and J. Zimmermann (2011). *Scheduling in the Context of Underground Mining.* Operations Research Proceedings 2010, eds. Hu B., K. Morasch, S. Pickl and M. Siegle, Springer, Berlin, pp. 611-616.

An exact algorithm for the chance-constrained resource-constrained project scheduling problem

Patricio Lamas (Speaker) * Erik Demeulemeester †

1 Introduction

In practice, the baseline project schedules that are obtained assuming deterministic durations for the project activities will almost certainly differ from the realized project schedules after its execution. This is due to the fact that any forecast on the duration of the project activities will have some degree of variability as a consequence of a broad set of uncertainty sources.

We propose an exact method for solving the chance-constrained resource-constrained project scheduling problem (CC RCPSP) (introduced in [2]) that generates a baseline project schedule with the following characteristics: it will be identical to the realized schedule for a given probability (under the mild assumption that no activity is executed before its baseline starting time), that probability will be independent of any reactive policy that is planned to be deployed and finally, this baseline schedule does not have the objective of minimizing the expected penalty (of the deviation between baseline and realized starting times) which is adequate only if we would be considering the execution of many projects in the long term. The main drawback of our approach is that the CC RCPSP is an extremely difficult problem to solve.

The contributions of this paper are the following: we provide complexity proofs for the CC RCPSP, we introduce theorems related to the obtention of preprocessing rules, lower bounds and feasible solutions of the problem. We present an exact algorithm that integrates and applies the previous theoretical results in order to obtain optimal solutions for the problem. Finally, we present results of numerical experiments for testing and comparing the practical performance of the algorithm.

2 Problem statement, complexity, preprocessing, bounds and algorithms

In this section we start presenting the problem statement for the CC RCPSP. We will assume that the probability distribution of the durations vectors is discrete over a finite support and that the probabilities of occurrence for each of these vectors are identical.

*Patricio.LamasVilches@kuleuven.be. Department of Decision Sciences and Information Management - KBI, Faculty of Business and Economics, KU LEUVEN

†Erik.Demeulemeester@kuleuven.be. Department of Decision Sciences and Information Management - KBI, Faculty of Business and Economics, KU LEUVEN

This is not a very restrictive assumption given that it is possible to obtain good approximations of problems with general probability distributions replacing the original distribution by a (finite and discrete) Monte Carlo sample (see [3]).

Therefore, the CC RCPSP is the problem of finding a subset of the total sample set that guarantees a predefined confidence level and that the obtained schedule is of minimum makespan given the durations defined by the selected sample subset. We define the problem more formally as follows:

INSTANCE: A confidence level $(1-\alpha)$, such that $0 < \alpha \leq 1$. A set W of n-dimensional activity-durations vectors $d^w = (d_1^w, \ldots, d_n^w)$, $w \in W$. Also we are given an uncomplete instance I of the traditional deterministic RCPSP composed by the resource availabilities, resources consumptions and precedence relations but not the activity durations, i.e. $[I, d]$, with d any n-dimensional vector of durations, is an instance of the RCPSP. An optimization oracle $O(\quad)$ for the RCPSP, such that for a given $[I, d]$, $O(I, d)$ always returns an optimal (non necessarily unique) set $S_{I,d}$ of starting times for each activity and consequently the optimal value $v_{I,d}$ for the RCPSP.

QUESTION: Find $W^* \subseteq W$ such that $|W^*| \geq \lceil |W|(1-\alpha) \rceil$ and v_{I,d^*} is minimum, with $d^* = \max_{w \in W^*} d^w$ (function max is component-wise).

For all the following theorems we provide no proof in this abstract due to space constraints. The following two complexity theorems are extensions of the NP-hard proof presented in [4].

Theorem 1 *There is no oracle-polynomial-time algorithm for solving CC RCPSP unless* $P = NP$

Theorem 2 *There is no polynomial-time approximation algorithm for CC RCPSP with approximate ratio bounded by a constant unless* $P = NP$

Let h be a matrix where every row h_i (with $i = 1, \ldots, n$) corresponds to the vector of the sample values of duration of activity i sorted in non-increasing order. Each column h^j (with $j = 1, \ldots, |W|$) corresponds to a duration vector, such that each of its components i is the j-th largest duration of activity i in the sample.

Theorem 3 $v_{I,\underline{d}}$ *is a lower bound for* $v_{I,d}$*, with* $\underline{d} = h^{\lceil |W|\alpha \rceil + 1}$

Finally, below we present a theorem that can be applied in the definition of a preprocessing rule that allows us to reduce the original sample size. First we define a reduced sample size W' as follows: $W' = \{w \in W | \exists i \in \{1, \ldots, n\} : h_i^w > h_i^{\lceil |W|\alpha \rceil + 1}\}$.

Theorem 4 $v_{I,d'^*} = v_{I,d^*}$*, with* v_{I,d'^*} *the optimal value of CC RCPSP considering the reduced sample* W'

Our exact algorithm is composed of three elements: a preprocessing procedure, an initial feasible solution heuristic and a Branch and Bound (B&B) algorithm. The preprocessing procedure is basically an application of Theorem 4. It outputs the reduced sample $W' \subseteq W$. The heuristic allows us to find an initial feasible solution simply selecting a set $W^0 \subseteq W'$ with $|W^0| = \lceil |W|(1-\alpha) \rceil$. The scenarios that belong to W^0 are the $\lceil |W|(1-\alpha) \rceil$ worst scenarios in W' according to some reasonable criterium. Finally, the optimal combination W^* is found using a B&B algorithm, where the bounding procedure is an application of Theorem 3.

3 Numerical results and conclusions

We developed computational experiments in order to test the performance of our exact algorithm. We considered 48 instances belonging to the PSPLIB (see [1]) with confidence levels equal to 0.99 and 0.95, and sample sizes varying between 100 and 1600. We compared its performance with a mathematical programming formulation modeled and solved with CPLEX 12.5. Based on such comparison we can conclude that the performance of our algorithm is competitive. Also, it has two other conceptual advantages. First, it only depends on an optimization oracle (or solver) for the RCPSP that can be used as a black-box with no major knowledge of its internal logic. Second, our algorithm is easily extensible to general chance-constrained programming problems.

References

[1] R. Kolisch and A. Sprecher (1996). *PSPLIB – A project scheduling problem library.* European Journal of Operational Research.

[2] P. Lamas and E. Demeulemeester (2014). *A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations.* Working paper.

[3] J. Luedtke and S. Ahmed (2008). *A sample aproximation approach for optimization with probabilistic constraints.* SIAM Journal on Optimization.

[4] J. Luedtke, S. Ahmed and G. Nemhauser (2010). *An integer programming approach for linear programs with probabilistic constraints.* Mathematical Programming.

Colored Bin Packing: Online Algorithms and Lower Bounds

Martin Böhm * György Dósa † Leah Epstein ‡ Jiří Sgall *
Pavel Veselý (Speaker) *

Introduction. In the *Online Black and White Bin Packing* problem proposed by Balogh et al. [2, 1] as a generalization of classical bin packing, we are given a list of items of size in $[0, 1]$, each item being either black, or white. The items are coming one by one and need to be packed into bins of unit capacity. The items in a bin are ordered by their arrival time. The additional constraint to capacity is that the colors inside the bins are alternating, i.e., no two items of the same color can be next to each other in the same bin. The goal is to minimize the number of bins used.

Online Colored Bin Packing is a natural generalization of Black and White Bin Packing in which items can have more than two colors. As before, the only additional condition to unit capacity is that we cannot pack two items of the same color next to each other in one bin.

Observe that optimal offline packings with and without reordering the items differ in this model. The packings even differ by a non-constant factor: Let the input sequence have n black items and then n white items, all of size zero. The offline optimal number of bins with reordering is 1, but an offline packing without reordering (or an online packing) needs n bins, since the first n black items must be packed into different bins. Hence we need to use the offline optimum without reordering in the analysis of online colored bin packing algorithms.

We use two lower bounds on the number of bins in any packing. The first is the sum of item sizes and the second is the maximal color discrepancy. The color discrepancy for a color c is the maximal difference between the number of items of color c and the number of other items on a segment of the input sequence. Formally, let $s_{c,i} = 1$ if the i-th item from the input sequence has color c, and $s_{c,i} = -1$ otherwise. We define the maximal discrepancy as $\max_c \max_{i,j} \sum_{\ell=i}^{j} s_{c,\ell}$.

There are several well-known and often used algorithms for classical Bin Packing. We investigate the *Any Fit* family of algorithms (AF). These algorithms pack an incoming item into some already open bin whenever it is possible with respect to the size and color constraints. The choice of the open bin (if more are available) depends on the algorithm. AF algorithms thus open a new bin with an incoming item only when there is no other possibility. Among AF algorithms, *First Fit* (FF) packs an incoming item into the first bin where it fits (in the order by creation time), *Best Fit* (BF) chooses the bin with the highest level where the item fits and *Worst Fit* (WF) packs the item into the bin with the lowest level where it fits.

*{bohm,sgall,vesely}@iuuk.mff.cuni.cz. Computer Science Institute of Charles University, Prague, Czech Republic.
†dosagy@almos.vein.hu. Department of Mathematics, University of Pannonia, Veszprém, Hungary.
‡lea@math.haifa.ac.il. Department of Mathematics, University of Haifa, Haifa, Israel.

Next Fit (NF) is more restrictive than Any Fit algorithms, since it keeps only a single open bin and puts an incoming item into it whenever the item fits, otherwise the bin is closed and a new one is opened.

Previous results. Balogh et al. [2, 1] introduced the Black and White Bin Packing problem. As the main result, they give an algorithm *Pseudo* with the absolute competitive ratio exactly 3 in the general case and $1 + d/(d-1)$ in the parametric case, where the items have sizes of at most $1/d$ for a real $d \geq 2$. They also proved that there is no deterministic or randomized online algorithm whose asymptotic competitiveness is below $1 + \frac{1}{2 \ln 2} \approx 1.721$.

Concerning specific algorithms, they proved that Any Fit algorithms are at most 5-competitive and even optimal for zero-size items. They show input instances on which FF and BF create asymptotically $3 \cdot OPT$ bins. For WF there are sequences of items witnessing that it is at least 3-competitive and $(1+d/(d-1))$-competitive in the parametric case for an integer $d \geq 2$. Furthermore, NF is not constant competitive.

In the offline setting, Balogh et al. [2] gave a 2.5-approximation algorithm with $\mathcal{O}(n \log n)$ time complexity and an asymptotic polynomial time approximation scheme, both when reordering is allowed.

Our results. We completely solve the case of Colored Bin Packing for zero-size items which is an important case for constructing general algorithms. The offline optimum (without reordering) is actually not only lower bounded by the color discrepancy, but equal to it for zero-size items. For online algorithms, we give an (asymptotically) 1.5-competitive algorithm Balancing Any Fit (BAF) which is optimal. The algorithm mostly puts an incoming item into a bin of the most frequent other color.

In fact, BAF always uses at most $\lceil 1.5 \cdot OPT \rceil$ bins and we can force any deterministic online algorithm to use at least $\lceil 1.5 \cdot OPT \rceil$ bins while the optimum is OPT for any value of $OPT \geq 2$. This shows that the absolute ratio of our algorithm is $5/3$ which is optimal.

For items of arbitrary size we prove a lower bound of 2 on the asymptotic competitive ratio of any deterministic online algorithm using only two colors, i.e., for Black and White Bin Packing. Then by combining this lower bound with the lower bound of 1.5 for zero-size items we obtain a lower bound of 2.5 for items of arbitrary size and at least three colors.

We use the optimal algorithm for zero-size items and the algorithm Pseudo to design an (absolutely) 3.5-competitive algorithm which is also (asymptotically) $(1.5+d/(d-1))$-competitive in the parametric case, where the items have sizes of at most $1/d$ for a real $d \geq 2$. (Note that for $d < 2$ we have $d/(d-1) > 2$ and the bound for arbitrary items is better.)

We show that algorithms BF, FF, WF and Pseudo are not constant competitive, even for instances with only three colors and very small items, in contrast to their 3-competitiveness for two colors. Their competitiveness cannot be bounded by any function of the number of colors even for only three colors and very small items.

For Black and White Bin Packing, we improve the upper bound on the absolute competitive ratio of Any Fit algorithms in the general case to 3 which is tight for BF, FF and WF. For WF in the parametric case we prove that it is absolutely $(1+d/(d-1))$-competitive for a real $d \geq 2$ which is also tight. Therefore, WF has the same competitive ratio as the Pseudo algorithm.

Our results were presented at the 14th Scandinavian Symposium and Workshops

on Algorithm Theory (SWAT 2014) [4] and at the 12th Workshop on Approximation and Online Algorithms (WAOA 2014) [3]. The abstract of the paper was accepted to the workshop Trends in Online Algorithms 2014 (TOLA 2014) without a publication in proceedings. The results for Black and White Bin Packing were presented at the conference MATCOS 2013 (also without a publication in proceedings).

Conclusions and open problems. The Colored Bin Packing for zero-size items is completely solved. For items of arbitrary size, our online algorithm still leaves a gap between our lower bound 2.5 and our upper bound of 3.5. The upper bounds are only 0.5 higher than for two colors (Black and White Bin Packing) where a gap between 2 and 3 remains for general items.

Classical algorithms FF, BF and WF, although they maintain a constant approximation for two colors, start to behave badly when we introduce the third color. For two colors, we now know their exact behavior. In fact, all Any Fit algorithms are absolutely 3-competitive which is a tight bound for FF, BF and WF. However, for items of size up to $1/d$, $d \geq 2$, FF and BF remain 3-competitive, while WF has the absolute competitive ratio $1+d/(d-1)$. Thus we now know that even the simple Worst Fit algorithm matches the performance of Pseudo, the online algorithm with the best competitive ratio known so far. It is also an interesting question whether it holds that Any Fit algorithms cannot be better than 3-competitive for two colors.

References

[1] J. Balogh, J. Békési, G. Dósa, L. Epstein, H. Kellerer, and Z. Tuza (2014). *Online results for black and white bin packing.* In *Theory of Computing Systems*, pages 1–19.

[2] J. Balogh, J. Békési, G. Dósa, H. Kellerer, and Z. Tuza (2013). *Black and white bin packing.* In *Approximation and Online Algorithms*, LNCS 7846, pages 131–144, Springer.

[3] M. Böhm, J. Sgall, and P. Veselý (2014). *Online colored bin packing.* To appear in proceedings of *12th Workshop on Approximation and Online Algorithms* (WAOA 2014). Also ArXiv 1404.5548.

[4] G. Dósa and L. Epstein (2014). *Colorful bin packing.* In *Algorithm Theory – SWAT*, LNCS 8503, pages 170–181, Springer.

Algorithms and Lower Bounds for Online Bin Stretching

Martin Böhm (Speaker) * Jiří Sgall * Rob van Stee † Pavel Veselý *

Online Bin Stretching, introduced by Azar and Regev in 1998 [1], is an online problem in the vein of Bin Packing problems. Items of size between 0 and 1 arrive in a sequence, and the algorithm needs to pack them as soon as each item arrives, but it has two advantages: (i) The packing algorithm knows m, the number of bins that an optimal offline algorithm would use, and must also use only at most m bins, and (ii) the packing algorithm can use bins of capacity S for some $S \geq 1$. The goal is to minimize the *stretching factor* S.

While formulated as a bin packing variant, Online Bin Stretching can also be thought of as a semi-online scheduling problem, in which we schedule jobs in an online manner on exactly m machines, before any execution starts. We have a guarantee that the optimum offline algorithm could schedule all jobs with makespan 1. Our task is to present an online algorithm with makespan of the schedule being at most S.

History. Online Bin Stretching has been proposed by Azar and Regev [1]. The original lower bound of 4/3 has appeared even before that, in [6], for two bins along with a matching algorithm. Azar and Regev extended this bound to any number of bins.

The problem has been revisited recently. Kellerer and Kotov [5] have achieved a stretching factor $11/7 \approx 1.57$ and Gabay et al. [4] have achieved $26/17 \approx 1.53$. In the case with only three bins, the previously best algorithm was due to [1], with $S = 1.4$.

On the lower bound side, the lower bound 4/3 of [1] was surpassed only for three bins by Gabay et al. [3], who show a lower bound of 19/14, using computer search.

Our contributions. On the algorithmic side, we present a new algorithm for Online Bin Stretching with a stretching factor of 1.5. We also focus on the case of three bins. For it, we present an algorithm with stretching factor $22/16 = 1.375$.

On the lower bound side, we focus on the problem of Online Bin Stretching on three bins. We show a lower bound of $45/33 = 1.\overline{36}$ for the problem. We build on the ideas of Gabay et al. [3], improving their algorithm both technically and conceptually.

A subset of our results was presented at WAOA 2014 [2].

1 Algorithm for an arbitrary number of bins

We build on the techniques of [5, 4] who designed two-phase algorithms where the first phase tries to fill some bins close to $S - 1$ and achieve a fixed ratio between these bins and empty bins, while the second phase uses the bins in blocks of fixed size and analyzes each block separately. This technique, with some case analysis, seemed to be able to lead to improved results approaching 1.5.

*{bohm,sgall,vesely}@iuuk.mff.cuni.cz. Computer Science Inst. of Charles Univ., Prague, Czech Republic.

†rob.vanstee@leicester.ac.uk. Dept. of Computer Science, Univ. of Leicester, Leicester, UK.

We rescale the bin sizes so that the optimal bins have size 12 and the bins of the algorithm have size 18. In the first phase of the algorithm we try to fill the bins so that their size is at most 6, as this leaves space for an arbitrary item in each bin. Of course, if items larger than 6 arrive, we need to pack them differently, namely in bins of size at least 12, whenever possible. We stop the first phase when the number of non-empty bins of size at most 6 is three times the number of empty bins. In the second phase, we work in blocks of three non-empty bins and one empty. The goal is to show that we are able to fill the bins so that the average size is at least 12, which guarantees we are able to pack the total size of $12m$ which is the upper bound on the size of all items.

The limitation of the previous results using this scheme was that the volume achieved in a typical block of four bins is slightly less than four times the size of the optimal bin, which then leads to bounds strictly above 3/2. This is also the case in our algorithm. However, we notice that such a block contains five items of size larger than 6 which the optimum cannot fit into four bins. To take an advantage of this, we cannot analyze each block separately. Instead, we need to show that a bin with no item of size more than 6 typically has size at least 13 and amortize among the blocks of different types using a weight function. This is the main new technical idea of our proof.

2 Algorithm for three bins

We scale the input sizes by 16 and set the stretched capacity of a bin to 22.

A natural idea is to try to pack first all items in a single bin, as long as possible. In general, this is the strategy that we follow. However, it turns out that from the very beginning we need to put items in two bins even if the items as well as their total size are relatively small.

We introduce several *good situations* as a heart of our algorithm. These are configurations of the three bins which allow us to complete the packing regardless of the following input. One example is this:

Good Situation. *Given a partial packing such that* $s(A) + s(B) \geq 26$*, there exists an online algorithm that packs remaining items into three bins of capacity* 22*.*

The algorithm repeatedly uses a special variant of First Fit with modified smaller bin sizes; moreover, whenever we can pack an item so that we can reach one of the good situations, we do so despite exceeding the modified bin capacities.

3 Lower bound for three bins

As with many other online algorithms, we can think of Online Bin Stretching as a two player game. The first player (Algorithm) is presented with an item i. Algorithm's goal is to pack it into m bins of capacity S. This mimics the task of any algorithm for Online Bin Stretching. The other player (Adversary) decides which item to present to the Algorithm in the next step. The goal of the Adversary is to force Algorithm to overpack at least one bin.

The two main obstacles to implementing a search of this game are the following:

1. Adversary can send an item of arbitrary small size;
2. Adversary needs to make sure that at any time of the game, an offline optimum can pack the items arrived so far into three bins of size 1.

To overcome the first problem, it makes sense to create a sequence of games based on the granularity of the smallest item that can be packed. A natural granularity for the scaled game are integer items, which correspond to multiples of $1/T$ in the non-scaled problem. The second problem increases the complexity of every game turn of the ADVERSARY.

Note that the ideas described above have been described previously in [3]. Our improvements include:

1. We avoid using CSP and instead employ a sparse dynamic programming solution for the knapsack problem. This improves the running time significantly.
2. We use good situations (as in Section 2) to prune the tree.
3. We make extensive use of hashing yet keep our memory below a fixed limit.
4. We implement the search in the lower-level C programming language.

We were able to check all trees of granularity up to 41 (compared to 20 in [3]). Selected results and running times are given below:

Target fraction	*Decimal form*	*L. b. found*	*Elapsed time*
19/14	1.3571	Yes	2s.
30/22	$1.\overline{36}$	No	6s.
34/25	1.36	**Yes**	15s.
45/33	$1.\overline{36}$	**Yes**	1min. 48s.
56/41	1.3659	No	30min.

References

[1] Y. Azar and O. Regev. On-line bin-stretching. In *Randomization and Approximation Techniques in Computer Science*, pages 71–81. Springer, 1998.

[2] M. Böhm, J. Sgall, R. van Stee, P. Veselý. Better Algorithms for Online Bin Stretching. 12th Workshop on Approximation and Online Algorithms (WAOA 2014), LNCS, Springer. To appear in 2015.

[3] M. Gabay, N. Brauner, V. Kotov. Computing lower bounds for semi-online optimization problems: Application to the bin stretching problem. HAL preprint hal-00921663, 2013.

[4] M. Gabay, V. Kotov, N. Brauner. Semi-online bin stretching with bunch techniques. HAL preprint hal-00869858, 2013.

[5] H. Kellerer and V. Kotov. An efficient algorithm for bin stretching. *Operations Research Letters*, 41(4):343–346, 2013.

[6] H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Oper. Res. Lett.*, 21:235–242, 1997.

Online Nonpreemptive Scheduling for Electricity Cost in Smart Grid

Wing-Kai Hon * Hsiang-Hsuan Liu (Speaker) *†‡ Prudence W.H. Wong †

1 Introduction

We study an online scheduling problem arising in demand response management in electrical smart grid [5, 8]. In a smart grid consumers send in power requests with the power requirement, required duration of service, and the time intervals that this request can be served (giving some flexibility). For example, a consumer may request the dishwasher to operate for one hour contiguously during the periods 8am to 12pm. Peak demand hours happen only for a short duration, yet makes existing electrical grid less efficient. E.g., in the US power grid, 10% of all generation assets and 25% of distribution infrastructure are required for less than 400 hours per year, roughly 5% of the time [3]. *Demand response management* attempts to overcome this problem by shifting user's demand to off-peak hours to reduce peak load [7, 9]. Research initiatives include GridWise [6], EnviroGrid$^{\text{TM}}$ [4], etc.

The problem. Formally, each request j comes with its release time r_j, deadline d_j, and length ℓ_j. We assume the power requirement is unit for all requests. A request needs to be assigned non-preemptively for ℓ_j timeslots within the interval $I_j = [r_j, d_j]$. Several requests can be assigned to the same timeslot and the load at t, $load(t)$ is the sum of the power requirements of all requests allocated to it (for unit power requirement, the load is the number of requests assigned). The electricity cost is measured by a convex function of the load in each timeslot, in particular, we consider $(load(t))^\alpha$, where $\alpha \geq 2$. The aim is to minimize the total electricity cost to complete all requests. We consider the online setting where requests arrive at unpredictable time and once a request starts being served, it cannot be moved to another time. We measure the performance of the online algorithm by the competitive ratio, which is the worst-case ratio of the electricity cost of the algorithm to the optimal offline algorithm.

Related work. Koutsopoulos et al. [7] has formulated a similar problem to our problem where both request duration and power requirement are variable, and the cost function is piecewise linear. They show that the problem is NP-hard, and their proof can be adapted to show the NP-hardness of the general problem studied in this paper for which requests have arbitrary duration or arbitrary power requirement. They

*{wkhon,hhliu}@cs.nthu.edu.tw Department of Computer Science, National Tsing Hua University, Delta Building, 101 Kuang Fu Road, Section 2, Hsinchu, Taiwan 300.

†{hhliu,pwong}@liverpool.ac.uk. Department of Computer Science, University of Liverpool, Ashton Building, Ashton Street, Liverpool, L69 3BX, UK.

‡The work is supported by UoL-NTHU Dual-PhD programme studentship.

also provided stochastic analysis of their proposed online algorithms. For unit power requirement and unit length, fast optimal offline algorithms have been proposed [2].

Another related problem is the *dynamic voltage scaling (DVS)* problem [10] where processor speed can be scaled (hence job processing time varies), running at speed s finishes s units of work and consumes s^α units of energy per time. Non-preemptive algorithms have been studied for this problem (e.g., [1]). The two problems have analogy that running at higher speed in a DVS processor is like serving several requests in our problem; both cost functions are convex functions of the load/speed. The major differences are that in our problem each request takes a fixed given length to complete and more than one request can be served at the same time; while in the DVS problem the length varies depending on the speed and only one job can be processed at each time. For the DVS problem, an online algorithm called AVR (Average Rate) has been proposed and showed to be $(2\alpha)^\alpha/2$-compeitive.

2 Results

To illustrate the difference between our problem and the DVS problem, consider an instance with a unit-length request j with arbitrarily large deadline. The request will be allocated to one timeslot with load 1 and the electricity cost is 1. In the DVS problem, the job of size 1 will be scheduled using a speed of $1/(d_j - r_j)$ for a duration of $(d_j - r_j)$, hence the power consumption is $1/(d_j - r_j)^{\alpha-1}$. Hence the ratio of the electricity cost to the power consumption can be arbitrarily large. That is, in the smart grid problem, at each timeslot, we have to schedule an integer number of requests while in the DVS problem, the speed at any time can be a real number. This leads to problem when the "minimum required speed" for the DVS problem is below 1. However, if the "minimum required speed" is above 1, then the load in our problem is more comparable.

In the discussion below, we focus on determining the number of requests to be scheduled at each time and leave the choices of which requests to serve to the full paper. Our results make use of the relationship with the DVS problem. Let $\mathcal{O}_{dvs}$ and $\mathcal{O}_{grid}$ denote the optimal offline algorithm for the DVS problem and the smart grid problem respectively. We denote the AVR algorithm as $\mathcal{A}_{avr}$ and our online algorithms $\mathcal{A}_{grid}$. We also use these notations to denote the costs of the corresponding algorithms. One can show that any schedule for the grid problem can be converted to a schedule for the DVS problem with the same cost, hence we have the following property: $\mathcal{O}_{grid} \geq \mathcal{O}_{dvs}$.

This property suggests an approach based on AVR. Recall that AVR schedules at speed $avg(t)$ which is the sum of "density" of all jobs whose interval contains t, where the density of a job j is defined by $\ell(j)/(d_j - r_j)$. We also make use of this $avg(t)$ function but since we need to schedule integral number of requests, we make reference to $\lceil avg(t) \rceil$ instead. We show that it is possible to get constant competitive ratio which is a multiple of $(2\alpha)^\alpha/2$, the competitive ratio of AVR with respect to $\mathcal{O}_{dvs}$. The main idea is to split the time line into two parts: $\mathcal{I}_{\leq 1}$ contains all timeslots t such that $avg(t) \leq 1$ and $\mathcal{I}_{>1}$ for $avg(t) > 1$ respectively.

Lemma 1 *Given an instance with $avg(t) > 1\ \forall t$, if we have an algorithm $\mathcal{A}_{grid}$ such that there exists a constant c_1, $load(t) \leq c_1 \lceil avg(t) \rceil\ \forall t$, then we have $\mathcal{A}_{grid} \leq \frac{(4c_1\alpha)^\alpha}{2} \cdot \mathcal{O}_{grid}$.*

Theorem 1 *If we have $\mathcal{A}_{grid}$ such that (1) $load(t) \leq c_1 \cdot \lceil avg(t) \rceil$ for all $t \in \mathcal{I}_{>1}$, and (2) $load(t) \leq c_2$ for all $t \in \mathcal{I}_{\leq 1}$. Then we have $\mathcal{A}_{grid} \leq (\frac{(4c_1\alpha)^\alpha}{2} + c_2^{\alpha-1}) \cdot \mathcal{O}_{grid}$.*

With Theorem 1, we propose several algorithms for different input scenarios (see Table 1). We first focus on the simple setting studied in [2] for unit length jobs (Case 1). Then we turn to arbitrary lengths (Case 4) based on two simpler settings (Cases 2 and 3). Table 1 also shows lower bounds on the competitive ratios. A natural extension is to consider arbitrary release time, deadline, and request length.

Case	Release Time	Deadline	Time Duration	Upper Bound			Lower Bound
				c_1	c_2	Competitive Ratio	Competitive Ratio
1	Arbitrary	Arbitrary	Unit	1	1	$\frac{(4\alpha)^\alpha}{2}+1$	$2^{\alpha-1}$
2	Same	Same	Arbitrary	2	0	$\frac{(8\alpha)^\alpha}{2}$	
						$2^{2\alpha}$	
3	Same	Arbitrary	Same	2	1	$\frac{(8\alpha)^\alpha}{2}+1$	
4	Same	Arbitrary	Arbitrary	2	1	$\frac{(8\alpha)^\alpha}{2}+1$	
5	Agreeable		Arbitrary	3	1	$\frac{(12\alpha)^\alpha}{2}+1$	[Case 1]
6	Laminar		Same	4	2	$\frac{(16\alpha)^\alpha}{2}+2^{\alpha-1}$	[Case 1]
7	Arbitrary	Arbitrary	Arbitrary			$\log^\alpha n$	$\max\{2^{\alpha-1}, 3^{\alpha-3}\}$

Table 1: Different settings and the corresponding competitive ratio.

References

[1] Antonios Antoniadis and Chien-Chung Huang. Non-preemptive speed scaling. *J. Scheduling*, 16(4):385–394, 2013.

[2] M. Burcea, W.-K. Hon, H.-H. Liu, P. W. H. Wong, and D. K. Y. Yau. Scheduling for electricity cost in smart grid. In *COCOA*, pages 306–317, 2013.

[3] C. Chen, K. G. Nagananda, G. Xiong, S. Kishore, and L. V. Snyder. A communication-based appliance scheduling scheme for consumer-premise energy management systems. *IEEE Trans. Smart Grid*, 4(1):56–65, 2013.

[4] REGEN Energy Inc. *Environgrid^{TM}smart Grid Bundle.* `http://www.regenenergy.com/press/announcing-the-envirogrid-smart-grid-bundle/`.

[5] A. Ipakchi and F. Albuyeh. Grid of the future. *Power and Energy Magazine, IEEE*, 7(2):52–62, March 2009.

[6] L. D. Kannberg, D. P. Chassin, J. G. DeSteese, S. G. Hauser, M. C. Kintner-Meyer, R. G. Pratt, L. A. Schienbein, and W. M. Warwick. Gridwisetm: The benefits of a transformed energy system. *CoRR*, nlin/0409035, 2004.

[7] I. Koutsopoulos and L. Tassiulas. Control and optimization meet the smart power grid: Scheduling of power demands for optimal energy management. In *e-Energy*, pages 41–50, 2011.

[8] T.J. Lui, W. Stirling, and H.O. Marcy. Get smart. *Power and Energy Magazine, IEEE*, 8(3):66–78, May 2010.

[9] S. Salinas, M. Li, and P. Li. Multi-objective optimal energy consumption scheduling in smart grids. *IEEE Trans. Smart Grid*, 4(1):341–348, March 2013.

[10] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.

Scheduling Imprecise Computations on Parallel Machines with Linear and Non-Linear Error Penalties*

Akiyoshi Shioura † Natalia V. Shakhlevich (speaker) ‡
Vitaly A. Strusevich §

1 Introduction

We consider scheduling problems that have been mainly studied within the body of research on imprecise computation. In these problems computation tasks have to be assigned to parallel processors in such a way that their mandatory parts are fully executed, while their optional parts are processed only if a task can complete before the due date. The remaining optional part is understood as the error of computation for a task. The objectives include minimizing the total weighted error, the maximum weighted error, and various constrained versions, such as minimizing the total error subject to smallest maximum error. Additionally, we also study the problems of minimizing the quadratic error cost function and its various constrained versions.

Unlike the earlier algorithms, which are often applicable to only specific versions of the problem, the new approach we propose uses a common tool based on advanced network flow techniques, namely parametric max-flow in combination with improved algorithms for bipartite networks. It is applicable to a broad range of problems, with linear and non-linear objectives, is easier to justify and analyze, and achieves the time complexity known for solving the feasibility versions of the same problems with fixed processing times.

2 Description of models

Formally, in the imprecise computation model the jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on parallel machines. For each job $j \in N$, its processing time $p(j)$ is not given in advance but has to be chosen by the decision-maker from a given interval $[l(j), u(j)]$, where $l(j)$ is the duration of the mandatory part, while the remaining part $u(j) - l(j)$ is optional. The value $x(j) = u(j) - p(j)$ is the computation error which affects the accuracy of computation.

Each job $j \in N$ is given a release date $r(j)$ and a deadline $d(j)$. Processing of a job can be preempted and resumed later, possibly on another machine. Typically, the

*This research was supported by the EPSRC funded project EP/J019755/1

†shioura@dais.is.tohoku.ac.jp. Graduate School of Information Sciences, Tohoku University, Sendai, Japan.

‡n.shakhlevich@leeds.ac.uk. School of Computing, University of Leeds, Leeds LS2 9JT, U.K.

§v.strusevich@gre.ac.uk. Department of Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, London SE10 9LS, U.K.

problems of imprecise computation are those of finding a deadline feasible preemptive schedule that minimizes a certain function F that depends on errors, e.g., the total error cost or the maximum error cost. We associate each job j with two unit-costs, $w_T(j)$ and $w_M(j)$. In such a doubly-weighted system of imprecise computation the costs $w_T(j)$ are involved in computing the total error cost, which in the linear case is defined as

$$F_\Sigma = \sum_{j \in N} w_T(j)\, x(j)$$

and in the quadratic case as

$$F_{\text{quad}} = \sum_{j \in N} w_T(j) x(j)^2.$$

Similarly, the costs $w_M(j)$ are involved in computing the maximum error cost, which is defined as

$$F_{\max} = \max\{x(j)/w_M(j) \,|\, j \in N\}.$$

As far as the machine environment is concerned, we are given m parallel machines. Identical machines have the same speed, so that for a job j with an actual processing time $p(j)$ the total length of the time intervals in which this job is processed in a feasible schedule is equal to $p(j)$. If the machines are uniform, then it is assumed that machine M_i has speed s_i, $1 \leq i \leq m$.

Depending on the machine environment and the objective function we generically denote the problems under consideration by $\Pi_\beta(\alpha)$, where $\beta \in \{\Sigma, \max, \text{quad}\}$ is the objective function, and $\alpha \in \{P, Q\}$ is the machine system, identical (P) or uniform (Q). For example, $\Pi_\Sigma(P)$ denotes the problem of minimizing the total error cost on identical machines, while $\Pi_{\max}(Q)$ denotes the problem of minimizing the maximum error cost on uniform machines.

As is traditional in the imprecise computation literature, we also look at the constrained problems, which we denote by $\Pi_{\beta'|\beta''}(\alpha)$. For these problems, the objective function $F_{\beta'}$ is minimized in the class of the schedules with the minimum value of $F_{\beta''}$, where $\beta', \beta'' \in \{\Sigma, \max, \text{quad}\}$, $\beta' \neq \beta''$. For example, $\Pi_{\Sigma|\max}(P)$ denotes the problem of finding a schedule on parallel identical machines that minimizes the total error cost among all schedules with the smallest maximum error cost.

Each problem with $p(j) \in [l(j), u(j)]$ can be seen as an extension of the feasibility problem $\Pi(\alpha)$, in which the processing times of all jobs are fixed, i.e., equal to given values $p(j), 1 \leq j \leq n$. To solve problem $\Pi(\alpha)$ means either to find a feasible schedule for the corresponding machine environment if it exists or to report that such a schedule does not exist.

3 The main result

Theorem 1 *For $\beta \in \{\Sigma, \max, \text{quad}\}$, each problem $\Pi_\beta(P)$ on m identical parallel machines is solvable in $O(n^3)$ time, and each problem $\Pi_\beta(Q)$ on m uniform parallel machines is solvable in $O(mn^3)$ time. The constrained versions $\Pi_{\beta'|\beta''}(P)$ and $\Pi_{\beta'|\beta''}(Q)$, where $\beta', \beta'' \in \{\Sigma, \max, \text{quad}\}$, $\beta' \neq \beta''$, are also solvable in $O(n^3)$ and $O(mn^3)$ time, respectively. These running times meet the best known running times required for solving the feasibility problems $\Pi(P)$ and $\Pi(Q)$, respectively.*

For comparison, the table below lists the complexity estimates of the known approaches.

Identical machines			Uniform machines		
$\Pi_{\Sigma}(P)$	$O\left(n^4 \log n\right)^*$	[1, 5, 7]	$\Pi_{\Sigma}(Q)$	$O\left(m^2 n^4 \log mn\right)$	[5]
				$O\left(mn^4\right)$	[6]
$\Pi_{\max}(P)$	$O\left(n^4\right)^*$	[2, 4]	$\Pi_{\max}(Q)$	$O\left(mn^4\right)$	[8]
$\Pi_{\Sigma\mid\max}(P)$	$O\left(n^4\right)^*$	[2, 5]	$\Pi_{\Sigma\mid\max}(Q)$	$O\left(mn^4\right)$	[8]
$\Pi_{\max\mid\Sigma}(P)$	$O\left(n^5\right)^*$	[2, 3]	$\Pi_{\max\mid\Sigma}(Q)$	$O\left(mn^5\right)$	[8]
	(if all w_T-weights are distinct)			(if all w_T-weights are distinct)	

*after correcting a faulty claim that problem $\Pi(P)$ is solvable in $O(n^2 \log^2 n)$ time

Notice that the quadratic objective F_{quad} has not been studied in the past, while our technique handles it as a slight generalization. The complexity estimate for all versions of the problem involving F_{quad}, even in combination with Σ or max, remains the same as for the feasibility problem $\Pi(\alpha)$.

References

[1] J.Y. Chung, W.-K. Shih, J.W.S. Liu, and D.W. Gillies (1989) Scheduling imprecise computations to minimize total error. *Microproc. Microprog.* 27: 767–774.

[2] K.I.-J. Ho (2004) Dual criteria optimization problems for imprecise computation tasks. In *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, ed. J.Y.-T. Leung (Chapman & Hall/CRC) pp. 35-1 – 35-26.

[3] K.I.-J. Ho, and J. Y.-T. Leung (2004) A dual criteria preemptive scheduling problem for minimax error of imprecise computation tasks, *Int. J. Found. Comput. Sci.* 15: 717–731.

[4] K.I.-J. Ho, J. Y.-T. Leung, and W.-D. Wei (1994) Minimizing maximum weighted error for imprecise computation tasks, *J. Algorithms* 16: 431–452.

[5] J.Y.-T. Leung (2004) Minimizing total weighted error for imprecise computation tasks. In *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, ed. J.Y.-T. Leung (Chapman & Hall/CRC, London), 34-1 – 34-16.

[6] N.V. Shakhlevich and V.A. Strusevich (2008) Preemptive scheduling on uniform parallel machines with controllable job processing times. *Algorithmica* 51**:** 451–473.

[7] W.-K. Shih, J.W.S. Liu, J.-Y. Chung, D.W. Gillies (1989) Scheduling tasks with ready times and deadlines to minimize average error, *ACM SIGOPS Oper. Syst. Review* 23: 14–28.

[8] G. Wan, J.Y.-T. Leung, M.L Pinedo (2007) Scheduling imprecise computation tasks on uniform processors, *Inform. Process. Lett.* 104, 45–52.

Scheduling Tasks to Minimize Active Time on a Processor with Unlimited Capacity*

Ken C.K. Fong † Minming Li ‡ Shi Li § Sheung-Hung Poon ¶
Weiwei Wu ‖ Yingchao Zhao **

1 Introduction

Energy efficiency problems have been well studied by researchers in the past decades [1, 4, 9]. Its objective is to reduce the energy consumption without performance degradation. With the trend of cloud computing and big data, there are millions of machines running in the data centers in each second. One way to reduce the energy consumption is to turn off the idle machines when the machines do not have any jobs to process. For instance, the storage cluster in the data centers can be turned off to save energy during low utilization period [2].

Ikura and Gimple introduced batch scheduling which scheduled jobs together in order to maximize their idle periods to save the energy[8]. They proposed an algorithm to minimize the completion time for single batch processing machine with agreeable deadlines where $d_i \leq d_j$ if $r_i < r_j$. In batch scheduling, the processor can schedule up to B jobs at any time t. Let J_t be the number of jobs which are scheduled at time t. The objective is to schedule all the jobs in the time slots satisfying $J_t < B$ to minimize the number of active time slots. Chang et al [6] proposed a linear time algorithm to schedule unit length jobs in batch scheduling. Moreover, they consider another version of the batch scheduling problem where jobs have arbitrary lengths and preemption is allowed. They presented an $O(\sqrt{Lm})$ time algorithm to solve this preemptive scheduling problem for $B = 2$ where L is the the sum of all job lengths and m is the total number of time slots which are feasible for some job.

Besides batch scheduling, the "min-gap"strategy [4, 5] is another approach for energy saving. When the machines are idle, they are transited to the suspended state without any energy consumption. However, in practice, small amount of energy will be consumed in the process of waking up the machines from suspended state. If the number of idle periods can be minimized, less energy is consumed for waking up the idle machines. Hence, the objective of min-gap strategy is to find a schedule such that the number of

*This work was partly supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 122512]

†ken.fong@my.cityu.edu.hk. Department of Computer Science, City University of Hong Kong, HK.
‡minming.li@cityu.edu.hk. Department of Computer Science, City University of Hong Kong, HK.
§shili@ttic.edu. Toyota Technological Institute, Chicago, US .
¶spoon@cs.nthu.edu.tw. National Tsing Hua University, Hsin-Chu, Taiwan.
‖weiweiwu@seu.edu.cn. Southeast University, China.
**yczhao@cihe.edu.hk. Department of Computer Science, Caritas Institute of Higher Education, HK.

idle periods can be minimized. Baptiste[4] was the first to propose a polynomial-time algorithm with running time of $O(n^7)$. Later, Baptiste et al.[5] improved the complexity to $O(n^5)$. They also presented an $O(n^4)$ algorithm for a special instance of this problem where all jobs have unit length. Angel et al.[3] consider the special setting of this problem, where the jobs have agreeable deadlines. They presented an $O(n^2)$ algorithm for the single processor case, and in the multiprocessor case with m machines, they presented an $O(n^2m)$ algorithm for unit time tasks.

The recent work of Tavakoli et al. [10] and Fang et al.[7] studied this problem in a different scenario. The scenario they use is the interval data sharing problem which is abstracted from wireless sensor networks. When the base station broadcasts one sampled data, it can be received and used by any number of sensors whose sampling period contains the time point. On the condition that each sensor receives a certain amount of data during the sampling period, their objective is to minimize the transmission energy of the base station. In [10], each application only requires discrete data at some time points whereas the work in [7] considers a continuous interval of sampling data which is most relevant to this paper. They provide a 2-approximation algorithm for the general case and an $O(n^2)$ instance where the lengths of the execution for all the jobs are the same.

2 Problem Formulation

Formally, we are given n jobs, where each job j_i has a release time r_i, processing time p_i and the deadline d_i. For any job j_i, it can only be scheduled within $[r_i, d_i]$ and we focus on the non-preemptive version of the problem where the execution starts till it finishes without any interruption in between. The machine can execute an unlimited number of jobs at any time. For any time t, whenever there exists a job executed in t, we denote t as "active time". Otherwise we refer time t as "idle time". The objective is to schedule all jobs in $[r_{min}, d_{max}]$ by deciding the starting time s_i and the finishing time f_i for each job j_i, such that the active time is minimized where r_{min} is the earliest release time and d_{max} is the latest deadline of the given jobs.

Besides the general case of the problem, we also study various special cases for this problem with different assumptions on the input jobs which are defined below.

1. Agreeable deadlines: for any two jobs, we have $d_i \leq d_j$ if $r_i < r_j$.
2. Large jobs: for each job j_i, $p_i = (d_i - r_i)/2$.

3 Our results

We solve the active time minimization problem by providing an $O(n^4)$ dynamic programming algorithm to compute the optimal schedule for general jobs which improves the result in [7]. We present an $O(n^3)$ dynamic programming algorithm for the case of agreeable deadlines where $d_i \leq d_j$ if $r_i < r_j$ or for big jobs. Besides off-line cases, we extend the algorithm proposed by [7] to an online algorithm with competitive ratio 2.

In the general case, our objective is to find a feasible schedule such that all jobs can be scheduled within $[r_{min}, d_{max}]$ with maximum number of idle time slots. The high level idea is to properly fix an interval $[A, B]$ for the job with the longest processing time, so that the time interval can be divided into two sub-intervals where the problem

in the partitioned two sub-intervals can be dealt with separately. Then we use dynamic programming to compute the solution for interval $[r_{min}, d_{max}]$ which can be obtained by combining the solutions in the partitioned sub-intervals.

4 Future works

It remains an interesting problem whether there exists a faster algorithm for the special case of laminar jobs where the feasible intervals of any two jobs either have no overlap or with one containing the other. An online algorithm with competitive ratio better than 2 or inapproximability results would be another interesting direction.

References

[1] Susanne Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

[2] Hrishikesh Amur, James Cipar, Varun Gupta, Gregory R Ganger, Michael A Kozuch, and Karsten Schwan. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM SoCC*, pages 217–228. ACM, 2010.

[3] Eric Angel, Evripidis Bampis, and Vincent Chau. Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines. *Discrete Applied Mathematics*, 175:1–10, 2014.

[4] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proceedings of the 17th Annual ACM-SIAM SODA*, pages 364–367. Society for Industrial and Applied Mathematics, 2006.

[5] Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial-time algorithms for minimum energy scheduling. *ACM Transactions on Algorithms (TALG)*, 8(3):26, 2012.

[6] Jessica Chang, Harold N Gabow, and Samir Khuller. A model for minimizing active processor time. In *Algorithms–ESA 2012*, pages 289–300. Springer, 2012.

[7] Xiaolin Fang, Hong Gao, Jianzhong Li, and Yingshu Li. Application-aware data collection in wireless sensor networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 1645–1653. IEEE, 2013.

[8] Yoshiro Ikura and Mark Gimple. Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5(2):61–65, 1986.

[9] Sandy Irani and Kirk R Pruhs. Algorithmic problems in power management. *ACM SIGACT News*, 36(2):63–76, 2005.

[10] Arsalan Tavakoli, Aman Kansal, and Suman Nath. On-line sensing task optimization for shared sensors. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 47–57. ACM, 2010.

On the Assignment Problem with a Nearly Monge Matrix and its Applications in Scheduling

C. Weiß (Speaker) * S. Knust † N. V. Shakhlevich * S. Waldherr †

1 Introduction

The linear assignment problem is one of the most extensively studied problems in combinatorial optimization. While it is generally solvable in $O(n^3)$ time, there are a number of important special cases which can be solved faster, see [1]. The famous class of Monge cost matrices is of particular importance from both theoretical and applied viewpoints. For an $n \times n$ Monge matrix $W = (w_{ij})$, every quadruple of entries in rows i, s $(i < s)$ and columns j, t $(j < t)$ satisfies

$$w_{ij} + w_{st} \leq w_{it} + w_{sj}. \tag{1}$$

This property is associated with the French mathematician Gaspard Monge who in 1781 made the observation that efficient transportation from supply points i, s to demand points j, t should use the arcs that do not intercross. Since the transportation problem with a Monge matrix is greedily solvable, using the north-west corner rule, an optimal solution to the assignment problem with a Monge matrix has a diagonal structure. This holds even in the multi-dimensional case, with a generalized Monge condition. For surveys of results on Monge matrices see [2, 3].

Cost matrices satisfying the Monge condition arise in many applications. Some of them give rise to ∞-entries which model forbidden assignments: if $w_{ij} = \infty$, then i cannot be assigned to j. Depending on the position of the ∞-entries, the Monge property may still be satisfied, so that the problem with ∞'s remain greedily solvable. The generalized version of a multi-dimensional problem of such type is studied in [7] which exploits the relationship between multi-dimensional Monge arrays (with and without ∞'s) and submodular functions. Note that the requirement of [7] that the finite entries form a sublattice implies that ∞'s do respect the Monge property.

In general, however, an arbitrary introduction of ∞-entries may destroy the Monge property in a matrix that initially satisfied it. We call an $n \times n$ matrix W *nearly Monge* if the Monge condition holds for all quadruples with finite entries, while it may be violated for quadruples containing ∞-entries. Two typical scenarios that give rise to assignment problems with nearly Monge matrices are related to satellite communication (SC) and synchronous open shop scheduling (SO).

In the basic SC problem, m senders should transmit messages to n receivers. The duration of a transmission from sender s, $1 \leq s \leq m$, to receiver r, $1 \leq r \leq n$, is t_{sr}.

*{mm12cw, N.Shakhlevich}@leeds.ac.uk. School of Computing, University of Leeds, Leeds LS2 9JT, United Kingdom.

†sigrid@informatik.uni-osnabrueck.de, stefan.waldherr@uni-osnabrueck.de. Universität Osnabrück, Informatik, Albrechtstr. 28, 49076 Osnabrück.

Each sender and each receiver can handle at most one message at a time. In order to avoid conflicts, configuration changes are done simultaneously, so that a feasible solution consists of periods in which simultaneous transmissions happen. The duration of every period is given by the time needed to send the longest message. The goal is to minimize the total duration of completing all transmissions. Various versions of this problem have been extensively studied since the 1980s, see [5, 6, 8, 9]. The version with two senders ($m = 2$) corresponds to the two-dimensional assignment problem with weights

$$w_{ij} = \max\{t_{1i}, t_{2j}\}. \tag{2}$$

Here w_{ij} is the the cost of performing two transmissions $1 \to i$ and $2 \to j$ simultaneously. Notice that if the two arrays (t_{1i}) and (t_{2j}) are renumbered in non-decreasing order, then $W = (w_{ij})$ is a Monge matrix. Additional condition $w_{ij} = \infty$ for $i = j$ prohibits simultaneous transmission to the same receiver and makes the resulting cost matrix nearly Monge. The general version of the SC problem with $m > 2$ senders corresponds to the m-dimensional assignment problem with costs satisfying the multi-dimensional nearly Monge condition.

The SO scheduling problem can be seen as a reformulation of the SC problem in scheduling terminology. There are given m machines and n jobs, together with operation durations t_{sr}, where s is the machine index, $1 \le s \le m$, and r is the job index, $1 \le r \le n$. At any time, a machine can process at most one job and a job can be processed by at most one machine. In the synchronous version of open shop, job changes on machines should be done simultaneously, so that a schedule consists of periods with changes happening at the end of every period. If two jobs i and j are to be processed within the same period by machines 1 and 2, then the duration of the period w_{ij} is defined similarly to (2). The objective is to schedule all operations so that the makespan is minimum. Similar to the SC problem, the SO problem can be modelled as the m-dimensional assignment problem with a nearly Monge cost matrix, where ∞-entries prohibit allocation of two operations of the same job to one period. We are not aware of any results for the SO problem; its counterpart of the flow shop type was studied in [10, 11].

Both problems, SC and SO, have a common underlying model known as *max-weight edge coloring* (MEC). The MEC problem has been extensively studied since the 2000s, see, e.g., [4]. Given a weighted graph G and a feasible k-edge-coloring f, the weight of a color $c \in \{1, \dots, k\}$ is defined as $w(c) = \max\{w_e \mid f(e) = c\}$. The goal is to find a feasible edge-coloring of minimum total weight $\sum_{c=1}^{k} w(c)$. The SO and SC problems correspond to the MEC problem defined on a complete bipartite graph $G = K_{m,n}$, which can again be rewritten as an assignment problem with a weight matrix of type (2).

Observe that the subject of our study is a generalization of the problems mentioned above, namely the assignment problem with a nearly Monge matrix, which entries are not necessarily defined by (2). We denote such a problem by $A(W, d, \lambda)$, where W is a d-dimensional nearly Monge cost matrix with at most λ incompatible partners for a fixed index $i_u = i_u^*$ in any position i_v, $v \neq u$.

2 Structural properties, algorithms and complexity

The NP-hardness of the problem $A(W, d, \lambda)$ with an arbitrary d follows from a similar result known for the MEC problem [8].

Theorem 1 *Problem $A(W, d, \lambda)$ is strongly NP-hard for an arbitrary d, even if $\lambda = 1$ and even if there are only three different finite weights in W.*

The main subject of our study is problem $A(W, d, \lambda)$ with a fixed d, which is a typical condition for applied problems. For this problem we establish a so-called "corridor property" that characterizes the structure of an optimal solution. It implies that 1-entries of a solution matrix belong to a corridor of limited width around the main diagonal.

Theorem 2 *For problem $A(W, d, \lambda)$ there exists an optimal solution such that its 1-entries belong to a corridor of width $2(d-1)\lambda$ around the main diagonal.*

Theorem 2 gives rise to an efficient dynamic programming algorithm that solves problem $A(W, d, \lambda)$ in $O(n)$ time, if d and λ are fixed, and in FPT time, if d and λ are parameters. Since the three problems discussed in the introduction are special cases of problem $A(W, d, \lambda)$ with $\lambda = 1$ after sorting is performed in order to achieve the Monge condition for finite entries, our result has the following implications.

Corollary 1 *Problems (1) MEC in a complete bipartite graph $K_{m,n}$, (2) SC with m senders and n receivers and (3) SO with m machines and n jobs are solvable in $O(n \log n)$ time if m (or symmetrically n) is fixed, and in FPT time, if m (or symmetrically n) is viewed as a parameter.*

References

[1] R. E. Burkard, M. Dell'Amico, S. Martello (2009). Assignment Problems. SIAM, Philadelphia, USA.

[2] R. E. Burkard, B. Klinz, R. Rudolf (1996). Perspectives of Monge properties in optimization. Discr. Appl. Math. 70, 95–161.

[3] R. E. Burkard (2007). Monge properties, discrete convexity and applications. Eur. J. of Oper. Res. 176, 1–14.

[4] M. Demange, B. Escoffier, G. Lucarelli, I. Milis, J. Monnot, V. Th. Paschos, D. de Werra (2008). Weighted Edge Coloring. In Combinatorial Optimization and Theoretical Computer Science (ed V. Th. Paschos), ISTE, London.

[5] I. S. Gopal, C. K. Wong (1985). Minimizing the number of switchings in an SS/TDMA system. IEEE T. Commun. 33, 497–501.

[6] A. Kesselman, K. Kogan (2007). Nonpreemtive scheduling of optical switches. IEEE T. Commun. 55, 1212–1219.

[7] M. Queyranne, F. Spieksma, F. Tardella (1998) A general class of greedily solvable linear programs. Math. Oper. Res. 23, 892–908.

[8] F. Rendl (1985). On the complexity of decomposing matrices arising in satellite communication. Oper. Res. Lett. 4, 5–8.

[9] C. C. Ribeiro, M. Minoux, M. C. Penna (1989). An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment. Eur. J. of Oper. Res. 41, 232–239.

[10] B. Soylu, Ö. Kirca, M. Azizoğlu (2007). Flow shop-sequencing problem with synchronous transfers and makespan minimization. Int. J. of Prod. Res. 45, 3311–3331.

[11] S. Waldherr, S. Knust (2015). Complexity results for flow shop problems with synchronous movement. Eur. J. of Oper. Res. 242, 34–44.

Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters

Dorin Maxim (speaker) * Liliana Cucu-Grosjean †

1 Introduction

We consider a system of n synchronous tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ to be scheduled on one processor according to a preemptive fixed-priority task-level scheduling policy. Without loss of generality, we consider that τ_i has a higher priority than τ_j for $i < j$. We denote by $hp(i)$ the set of tasks' indexes with higher priority than τ_i. By synchronous tasks we understand that all tasks are released simultaneously the first time at $t = 0$.

Each task τ_i generates an infinite number of successive jobs $\tau_{i,j}$, with $j = 1, \ldots, \infty$. All jobs are assumed to be independent of other jobs of the same task and those of other tasks.

Each task τ_i is a generalized sporadic task [1] and it is represented by a probabilistic worst case execution time (pWCET) denoted by $\mathcal{C}_i$[1] and by a probabilistic minimal inter-arrival time (pMIT) denoted by $\mathcal{T}_i$.

The probabilistic execution time (pET) of a job of a task describes the probability that the execution time of the job is equal to a given value. A safe pWCET $\mathcal{C}_i$ is an upper bound on the pETs $\mathcal{C}_i^j$, $\forall j$ and it may be described by the relation $\succeq$ as $\mathcal{C}_i \succeq \mathcal{C}_i^j$, $\forall j$. Graphically this means that the CDF of $\mathcal{C}_i$ stays under the CDF of $\mathcal{C}_i^j$, $\forall j$.

Following the same reasoning the probabilistic minimal inter-arrival time (pMIT) denoted by $\mathcal{T}_i$ describes the probabilistic minimal inter-arrival times of all jobs. The probabilistic inter-arrival time (pIT) of a job of a task describes the probability that the job's arrival time occurs at a given value. A safe pMIT $\mathcal{T}_i$ is a bound on the pITs $\mathcal{T}_i^j$, $\forall j$ and it may be described by the relation $\succeq$ as $\mathcal{T}_i^j \succeq \mathcal{T}_i$, $\forall j$. Graphically this means that the CDF of $\mathcal{T}_i$ stays below the CDF of $\mathcal{T}_i^j$, $\forall j$.

Hence, a task τ_i is represented by a tuple $(\mathcal{C}_i, \mathcal{T}_i)$. A job of a task must finish its execution before the arrival of the next job of the same task, i.e., the arrival of a new job represents the deadline of the current job. Thus, the task's deadline may also be represented by a random variable $\mathcal{D}_i$ which has the same distribution as its pMIT, $\mathcal{T}_i$. Alternatively, we can consider the deadline described by a distribution different from the distribution of its pMIT if the system under consideration calls for such model, or the simpler case when the deadline of a task is given as one value. The latter case is probably the most frequent in practice, nevertheless we prefer to propose an analysis as

*dorin@isep.ipp.pt. CISTER Research Center Building. Rua Alfredo Alle, 535. Porto. Portugal.

†liliana.cucu@inria.fr. INRIA Paris-Rocquencourt. Domaine de Voluceau, BP 105. 78153, Le Chesnay. France.

[1] In this paper, we use calligraphic typeface to denote random variables.

general as possible and in the rest of the paper, we consider tasks with implicit deadlines, i.e., having the same distribution as the pMIT.

Our main result is the following:

Theorem 1 *We consider a task system of n tasks with τ_i described by probabilistic $\mathcal{C}_i$ and $\mathcal{T}_i, \forall i \in \{1, 2, \cdots, n\}$. The set is ordered according to the priorities of the tasks and the system is scheduled preemptively on a single processor. The response time distribution $\mathcal{R}_{i,1}$ of the first job of task τ_i (i.e. the job release at the critical instance) is greater than the response time distribution $\mathcal{R}_{i,j}$ of any j^{th} job of task τ_i, $\forall i \in \{1, 2, \cdots, n\}$.*

2 Probabilistic response time analysis

The probabilistic worst case response time (pWCRT) $\mathcal{R}_n$ of a task τ_n in the critical instance is computed by coalescing all the distributions $\mathcal{R}_n^{i,j}$ (called copies) resulted by iteratively solving the following equation:

$$\mathcal{R}_n^{i,j} = (\mathcal{R}_n^{i-1,head} \oplus (\mathcal{R}_n^{i-1,tail} \otimes \mathcal{C}_m^{pr})) \otimes \mathcal{P}_{pr} \tag{1}$$

where:

- $\mathcal{R}_n^{i,j}$ is the j^{th} copy of the response time distribution
- n is the index of the task under analysis;
- i is the current step of the iteration;
- j represents the index of the current value taken into consideration from the pMIT distribution of the preempting task;
- $\mathcal{R}_n^{i-1,head}$ is the part of the distribution that is not affected by the current preemption under consideration;
- $\mathcal{R}_n^{i-1,tail}$ is the part of the distribution that may be affected by the current preemption under consideration;
- m is the index of the higher priority task that is currently taken into account as a preempting task;
- $\mathcal{C}_m^{pr}$ is the execution time distribution of the currently preempting task;
- $\mathcal{P}_{pr}$ is a fake random variable used to scale the j^{th} copy of the response time with the probability of the current value i from the pMIT distribution of the preempting task. This variable has one unique value equal to 0 and its associated probability is equal to the i^{th} probability in the pMIT distribution of the preempting job.

The iterations end when there are no more arrival values $v_{m,i}^j$ of any job i of any higher priority task τ_m that is smaller than any value of the response time distribution at the current step. A stopping condition may be explicitly placed in order to stop the analysis after a desired response time accuracy has been reached. For example, the analysis can be terminated once an accuracy of 10^{-9} has been reached for the response time. In our case, the analysis stops when new arrivals of the preempting tasks are beyond the deadline of the task under analysis, i.e., the type of analysis required for systems where jobs are aborted once they reach their deadline.

Once the jobs' response time distribution is computed, the Deadline Miss Probability is obtained by comparing the response time distribution with that of the deadline, as follows:

$$\mathcal{B}_i = \mathcal{R}_i \ominus \mathcal{D}_i = \mathcal{R}_i \oplus (-\mathcal{D}_i), \tag{2}$$

where the $\ominus$ operator indicates that the values of the distribution are negated.

Note that the analysis can handle any combination of probabilistic and deterministic parameters, and in the case that all parameters are deterministic the returned result is the same as the one provided by the worst case response time analysis in [2]. More details about the analysis can be found in [3].

References

[1] Mok, Al.(1983). *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. Massachusetts Institute of Technology.

[2] Joseph, M. and Pandya, P. K.(1986). *Finding response times in a real-time system*. The Computer Journal 29(5):390–395.

[3] Maxim, D. and Cucu-Grosjean, L.(2013). *Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters*. Proceedings of the 34th IEEE Real-Time Systems Symposium.

A Branch & Price & Cut Approach for Single Machine Scheduling with Raw Material Availability and Setups

Paul Göpfert (Speaker) * Stefan Bock †

1 Introduction

In this talk, we consider a single machine scheduling problem that can be classified by the three field notation of [3] as follows:

$$1|setup, rm, chains, d_j| \sum w_j C_j. \tag{1}$$

The problem was motivated by a real-world decision problem faced by a supplier in an automotive industry. The company operates a highly automated production line consisting of several stations for the assembly of car components. Due to mass customization, a considerable number of different component types is produced by this machine, in what follows, denoted as product versions. Changing from one product version to another induces setup activities on all stations that are affected by a change of the raw materials build into the components. This leads to sequence dependent setup times (*setup*) between jobs of different product versions.

Before a job is released the needed raw materials have to be available (rm). The actual release date of a job in the schedule is therefore determined by the maximum of the makespan for the scheduled predecessors and the earliest point in time that ensures material availability after processing all preceding jobs. All jobs of a single product version have to be produced in EDD-sequence, i. e. in sequence of non-decreasing due dates. This results in a special set of *chain*-based job-precedence relations. As the supplier has to guarantee a timely delivery, due dates (d_j), either given by the end manufacturers or derived from the subsequent production stages, have to be fulfilled and no tardiness is allowed. The objective of the considered scheduling problem pursues the finding of a feasible production schedule that minimizes the total weighted completion time of all jobs.

Since some included subproblems regarding either the presence of sequence dependent family setup times ([4]) or the need for availability of raw materials ([2]) are proven to be $\mathcal{NP}$-hard, this also applies to our extended problem.

2 Integer Programming Formulation

In this section, we give an integer programming model for the problem:

*pgoepfert@winfor.de. Schumpeter School of Business and Economics, Bergische Universität Wuppertal, Gaußstraße 20, 42119 Wuppertal, Germany.

†sbock@winfor.de. Schumpeter School of Business and Economics, Bergische Universität Wuppertal, Gaußstraße 20, 42119 Wuppertal, Germany.

Every job j of the set of jobs J possesses a processing time p_j, a weight w_j and a due date d_j. For every pair of distinct jobs $(i,j) \in J \times J, i \neq j$, the parameter $s_{i,j}$ describes the setup time necessary for switching from job i to job j.

The completion time of a job $j \in J$ is denoted by the continuous but implicitly integral variable $p_j \leq C_j \leq d_j$. The objective function is given by

$$\min \sum_{j \in J} w_j \cdot C_j. \tag{2}$$

Job sequences have to be defined by the sought schedule. For this purpose, binary variables $\pi_{i,j} \in \{0,1\}$ indicate whether job $i \in J$ precedes Job $j \in J$ in a solution, or not. For technical reasons, we set $\forall j \in J : \pi_{j,j} = 1$. For every schedule and any pair of distinct jobs $(i,j) \in J \times J, i \neq j$ it either holds that i is a predecessor of j, or j is a predecessor of i. Thus, we conclude that

$$\pi_{i,j} + \pi_{j,i} = 1 \quad \forall i,j \in J : i < j. \tag{3}$$

All predetermined precedence relations $i \prec j$ of two jobs are defined by setting $\pi_{i,j} = 1$. They are given either by existing chain restrictions for jobs of the same product version or result from time dependencies due to material availability and existing due dates. Fixing a precedence relation leads to a strict sequence of the respective completion times:

$$\pi_{i,j} = 1 \Rightarrow C_i + s_{i,j} + p_j \leq C_j \tag{4}$$

If a precedence relation between two jobs $i,j \in J$ is not predetermined the implications of Formula (4) are defined by two BigM constraints ([5]).

The required amount of units of raw material type $m \in M$ for producing the units of job $j \in J$ is determined by $a_{j,m}$. As the raw material deliveries are known beforehand, we introduce T as a discrete and non-decreasingly ordered set of all delivery times of materials. Furthermore, let $r_{m,\tau}$ be the number of units of material type $m \in M$ that are delivered at time $\tau \in T$, while $R_{m,\tau} = \sum_{\tau' \in T, \tau' \leq \tau} r_{m,\tau'}$ gives the cumulated supply until $\tau \in T$.

Moreover, in order to ensure raw material availability for the production of job $j \in J$, we derive binary variables $\rho_{j,\tau} \in \{0,1\}$. $\rho_{j,\tau}$ indicates whether a production of job j is possible at time $\tau \in T$. Moreover, exactly one material delivery time is selected for every job. Therefore, it holds that

$$\sum_{i \in J} a_{i,m} \cdot \pi_{i,j} \leq \sum_{\tau \in T} R_{m,\tau} \cdot \rho_{j,\tau} \quad \forall m \in M, j \in J \text{ and } \sum_{\tau \in T} \rho_{j,\tau} = 1 \quad \forall j \in J. \tag{5}$$

The completion time of every job $j \in J$ is then lower bounded by

$$\sum_{\tau \in T} \tau \cdot \rho_{j,\tau} + p_j \leq C_j. \tag{6}$$

3 A Branch and Price and Cut Approach

Given any solution to the above model, we define the set of predecessors $P(j)$ of a job $j \in J$ by $P(j) := \{i \in J \mid \pi_{i,j} = 1\}$. Furthermore, for $|P(j)| > 1$ there is exactly one job $i \in P(j) \backslash \{j\}$, that fulfills constraint (4) with minimum slack. We say that i is

the direct predecessor of j. Due to transitivity, for each predefined precedence relation $i \prec j$, every solution has to fulfill $P(i) \subset P(j)$. For each $P(j)$ we can calculate a lower bound $C_j^{LB}(P(j))$ on the completion time C_j of job j. This lower bound is based on constraints (5) and (6) for the raw material availability as well as on the processing times and lower bounds of the needed setup activities for every job in $P(j)$. Given $i \prec j$, the lower bound $C_j^{LB}(P(j))$ on the completion time of j may be strengthened by the consideration of $C_i^{LB}(P(i))$ while deriving a lower bound of the needed time to process the jobs of the set $P(j) \backslash P(i)$. Based on these results, we define the objects that are used as columns in our approach:

Definition 1 *Let $\mathcal{C}$ be a partition of the jobset J into precedence chains. For any chain $C \in \mathcal{C}, C = \{j_1 \prec j_2 \prec \ldots \prec j_{|C|}\}$, we refer to a sequence of precedence sets $P(j_1) \subset P(j_2) \subset \ldots \subset P(j_{|C|})$ as an* **inclusion pattern** *with costs $\sum_{j \in C} w_j \cdot C_j^{LB}$. Additionally, the pattern has to define for every $j \in C$ a direct predecessor i of j as mentioned above.*

The linear programming master (LPM) selects and combines inclusion patterns to obtain a lower bound on the objective value of an optimal schedule. The main constraints are the well known convexity constraints ([1]) and a set covering formulation, that ensures that for every possible position $p = 1, \ldots, |J|$ in a schedule at least one predecessor set with size p is created. Cuts for the LPM are obtained by aggregated versions of constraints (3) as well as by the observation, that the decomposition of any feasible schedule into inclusion patterns yields a Hamiltonian path through the job set with respect to the direct predecessor choices. Branching decisions are deduced from a RLPM solution by identifying constraints that are overfulfilled (3) (Precedence Branching) or are not uniquely defined by direct predecessor choices (Edge Branching).

At the conference, several Branch & Bound algorithms for the finding of inclusion patterns with minimal reduced costs will be presented. Moreover, computational results show under which circumstances our approach outperforms the integer programming formulation described above.

References

[1] G. B. Dantzig and P. Wolfe (1960). Decomposition Principle for Linear Programs *Operations Research*, Vol. 8, pp. 101-111.

[2] A. Girgoriev, M. Holthuijsen and J. van de Klundert (2005). Basic Scheduling Problems with Raw Material Constraints, *Naval Research Logistics*, Vol. 52, pp. 527-535.

[3] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, Vol. 4, pp. 287-326.

[4] M. M. Liaee and H. Emmons (1997). Scheduling families of jobs with setup times, *Int. J. Production Economics*, Vol. 51, pp. 165-176.

[5] V. Roshanaei, A. Azab, H. ElMaraghy (2013). Mathematical modelling and a meta-heuristic for flexible job shop scheduling, *International Journal of Production Research*, Vol. 51, pp. 6247-6274.

Shortest Path with Alternatives for Uniform Arrival Times: Algorithms and Experiments

Tim Nonner (Speaker) * Marco Laumanns †

1 Introduction

Despite the increasing availability of smartphones and real-time information, it is still a common practice, especially in high-frequency public transportation systems, to simply wait for the next arriving suitable bus (or other means of transportation like metros). This holds especially for systems which do not provide exact time-table information, but manage buses instead by the frequency they leave the terminal, e.g. the Dublin bus system [5]. In such a situation, an experienced local traveler should be aware of alternative suitable buses in order to minimize his waiting time by picking the first arriving one. Formally, such a schedule selection process requires to find a trade-off between minimizing the waiting time by selecting a large set of alternatives, and minimizing the consequent travel time by selecting a small set of alternatives with short travel time, in the extreme case the single alternative with shortest travel time. Iterating this process through the whole network leads to an extension of the classical Shortest Path Problem, called *Shortest Path with Alternatives (SPA) Problem*. Datar and Ranade [1] observed that this extension can be solved efficiently in case of Poisson arrival times (with exponentially distributed inter-arrival times) of buses, which makes it practical even for large-scale public transportation systems. In contrast, Nonner showed that general arrival times result in an NP-hard problem [2], even for one-hop networks. Arguably, for systems where buses run with a given frequency, uniform arrival times (with uniformly distributed inter-arrival times) are the most suitable modelling choice, but they lack the nice properties of a memoryless process. In fact, Boyan and Mitzenmacher [3] showed that optimal policies for such a system have a more complicated structure, which might be hard to communicate: in addition to a list of alternatives for each stop, they require additional timing information for the bus picking process.

2 Contributions

We show that an optimal SPA policy/schedule for uniform arrival times can be efficiently computed subject to the constraint that it has the structure implied by Poisson arrival times, thus giving a trade-off between providing a simple policy to execute and a realistic time assessment. Second, we run several experiments to illustrate the benefits of SPA

*tno@zurich.ibm.com. IBM Research - Zurich
†mlm@zurich.ibm.com. IBM Research - Zurich

policies. In fact, we are not aware of any such study, the only related experimental evaluation was done in the context of data delivery in bus networks [4]. We are interested in comparing the following policies: (P1) a classical single shortest path using an exact timetable, (P2) a SPA policy using a post-processed timetable with frequency information, but where we allow only a single alternative at each stop, and (P3) a SPA policy without this restriction. Comparing policies (P1) and (P2) allows us to reason about how efficient frequency based systems are compared to exact time-tables, and comparing policies (P2) and (P3) gives insights in how much we are able to improve by allowing multiple alternatives in such systems. Note that policy (P2) corresponds to a traveler who navigates greedily through the system, waiting at each stop for the single bus with best combined waiting and travel time.

To run our experiments, we build on the increasingly popular *General Transit Format Specification (GTFS)* [6], which allows us to collect timetable information of multiple European capitals [7, 8]: Berlin, Budapest, Dublin, and Oslo. Interestingly, this format allows the specification of frequencies, exactly the information needed for our study. However, probably because the current shortest path computation methods do not benefit from this information, it is hardly ever provided. Even public transportation systems like the one of Dublin, which explicitly mention that their timetables should be interpreted as frequencies rather than exact times, do not make use of this extension. To deal with this lack of available frequency information, we derive it by counting runs-per-hour in standard time-tables, which aligns with the implicit behavior of a sample traveler.

3 Conclusions

Our main conclusions are: (1) frequency-based systems are not much worse than exact systems, at least on average, and (2) allowing multiple alternatives in a SPA policy provides a significant improvement. Specifically, although the average improvement in total travel time is relatively small, we could decrease the waiting time by at least 20% for roughly 25% of considered cases. Thus, policy (P3) is clearly superior to policy (P2). Hence, we think that providing SPA policies would be a natural extension to any public transportation planner. Another advantage of such policies is that they provide backup opportunities in case there are disruptions in the timetable.

References

[1] Mayur Datar and Abhiram G. Ranade. Commuting with delay prone buses. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pages 22–29, 2000.

[2] Tim Nonner. Polynomial-time approximation schemes for shortest path with alternatives. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, pages 755–765, 2012.

[3] Justin Boyan and Michael Mitzenmacher. Improved results for route planning in stochastic transportation. In *Proceedings of the 12th annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 895–902, 2001.

[4] Utku Günay Acer, Paolo Giaccone, David Hay, Giovanni Neglia, and Saed Tarapiah. Timely data delivery in a realistic bus network. *IEEE Transactions on Vehicular Technology*, 61(3):1251–1265, 2012.

[5] http://www.dublinbus.ie/en/, 2015.

[6] https://developers.google.com/transit/gtfs/, 2013.

[7] http://dublinked.com/datastore/datasets/dataset-254.php, 2014.

[8] http://www.gtfs-data-exchange.com/, 2014.

Lower bounds on the running time for scheduling and packing problems

Lin Chen * Klaus Jansen (Speaker) † Felix Land ‡ Kati Land §
Guochuan Zhang ¶

1 Introduction

In classical complexity theory the usual assumption $P \neq NP$ allows us to rule out polynomial time algorithms for many decision and optimization problems. On the other hand, this does not give us (non-polynomial) lower bounds on the running time for such algorithms. A stronger assumption was introduced by Impagliazzo, Paturi, and Zane:

Conjecture (Exponential Time Hypothesis (ETH) [4])**.** There is a positive real δ such that 3-Sat with n variables and m clauses cannot be solved in time $2^{\delta n}(n+m)^{O(1)}$.

The ETH can be used to show lower bounds on the running time of algorithms for other problems by the use of *strong reductions*, i.e. reductions which increase the parameter at most linearly [4]. Using the Sparsification Lemma by Impagliazzo et al. [4], the ETH assumption implies that there is no algorithm for 3-Sat with n variables and m clauses that runs in time $2^{\delta m}(n+m)^{O(1)}$ for a real $\delta > 0$ as well. This allows us to parameterize by the number of clauses.

Known Results. There is a large number of lower bounds based on the ETH, mostly in the area of graph problems. For a good survey of these results and useful techniques we refer to the work of Lokshtanov et al. [10]. For example, there is no $2^{\delta n}$ time algorithm for 3-Coloring, Independent Set, Vertex Cover, and Hamiltonian Path for some sufficiently small $\delta > 0$, unless the ETH fails. These lower bounds together with the fact that there exist algorithms of running time $2^{O(n)}$ gives us some evidence that the ETH is true. There are only few lower bounds known for scheduling and packing problems. Chen et al. [2] showed that precedence constrained scheduling on m machines cannot be solved in time $f(m)|I|^{o(m)}$ (where $|I|$ is the length of the instance), unless the parameterized complexity class $W[1] = FPT$. Kulik and Shachnai [9] proved that there is no PTAS for the 2D Knapsack problem with running time $f(\varepsilon)|I|^{o(\sqrt{1/\varepsilon})}$, unless all problems in SNP are solvable in sub-exponential time. Patrascu and Williams [12] proved using the ETH assumption a lower bound of $n^{o(k)}$ for sized subset sum with n items and cardinality value k.

*lchen@math.tu-berlin.de Department of Mathematics, TU Berlin.
†kj@informatik.uni-kiel.de Department of Computer Science, University of Kiel.
‡fku@informatik.uni-kiel.de Department of Computer Science, University of Kiel.
§kla@informatik.uni-kiel.de Department of Computer Science, University of Kiel.
¶zgc@zju.edu.cn College of Computer Science, Zhejiang University.

2 Lower Bounds for Exact Algorithms

We first focus on proving tight lower bounds on the running time of algorithms for problems related to Subset Sum. The following theorem can be obtained with the observation that the reduction from 3-Sat to Subset Sum given by Wegener [13] is strong.

Theorem 2.1. *[8] The problems Partition, Subset Sum and Knapsack cannot be decided in time $2^{o(n)}|I|^{O(1)}$, unless the ETH fails.*

These bounds are asymptotically tight, as naive algorithms solve the problems in time $2^n|I|^{O(1)}$. We also found algorithms [8] that solve the scheduling problem $P||C_{max}$ and Bin Packing in time $2^{O(n)}|I|^{O(1)}$. Furthermore, we proved that no algorithm can decide Subset Sum, Partition, Knapsack, m-Bin Packing and $Pm||C_{max}$ with $m \geq 2$ fixed in time $2^{o(\sqrt{|I|})}$, unless the ETH fails. These bounds are tight, as there exist algorithms with running time $2^{O(\sqrt{|I|})}$ [11] (with m fixed). In addition we proved a stronger result for $P||C_{max}$ with non-fixed number m of machines.

Theorem 2.2. *[3] For any $\delta > 0$, there is no $2^{O(m^{1/2-\delta}\sqrt{|I|})}$ time exact algorithm for $P||C_{max}$, unless the ETH fails.*

We also showed that a dynamic programming algorithm for $P||C_{max}$ runs in $2^{O(\sqrt{m|I|\log^2 m})}$ time. For further results on other scheduling problems we refer to [8].

3 Lower Bounds for Approximation Schemes

One can also prove lower bounds for polynomial time approximation schemes (PTAS).

Theorem 3.1. *[8] There is no efficient PTAS (EPTAS) for Multiple Knapsack with running time $2^{o(1/\varepsilon)}|I|^{O(1)}$, unless the ETH fails. This bound even holds for $m = 2$ knapsacks of equal capacity and when either all items have the same profit, or the profit of each item equals its size.*

The fastest known EPTAS for Multiple Knapsack has a running time of $2^{O(1/\varepsilon \log^4(1/\varepsilon))} + |I|^{O(1)}$ [7]. Another result is concerned with the 2D Knapsack problem.

Theorem 3.2. *[8] There is no PTAS for 2D Knapsack with running time $n^{o(1/\varepsilon)}|I|^{O(1)}$, unless the ETH fails.*

This lower bound asymptotically matches the best running time $n^{O(1/\varepsilon)}|I|^{O(1)}$ of a known approximation scheme [1]. Further results are related to $P||C_{max}$ on identical machines. Jansen [5] presented an EPTAS for the scheduling problem on identical machines $P||C_{max}$ and uniform machines $Q||C_{max}$ with running time $2^{O(1/\varepsilon^2 \log^3(1/\varepsilon))} + n^{O(1)}$. Here we proved the following lower bound that leaves a small gap between the lower and upper bound.

Theorem 3.3. *[3] For any $\delta > 0$, there is no $2^{O((1/\varepsilon)^{1-\delta})} + n^{O(1)}$ time EPTAS for $P||C_{max}$, unless the ETH fails.*

For a constant number of processors we showed the following result.

Theorem 3.4. *[3] For any $\delta > 0$, there is no $(1/\varepsilon)^{O(m^{1-\delta})} + n^{O(1)}$ time fully PTAS (FPTAS) for $Pm||C_{max}$, unless the ETH fails.*

Jansen and Mastrolilli [6] presented an FPTAS for the more general scheduling problem $Rm||C_{max}$ on unrelated machines, where each job j could have different execution times p_{ij} on different machines. Its running time is $(m/\varepsilon)^{O(m)} + O(n)$. If ε is small enough (e.g. $\varepsilon < 1/m$), the running time can be bounded by $(1/\varepsilon)^{O(m)} + O(n)$ [6].

References

[1] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger, *Approximation algorithms for knapsack problems with cardinality constraints*, European Journal of Operational Research, 123(2), 2000, pp. 333–345.

[2] J. Chen, X. Huang, I. Kanj, and G. Xia, *On the computational hardness based on linear FPT-reductions*, Journal of Combinatorial Optimization, 11 (2006), pp. 231-247.

[3] L. Chen, K. Jansen, and G. Zhang, *On the optimality of approximation schemes for the classical scheduling problem*, ACM-SIAM Symposium on Discrete Algorithms (SODA 2014), pp. 657–668.

[4] R. Impagliazzo, R. Paturi, and F. Zane, *Which problems have strongly exponential complexity?*, Journal of Computer and System Science, 63 (2001), pp. 512–530.

[5] K. Jansen, *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 (2010), pp. 457–485.

[6] K. Jansen and M. Mastrolilli, *Scheduling unrelated parallel machines: linear programming strikes back*. University of Kiel, Technical Report 1004 (2010).

[7] K. Jansen, *A fast approximation scheme for the multiple knapsack problem*, Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2012), LNCS 7147, pp. 313–324.

[8] K. Jansen, F. Land, and K. Land, *Bounding the running time of algorithms for scheduling and packing problems*, Workshop on Algorithms and Data Structures (WADS 2013), LNCS 8037, pp. 439–450

[9] A. Kulik and H. Shachnai, *There is no EPTAS for two-dimensional knapsack*, Information Processing Letters, 110(2010), pp. 707–710.

[10] D. Lokshtanov, D. Marx, and S. Saurabh, *Lower bounds based on the Exponential Time Hypothesis*, Bulletin of the EATCS, 105 (2011), pp. 41–72.

[11] T.E. O'Neil, *Sub-exponential algorithms for 0/1-knapsack and bin packing*, International Conference on Foundations of Computer Science (FCS 2011), pp. 209–214.

[12] M. Patrascu and R. Williams, *On the possibility of faster SAT algorithms*, ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 1065–1075.

[13] I. Wegener, *Complexity Theory: Exploring the Limits of Efficient Algorithms*, Springer, 2003.

Non-Preemptive Scheduling with Setup Times*

Alexander Mäcker (Speaker) † Manuel Malatyali ‡ Sören Riechers §

1 Introduction

We present an approximation algorithm for makespan scheduling on parallel machines that require setups whenever switching from processing jobs of one type to jobs of another type. Our analysis shows that our approach guarantees an approximation factor that can be made arbitrarily close to $\frac{3}{2}$ and the algorithm runs in time polynomial in the number n of jobs, the number m of machines and the number k of types. Particularly, our work differs from previous work on related problems in the fact that we do not allow preemption [1, 2] nor assume that k is constant [3, 4].

Problem Description We consider a model in which there is a set $J = \{1, \dots, n\}$ of n independent jobs that are to be scheduled on m identical machines $M = \{M_1, \dots, M_m\}$. Each job i is available at time 0 and comes with a *size* $p_i \in \mathbb{N}_{>0}$. Additionally, the job set is partitioned into k disjoint classes $C = \{C_1, \dots, C_k\}$. Before a job $j \in C_i$ can be processed on a machine, this machine has to be configured properly and afterward jobs of class C_i can be processed without additional setups until the machine is reconfigured for a class $C_{i'} \neq C_i$. That is, a setup needs to take place before the first job is processed on a machine and whenever the machine switches from processing a job $j \in C_i$ to a job $j' \in C_{i'}$ with $C_i \neq C_{i'}$. Such a setup takes $s \in \mathbb{N}_{>0}$ time units and while setting up a machine, it is blocked and cannot do any processing. Given this setting, the objective is to find a feasible schedule that minimizes the makespan, i.e. the maximum completion time of a job, and does not preempt any job, i.e. once the processing of a job is started on a machine it finishes at this machine without interruption. We define $w(C_i) \coloneqq \sum_{j \in C_i} p_j$ and assume that for all $1 \leq i \leq n$ it holds that $w(C_i) \leq \gamma OPT$ for some constant γ and OPT being the optimal makespan. The processing time of the largest job in a given instance is denoted by $p_{max} \coloneqq \max_{1 \leq i \leq n}(p_i)$.

The considered problem models situations where the preparation of machines for processing jobs requires a non-negligible setup time. Even though these setups depend on the classes of jobs to be processed, we assume the setup time to be equal for all classes. Also, jobs might not be preempted, e.g. because of additional high preemption

*This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

†alexander.maecker@upb.de. Heinz Nixdorf Institute and Computer Science Department, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany.

‡manuel.malatyali@upb.de. Heinz Nixdorf Institute and Computer Science Department, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany.

§soeren.riechers@upb.de. Heinz Nixdorf Institute and Computer Science Department, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany.

costs. Possible examples of problems for which this model is applicable are (1) the processing of jobs on (re-)configurable machines (e.g. Field Programmable Gate Arrays) which only provide functionalities required for certain operations after a suitable setup or (2) a scenario where large tasks (consisting of smaller jobs) have to be scheduled on remote machines and it takes a certain (setup) time to make task-dependent data available on these distributed machines.

2 $(\frac{3}{2}+\varepsilon)$-Approximation Algorithm

We design and analyze an algorithm that runs in time polynomial in n, m and k and computes a solution with an approximation factor of at most $\frac{3}{2}+\varepsilon$, for any $\varepsilon > 0$.

The high-level idea of our algorithm is to first identify a class of schedules that feature a certain structural property, which we call block-schedules. Applying slight modifications to a given instance allows us to find a good schedule within this class, i.e. one whose makespan is not too far away from an optimal one. In a second step, we use the properties of block-schedules and show how to perform rounding of the involved job sizes and grouping of jobs and thereby significantly decrease the size of the search space. Finally, given such a transformed instance, it will be easy to optimize over the class of block-schedules to obtain an approximate solution to any given instance.

Block-Schedules One main ingredient of our approach is the identification of a class of schedules having a simple structure. On the one hand, the optimzation over this class becomes easy and, on the other hand, it always allows us to find a provably good solution if the search is narrowed to this class. Hence, at the heart of our approach are block-schedules, which we define as follows.

Definition 1 *Given an instance I, we call a schedule for I* block-schedule *if for all $1 \leq i \leq m$ the following holds: In the (partial) schedule for the machines $M_1, \ldots, M_i$, there is at most one class of which some but not all jobs are processed on $M_1, \ldots, M_i$.*

By some careful transformations of a given instance, it is possible to show that there always is a block-schedule with makespan at most $\frac{3}{2}OPT$ for the transformed instance, which also directly implies a schedule with the same bound on the makespan for the original instance.

The main advantage of restricting our search for a schedule to block-schedules is its manageable structure. Intuitively, if we construct a schedule by assigning jobs consecutively to the machines $M_1, \ldots, M_m$, then in each step there is at most one class whose processing was started but not yet finished and all jobs of a class are, roughly speaking, processed in a block of machines and not widespread.

Optimization over Block-Schedules Knowing that there always is a good block-schedule, we perform a rounding technique on the involved job sizes to simplify the given instance. Here we have to take care of jobs (and classes) that have a small size (overall workload) in terms of the optimal makespan and ε in order to be able to bound the introduced rounding error suitably. After performing the rounding, we are left with jobs only having a constant number of different sizes. Equally important is the fact that there is only a constant number of different class-types, where classes made up of identical sets of jobs are called to be of the same type.

Concluding, based on the structure of block-schedules and the properties obtained by rounding, we can describe our search space by a graph in which nodes correspond to machine configurations. While machine configurations are often based on jobs (more precisely, job sizes), we describe them in a more abstract way: For each class-type we specify how many classes of this type are processed on the considered machine along with those individual jobs of (at most) one additional class-type that is not completely finished on the machine. At this point, we gain the advantage that we need not assume the number k of classes to be a constant since configurations no longer rely on job sizes combined with their classes, but just on the more abstract class-types. Using a simple search on the constructed graph, which has a size essentially polynomial in m, we are able to obtain a schedule with the claimed approximation guarantee.

Theorem 2 *We can determine a schedule for any instance I with makespan at most*

$$(1+\varepsilon)\min\left\{\frac{3}{2}OPT, OPT + p_{max} - 1\right\}$$

in time polynomial in n, k and m.

References

[1] Monma, C.L., Potts, C.N.: Analysis of Heuristics for Preemptive Parallel Machine Scheduling with Batch Setup Times. Operations Research 41(5), 981–993 (1993)

[2] Schuurman, P., Woeginger, G.J.: Preemptive scheduling with job-dependent setup times. In: Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '99), pp. 759–767. ACM/SIAM, (1999)

[3] Shachnai, H., Tamir, T.: Polynominal time approximation schemes for class-constrained packing problem. In: Jansen, K., Khuller, S. (eds) APPROX 2000. LNCS, vol. 1913, pp. 238–249. Springer, (2000)

[4] Xavier, E.C., Miyazawa, F.K.: A Note on Dual Approximation Algorithms for Class Constrained Bin Packing Problems. RAIRO - Theoretical Informatics and Applications 43(2), 239–248 (2009)

On The Power of One Preemption on Uniform Parallel Machines

Alan J. Soper (Speaker) * Vitaly A. Strusevich †

1 Introduction

In this paper, we perform an analysis of the power of one preemption for scheduling problems on parallel machines.

In parallel machine scheduling, we are given the jobs of the set $N = \{J_1, J_2, \ldots, J_n\}$ and m parallel machines M_1, $M_2, \ldots, M_m$. If a job $J_j \in N$ is processed on machine M_i alone, then its processing time is known to be p_{ij}. There are three main types of scheduling systems with parallel machines: (i) *identical* parallel machines, for which the processing times are machine-independent, i.e., $p_{ij} = p_j$; (ii) *uniform* parallel machines, which have different speeds, so that $p_{ij} = p_j/s_i$, where s_i denotes the *speed* of machine M_i; and (iii) *unrelated* parallel machines, for which the processing time of a job depends on the machine assignment.

In all problems considered in this paper the objective is to minimize the *makespan,* i.e., the maximum completion time. For a schedule S, the makespan is denoted by $C_{\max}(S)$. In a non-preemptive schedule, each job is processed on the machine it is assigned to without interruption. In a preemptive schedule, the processing of a job on a machine can be interrupted at any time and then resumed either on this or on any other machine, provided that the job is not processed on two or more machines at a time. For an instance of a scheduling problem on parallel machines, let S_i^* and S_p^* denote an optimal schedule with at most i preemptions, and an optimal preemptive schedule which uses an unlimited number of preemptions, respectively. We will refer to schedules with an unlimited number of preemptions as simply preemptive. The case $i = 0$ corresponds to a non-preemptive schedule.

The problem of finding an optimal preemptive schedule with at most $i \leq m-2$ preemptions on identical parallel machines is NP-hard [1] and the corresponding problems on uniform or unrelated machines are obviously no easier. The preemptive counterparts for these problems are polynomially solvable, even in the most general settings with unrelated machines. See a focused survey [2] on parallel machine scheduling with the makespan objective for details and references.

*A.J.Soper@gre.ac.uk. Department of Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, Greenwich, London SE10 9LS, U.K.

†V.A.Strusevich@gre.ac.uk. Department of Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, Greenwich, London SE10 9LS, U.K.

2 Power of Preemption: Review

Consider an instance of a scheduling problem to minimize the makespan $C_{\max}$ on m parallel machines (identical, uniform or unrelated). For the corresponding problem, we define the *power of preemption* as the maximum ratio $C_{\max}(S_i^*)/C_{\max}(S_p^*)$ across all instances of the problem at hand. We denote the power of preemption by ρ_m^i. It determines what can be gained regarding the maximum completion time if unlimited preemptions are allowed.

In order to determine the exact value of ρ_m^i for a particular problem and to give the concept some practical meaning, the following should be done:

(i) demonstrate that the inequality

$$\frac{C_{\max}\left(S_i^*\right)}{C_{\max}\left(S_p^*\right)} \leq \rho_m^i \tag{1}$$

holds for all instances of the problem;

(ii) exhibit instances of the problem for which (1) holds as equality, i.e., to show that the value of ρ_m^i is tight; and

(iii) develop a polynomial-time algorithm that finds a heuristic non-preemptive schedule S_i with at most i preemptions such that

$$\frac{C_{\max}\left(S_i^*\right)}{C_{\max}\left(S_p^*\right)} \leq \frac{C_{\max}\left(S_i\right)}{C_{\max}\left(S_p^*\right)} \leq \rho_m^i. \tag{2}$$

If the machines are identical parallel, then it is known that $\rho_m^0 = 2 - 2/\left(m+1\right)$, as independently proved in [1] and [5]. It is shown in [7], that the value of ρ_m^0 can be reduced for some instances that contain jobs with fairly large processing times.

For unrelated parallel machines, a rounding procedure that is attributed to Shmoys and Tardos and reproduced in [6] and [3] finds non-preemptive schedules S_{0p} such that the bound (2) holds for $\rho_m^0 = 4$. This bound is tight, as proved in [3].

According to [10], for m uniform parallel machines $\rho_m^0 = 2 - 1/m$ and the LPT heuristic finds schedules that respect this bound. In [9] the necessary and sufficient conditions under which the global bound of $2 - 1/m$ is tight are given. If the makespan of the optimal preemptive schedule S_p^* is defined by the ratio of the total processing time of $r < m$ longest jobs over the total speed of r fastest machines, it is shown in [9] that the tight bound on the power of preemption ρ_m^0 is $2 - 1/\min\{r, m-r\}$.

For two uniform machines a parametric analysis of the power of preemption ρ_2^0 with respect to the speed of the faster machine is independently performed in [4] and [8]. For $m = 3$, a similar analysis is contained in [8], provided that the machine speeds take at most two values, 1 and $s \geq 1$.

Studies that compare optimal schedules with a limited number of preemptions to optimal preemptive schedules are fewer in number. For identical machines Braun and Schmidt [1] prove that $\rho_m^i = 2 - 2/\left(m\left(i+1\right)+1\right)$, where $0 \leq i \leq m-1$, and that this bound is tight. For uniform machines it is known that at most $2\left(m-1\right)$ preemptions are necessary in an optimal preemptive schedule S_p^*. Jiang et al. in [4] perform a parametric analysis for two uniform machines and show that $\rho_2^1 = \left(2s^2 + s - 1\right)/2s^2$ where $s \geq 1$ is the speed of the fastest machine.

3 Results on Single Preemption

In this paper we focus on the models with a single preemption.

We show that for $m = 2$ uniform machines the problem of finding an optimal schedule S_1^* with a single preemption is polynomially solvable in this paper. If the machines are unrelated, finding schedule S_1^* is NP-hard for $m = 2$.

Regarding the power of one preemption on $m \geq 3$ uniform machines, we derive a global tight bound $\rho_m^1 = 2 - 2/m$.

We also perform a parametric analysis of the power of a single preemption ρ_3^1 for three uniform machines in terms of the machine speeds.

References

[1] O. Braun and G. Schmidt. (2003) Parallel processor scheduling with limited number of preemptions. SIAM Journal on Computing, 32:671–680.

[2] B. Chen (2004) Parallel machine scheduling for early completion. In J. Y.-T. Leung, ed. Handbook of Scheduling: Algorithms, Models and Performance Analysis, Chapman & Hall/CRC, London, pp. 9-175–9-184.

[3] J.R. Correa, M. Skutella and J. Verschae. (2012). The power of preemption on unrelated machines and applications to scheduling orders. Mathematics of Operations Research, 37: 379–398.

[4] Y. Jiang, Z. Weng and J. Hu. (2014) Algorithms with limited number of preemptions for scheduling on parallel machines. Journal of Combinatorial Optimization, 27:711–723.

[5] C.-Y. Lee and V. A. Strusevich (2005). Two-machine shop scheduling with an uncapacitated interstage transporter. IIE Transactions, 37: 725–736.

[6] J-H. Lin and J.S. Vitter. (1992) ε-approximations with minimum packing constraint violation.In: STOC'92 Proceedings of the 24th Annual ACM Symposium on Theory of Computing, ACM: New York, pp. 771–782.

[7] K. Rustogi and V.A. Strusevich (2013). Parallel machine scheduling: Impact of adding extra machines. Operations Research, 61: 1243–1257.

[8] A.J. Soper, and V.A. Strusevich, (2014). Single parameter analysis of power of preemption on two and three uniform machines. *Discrete Optimization,* 12: 26–46.

[9] A. J. Soper and V.A. Strusevich. (2014). Power of Preemption on Uniform Parallel Machines. Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014*),* 28, 392-0402.

[10] G.J. Woeginger (2000). A comment on scheduling on uniform machines under chain-like precedence constraints. Operations Research Letters, 26: 107–109.

The *a priori* Traveling Repairman Problem

Martijn van Ee (Speaker) * René Sitters †

1 Introduction

In the last few decades, a lot of research has been done in stochastic combinatorial optimization. This field is concerned with classical combinatorial optimization problems, but with additional uncertainty in the instance. For example, there are situations where the problem instance changes on a daily basis. Instead of reoptimizing every instance, because it might be impossible or undesirable, one can alternatively choose to pick one solution that will be good on average. This is the setting of *a priori* optimization. Here, we consider *a priori* routing and in particular the *a priori* traveling repairman problem (TRP). An extended version of this abstract was published in [3].

In *a priori* routing, we are given a complete weighted graph $G = (V, E)$ and a probability distribution on subsets of V. Depending on the model, this distribution is given either explicitly or by a sampling oracle. It is assumed that the instances are metric. In the first stage, a tour τ on V has to be constructed. In the second stage, an active set $A \subseteq V$ is revealed, which is the set of vertices to be visited. The second-stage tour τ_A is obtained by shortcutting the first-stage tour over the active set. For each active set, the first-stage tour has a second-stage objective value. The goal is to find a first-stage tour that minimizes the expected cost of the second stage tour. When it is clear form the context, we may refer to this expected second stage cost simply as the expected cost of the solution.

In the literature, several models for the probability distribution over the active sets are used. In the *black-box* model, there is no knowledge on the probability distribution. The only instrument available is a sampling oracle, which gives a sample from the distribution on request. In the *scenario* model, the instance contains an explicit list with active sets and their corresponding probabilities. In the independent decision model, each vertex has its own probability of being active, independent of the other vertices. The special case where all vertex probabilities are equal is called the uniform model.

It was shown by Shmoys and Talwar [5] that you can get a 4-approximation for the *a priori* traveling salesman problem in the independent decision model. This result motivated us to look at the approximability of the *a priori* TRP in the independent decision model.

In the deterministic TRP, also known as the minimum latency problem, we have a complete graph $G = (V, E)$, a metric cost function c over the edges and a root vertex

*m.van.ee@vu.nl. Department of Econometrics and Operations Research, VU University Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands.

†r.a.sitters@vu.nl. Department of Econometrics and Operations Research, VU University Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands.

r. We want to find a tour τ starting at the root which minimizes the sum of latencies. Here, the latency of a vertex v is defined as the length of the path from r to v along τ. In the *a priori* TRP, the goal is to find a first-stage tour which minimizes the expected second-stage sum of latencies. Here, the second-stage sum of latencies for active set A is obtained by shortcutting the first-stage tour over A and summing up the latencies in the second-stage tour. In this paper, we establish a constant-factor approximation for the *a priori* TRP in the uniform model.

2 Algorithm

Our algorithm is based on algorithms for the deterministic TRP [1],[4],[2]. However, the *a priori* setting makes the problem a lot harder to solve. Our algorithm makes use of an (α, β)-TSP-approximator in the *a priori* setting, which is similar to the one introduced in [1].

Definition 1 *An (α, β)-TSP-approximator in the a priori setting will find a tour of expected length at most βL and visiting at least $(1 - \alpha\epsilon)n$ vertices if there exists a tour of expected length L visiting $(1 - \epsilon)n$ vertices.*

The algorithm can be described as follows. Let $L_0 = 2c^U$, where c is a parameter to be determined later and U is a random variable uniformly distributed on $[0, 1]$, and define $L_i = L_0 c^i$. Now for each length L_i, we obtain a tour $T(L_i)$ by applying the (α, β)-TSP-approximator in the *a priori* setting. These tours will then be concatenated, i.e. we first traverse tour $T(L_0)$, then we traverse tour $T(L_1)$ and so on until all vertices are visited, where we shortcut already visited vertices. We output the resulting tour.

Theorem 2 *Given an (α, β)-TSP-approximator in the a priori setting, our algorithm with $c = \mathrm{e}$ is a $(2\mathrm{e}\lceil\alpha\rceil\beta + 1)$-approximation for the a priori traveling repairman problem in the uniform model.*

Note that if $\alpha = 1$, the approximator corresponds to a β-approximation for *a priori* k-TSP, the problem of finding a tour on k vertices of minimum expected length. It turns out that the *a priori* k-TSP is polynomially solvable on trees in the uniform model. Using the theorem above, we obtain an 6.44-approximation for the *a priori* TRP on trees in the uniform model.

For general metrics, we show how to obtain an (α, β)-TSP-approximator for some constants α and β. It turns out that finding such an approximator reduces to approximating certain variations of the tour single-sink rent-or-buy problem (tour SRoB). Here, we are given a complete graph $G = (V, E)$ with a metric cost function c_e on the edges. There is a client at every vertex $j \in V$ with demand d_j. We have to open facilities at some of the vertices and connect the clients to the facilities. Connecting facility i with client j costs $d_j c_e$, where $e = (i, j)$. Further, we need to *buy* edges e at the cost of Mc_e, where $M \geq 1$, which together connect the facilities by a tour. The goal is to minimize the sum of connection cost and tour cost. In the *prize-collecting* variant of tour SRoB, it is allowed to leave a client unconnected at the cost of a penalty π_i. The goal is to minimize the sum of connection cost, tour cost and penalty cost. An (α, β)-approximator for tour SRoB is implied by an approximation algorithm for the prize-collecting version.

Definition 3 *An (α, β)-tour SRoB-approximator will find a tour SRoB-solution of cost at most βL and visiting at least $(1 - \alpha\epsilon)n$ vertices if there exists a tour SRoB-solution of cost L visiting $(1 - \epsilon)n$ vertices.*

Lemma 4 *If there is an α-approximation for prize-collecting tour SRoB, then there is an $(2\alpha, 2\alpha)$-tour SRoB-approximator.*

Lemma 5 *There is a 5.52-approximation for the prize-collecting tour SRoB problem.*

Now, a constant-factor approximation algorithm for the *a priori* TRP in the uniform setting follows from Theorem 2 and the lemma below.

Lemma 6 *If there is an (α, β)-tour SRoB-approximator, then there is an $(\alpha, 3\beta)$-TSP-approximator in the a priori setting.*

Theorem 7 *There is an $O(1)$-approximation for the a priori traveling repairman problem in the uniform model.*

3 Open Problems

There still are many open problems in the field of *a priori* optimization. For the *a priori* TRP we were only able to give a constant-factor approximation in the uniform model and the constant is still large. The problem is wide open in the independent probability and scenario model. In our analysis we used the theory of (α, β)-TSP-approximators. Better approximations may be obtained by using the *a priori* k-TSP. No constant-factor approximation is known for this problem.

References

[1] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 163–171. ACM, 1994.

[2] Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 36–45. IEEE, 2003.

[3] Martijn van Ee and René Sitters. Routing under uncertainty: The a priori traveling repairman problem. In *Approximation and Online Algorithms: 12th International Workshop, Revised Selected Papers*, pages 248–259. Springer, 2015.

[4] Michel X. Goemans and Jon Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82(1-2):111–124, 1998.

[5] David B. Shmoys and Kunal Talwar. A constant approximation algorithm for the a priori traveling salesman problem. In *Proceedings of the 13th International Conference on Integer Programming and Combinatorial Optimization*, pages 331–343. Springer, 2008.

Optimal Scheduling of Chemotherapy Deliveries under Quality of Care, Resources and Ethical Constraints

Renaud De Landtsheer (Speaker) * Yoann Guyot* Christophe Ponsard*
François Roucoux †

1 Background

A clinical pathway is a multi-disciplinary view of the treatment process required by a group of patients presenting the same pathology with a predictable clinical course [2]. While scheduling the activities of the pathway of a single patient is straightforward, scheduling a pool of patients in a clinic with limited resources raises lots of trade-off concerns. Such concerns must not impact the patient quality of care but must also be fair for all patients for ethical reasons. There is also an economic rationale to streamline the treatment of patients, meaning that resources should be used efficiently. In this setting, an optimizing tool can help to organize clinical pathways. Given that the patients flow is continuous and that a number of unforeseen personal or medical events can require treatment to be postponed or adapted, schedules need to be adjusted on the go, with no interference on already confirmed appointments.

Currently, tool support is still lagging behind and often still relies on manual or basic tools such as spreadsheets or scheduling templates [1]. A number of approaches have been put forward but are never fully coping with the full picture, especially taking explicitly the quality of care as part of the problem.

2 Scope of our research

The aim of this paper is to show that the scheduling of appointments in clinical pathways should be strongly driven by care quality indicators to ensure that all patient schedules remain compatible with optimal care while ensuring optimal resource allocation.

As a case study, we have focused our efforts on the scheduling of chemotherapy pathways in oncology. The timing of chemotherapy administration has a critical impact on the efficiency of the treatment. If a minimal interval between drugs delivery is not enforced, the patient can be severely hurt because chemotherapy drugs are toxic agents, and the body needs some time to recover between drugs administrations. If chemotherapy administrations are spaced too much, the efficiency of the treatment is decreased because cancerous cells left have more chances to multiply again. In order to measure the quality of care, a quantifiable indicator called the "relative dose intensity" (RDI)

*{rdl,yg,cp}@cetic.be. CETIC Research Centre, Gosselies, Belgium.
†francois.roucoux@uclouvain.be. UCL/King Albert II Cancer Institute, Brussels, Belgium.
This work was partly funded by the PIPAS (1017087) and the eHealth4Citizen projects.

[5] was defined. It captures both the respect of the required dose and timing on a scale from 0% (total failure) to 100 % (total conformance). Medical literature has shown, for a number of cancers, that the life expectancy is strongly correlated with the RDI, for instance, for breast cancer, a key threshold value is 85 % [7]. Hence this indicator can be seen has a gauge that should be carefully managed across the whole clinical pathway.

This paper does not focus on the technical aspects of the scheduling engine, but rather on the impact of different choices related to the definition of the optimization problem. This is critical because solving an inadequate problem might have dramatic effects for patients' clinical outcome.

3 Method

An Agile prototype-based approach was applied to experiment with the feasibility of a smart appointment scheduling maximizing chemotherapy efficiency while matching available resources. We started with a simplified model combining the chemotherapy workflows models resulting from a rigorous analysis process [3], resource constraints and possible interfering events. An appointment scheduler was developed along with key companion tools such as a scenario repository, a graphical interface for managing appointments and a simulator of patient-related events. This greatly made easier the validation with oncology practitioners that occurred once a month and involved three hospitals (UCL/Cancer Institute, Grand Hospital of Charleroi and UZ Leuven). The following features were progressively addressed to reach a model that is now realistic enough to consider on-site validation :

- simple resource models, expressed in bed/nurse hours evolving to a finer grained model where each nurse/bed is explicitly allocated.
- service opening days and hours.
- treatment plans, modeled as sequences of steps (day of cure, resting periods) with dose, duration and involved resources.
- constraints on treatment plan instances: earliest/latest start date, patient unavailability, set appointment (past or confirmed).

A strong requirement was to cope with large patient sets, typically involving hundreds of patients simultaneously at various stage of their clinical pathways. For handling such scales, we used local search-based approaches, mainly *iFlatRelax* for scheduling, and in a later phase, *BinPacking* for day level reasoning. Our prototypes were implemented on the CBLS engine of OscaR [4, 6]. The scheduler is working as a background service which is constantly trying to improve the solution in the open future (i.e. beyond all confirmed appointments).

4 Results and Validation

The objective function to maximize is the RDI over the pool of patients. We have developed two global criteria. A first criterion was to maximize the minimal RDI among all patients. It is implemented by minimizing the make span among all patients using *iFlatRelax*. The schedule of a patient is an interleaving of appointments and resting period, followed by a "stub" activity at the end. This stub is needed because all patients do not start their treatment at the same time. A second criterion was to maximize the summed RDI. This was solved using task swapping neighborhood starting from a solution

provided by *iFlatRelax* because it was tightly packed and computed very quickly. Our prototype is able to schedule chemotherapy appointments over roughly five hundred of patients in a few seconds and supports interactive adjustments.

The simulator helped conducting validation, with the ability to step inside the processes and understand the system behavior over long periods and under stressed conditions. The feedback from doctor is rather positive about our contribution to ensure both quality of care and the smarter use of resources. The prototype raised ethical concerns, such as the capacity of the tool to deciding about trade-off among patients under resource shortage. Our conclusion is that the system should report such situations ahead of time to allow the team to take corrective measures, like a transient increase of staffing. In order to keep the medical team in control, the developed graphical display was also a huge practical improvement. Some interesting feature such as the visualization of allocation windows ensuring a good RDI level definitely helped oncologists and nurses in charge of appointments updates.

5 Conclusions and Perspectives

By combining the use of a scalable Open Source CBLS scheduling technology with visualization and simulation components, we were able to show the feasibility of quality indicator-driven scheduling of a large pool of patients. However, at some point, the following provocative question for OR practitioner was raised: ***is it really a good idea to install an optimizing engine in such a critical setting?*** As usual the answer is not in the technology but in the way it is used and controlled. Clinical pathways involve an intricate decision making process and our experience shows that the scheduler can definitely support the medical actors in their work.

Our next step is to conduct on-site validation based on an extended prototype. A major request is to achieve finer tasks management, i.e. within each day. This requires some further work to integrate the *BinPacking* solver into the scheduling engine.

References

[1] Z. Ahmed, Elmekkawy T., and Bates S. Developing an efficient scheduling template of a chemotherapy treatment unit: A case study. *Australas Med J.*, 4(10):575–88, Oct 2011.

[2] H. Campbell, R. Hotchkiss, N. Bradshaw, and M. Porteous. Integrated care pathways. *British Medical Journal*, pages 133–137, 1998.

[3] Christophe Damas, Bernard Lambeau, and Axel van Lamsweerde. Analyzing critical decision-based processes. *IEEE Trans. Software Eng.*, 40(4):338–365, 2014.

[4] Renaud De Landtsheer and Christophe Ponsard. Oscar.cbls : an open source framework for constraint-based local search. In *Proceedings of ORBEL'27*, 2013.

[5] G.H. Lyman. Impact of chemotherapy dose intensity on cancer patient outcomes. *J Natl Compr Canc Netw*, pages 99–108, Jul 2009.

[6] OscaR Team. OscaR: Operational research in Scala, 2012. Available under the LGPL licence from `https://bitbucket.org/oscarlib/oscar`.

[7] M.J. Piccart, L. Biganzoli, and A. Di Leo. The impact of chemotherapy dose density and dose intensity on breast cancer outcome: what have we learned? *Eur J Cancer.*, 36(Suppl 1), Apr 2000.

Approximation Algorithms for Generalized Plant Location*

Alexander Souza (Speaker) †

1 Introduction

Plant Location is a prominent problem in operations planning and has numerous applications since many economic decisions involve selecting and placing facilities to serve certain demands. Examples include manufacturing plants, storage facilities, depots, warehouses, libraries, fire stations and so on. The budgeted version clearly reflects the willingness of customers to pay for being provided some utility.

In this paper, we consider generalizations of the classical the Plant Location problem and give a unified approach for these. Generalized Plant Location means the following problem: There are m possible plants and n customers. Customer j, say, has a *demand* of d_j units of some utility. Each plant i can be *constructed* at cost c_i. If a constructed plant i *serves* a customer j, it produces $u_{i,j}$ units of the *utility* at *service cost* $s_{i,j}$. The goal is to serve the demand of each customer at minimal total cost. In Budgeted Plant Location, each customer j additionally has a *budget* b_j. If a constructed plant i serves a customer j, the budget is charged an amount of $a_{i,j}$ and we require that the total charge can not be more than b_j. In the classical Plant Location problem we have $d_j = 1$ and $u_{i,j} = 1$ for all i and j. That is, it is only required that each customer is served by at least one facility. Observe that we obtain the Set Cover problem for $d_j = 1$, $s_{i,j} = 0$ and $u_{i,j} \in \{0, 1\}$. In our version, we have demands d_j and customer-dependent utility production $u_{i,j}$.

We give a unified randomized algorithm which is $O(\log n)$-approximate for both versions of the problem. However, in the budgeted version, we will violate the budget by a factor of at most $O(\log n)$. Our approach is based on LP-relaxations of the problems strengthened by additional valid inequalities, so-called Knapsack Cover inequalities introduced by Carr et al. [7]. To the best of our knowledge, there are no approximation algorithms known for Generalized Plant Location and Budgeted Plant Location. Since Set Cover can not be approximated better than $(1 - o(1)) \cdot \ln n$, see Feige [9], this is best possible up to a constant factor, unless P = NP.

2 Algorithms for Generalized Plant Location

To the best of our knowledge, there is no approximation algorithm known for the Generalized Plant Location problem. However, there are approximations for special cases and relaxations, e.g., [8, 10, 12, 4]. We close this gap by giving a randomized expected $(4 + \varepsilon) \cdot \ln n$-approximation algorithm for any $\varepsilon > 0$ for the general case. By

*The paper appeared earlier in Mathematical Foundations of Computer Science (MFCS '13)

†alexander.souza@quintiq.com. Quintiq GmbH, Bahnhofstrasse 100, CH-8001 Zurich, Switzerland.

inapproximability of Set Cover [9] this is best possible up to a constant factor, unless P = NP.

Theorem 1 *Let $c \in (0,1]$. There is an algorithm which runs in polynomial time and returns an expected $1/c$-approximate solution, which is feasible for* Generalized Plant Location *with probability at least $1 - 2n\exp(-(1-c)^2/4c)$.*

Corollary 1 *Choose $c = 1/((4+\varepsilon)\cdot \ln n)$ for any $\varepsilon > 0$ and the algorithm is expected $(4+\varepsilon)\cdot \ln n$-approximate and feasible with high probability.*

The notion *high probability* refers to probability converging to one as n tends to infinity. The two main technical ingredients of our algorithm are:

(I) We give an LP-relaxation of the Generalized Plant Location problem strengthened by Knapsack Cover valid inequalities [7]. As stated above, these valid inequalities have proved useful for several other (covering) problems, e.g., [11, 2, 3, 1, 6]. This strengthened formulation allows us to round a fractional solution randomly thus yielding logarithmic integrality gap.

(II) The second tool we use (for the analysis of the rounding) is Bernstein's inequality [5], which is a Chernoff-type bound on the concentration of measure of sums of independent random variables. A property of this bound is that it depends on the variance of the random sum under investigation and that the absolute values of the summands can be arbitrary constants. The dependence on the variance is useful: We use the Knapsack Cover inequalities to derive a rounding scheme in which some "large" variables are rounded deterministically and the other "small" ones are rounded randomly. For the latter we can prove that the variance of the associated sums is not "too large" and we derive a sufficiently strong bound on the concentration of measure.

We obtain the algorithm as follows: First we consider the case $n = 1$, in which the Generalized Plant Location problem becomes the Knapsack Cover problem. As a first step, we give a randomized rounding algorithm for this special case. Then we observe that the Generalized Plant Location problem can be seen as n simultaneous Knapsack Cover problems with the additional requirement of constructing plants. By solving all of these problems feasibly with high probability we obtain which plant shall serve which customer. Having these decisions made, we construct a plant if it serves some customer. With the help of the ingredients (I) and (II) we are able to prove that the algorithm is expected $(4+\varepsilon)\cdot \ln n$-approximate.

Then we turn our attention to Budgeted Plant Location. It is not hard to see that it is already NP-complete to even decide if there exists a *feasible* solution. To overcome this difficulty we will consider a relaxation and allow that these constraints be violated somewhat. More precisely, a solution is *λ-feasible* if the budget constraints are violated by at most a factor of λ and the remaining constraints are satisfied.

We find that the same rounding algorithm, which simply uses an adjusted LP-relaxation, also produces an expected $(4+\varepsilon)\cdot \ln n$-approximate solution, which violates the budgets by factors of at most $2\cdot(4+\varepsilon)\cdot \ln n$ with high probability.

Theorem 2 *Let $c \in (0,1]$. There is an algorithm which runs in polynomial time and returns an expected $1/c$-approximate solution, which is $2/c$-feasible for* Budgeted Plant Location *with probability at least $1 - 4n\exp(-(1-c)^2/4c)$.*

Corollary 2 *Choose* $c = 1/((4+\varepsilon)\cdot \ln n)$ *for any* $\varepsilon > 0$ *and the algorithm is expected* $(4+\varepsilon)\cdot \ln n$*-approximate and* $2\cdot(4+\varepsilon)\cdot \ln n$*-feasible with high probability.*

References

[1] Bansal, N., Buchbinder, N., and Naor, J. Randomized competitive algorithms for generalized caching. *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC '08)* (2008), 235 – 244.

[2] Bansal, N., Gupta, A., and Krishnaswamy, R. A constant factor approximation algorithm for generalized min-sum set cover. *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA '10)* (2010), 1539–1545.

[3] Bansal, N., and Pruhs, K. The geometry of scheduling. *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS '10)* (2010), 407 – 414.

[4] Bar-Ilan, J., Kortsarz, G., and Peleg, D. Generalized submodular cover problems and applications. *Theoretical Computer Science 250* (2001), 179 – 200.

[5] Bernstein, S. N. On a modification of chebyshevs inequality and of the error formula of laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math. 1 4*, 5 (1924).

[6] Carnes, T., and Shmoys, D. Primal-dual schema for capacitated covering problems. *13th MPS Conference on Integer Programming and Combinatorial Optimization (IPCO '08)* (2008), 288 – 302.

[7] Carr, R. D., Fleischer, L. K., Leung, V. J., and Phillips, C. A. Strengthening integrality gaps for capacitated network design and covering problems. *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms (SODA '00)* (2000), 106 – 115.

[8] Chvatal, V. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research 4*, 3 (1979), 233 – 235.

[9] Feige, U. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM 45* (1998), 634–652.

[10] Hochbaum, D. S. Heuristics for the fixed cost median problem. *Mathematical Programming 22* (1982), 148–162.

[11] Kolliopoulos, S. G., and Young, N. E. Tight approximation results for general covering integer programs. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS '01)* (2001), 522 – 528.

[12] Mahdian, M., Ye, Y., and Zhang, J. Approximation algorithms for metric facility location problems. *SIAM Journal on Computing 36*, 2 (2006), 411–432.

On the Benefits of a Hierarchical Version of Generalized Processor Sharing

Jasper Vanlerberghe * Joris Walraevens * Aditya Jain †
Tom Maertens * Herwig Bruneel *

1 Objective of Differentiated Packet Scheduling

Modern telecommunication networks must be capable of supporting a wide variety of heterogeneous services having extremely diverse Quality-of-Service (QoS) requirements. Interactive services, for example, tolerate only a minimal delay, whereas data services allow for more delay. This triggered the need for service differentiation in network routers; network traffic is divided into several service classes and some kind of differentiated, class-based transmission scheduling is implemented in these routers. Strict preemptive priority (SP) is the most drastic way to provide service differentiation, but is neither flexible nor fair. From the perspective of one particular class, however, performance cannot be better (worse) than when that class gets the highest (lowest) priority. This best (worst) performance, e.g., in terms of the mean queueing delay of the class, is often easy to calculate.

Let us assume more generally that we have a performance vector, e.g., the vector of the mean queueing delays of n service classes. Then for very general preemptive work-conserving scheduling policies it can be proved that the achievable region of this performance vector is a polyhedron with the $n!$ permutations of SP scheduling as extreme points [1]. An important objective for practitioners may be to construct a scheduling policy that achieves a desired performance vector that is known to be in the achievable performance region.

2 Differentiated Packet Scheduling Mechanisms

A straightforward scheduling is a randomization of SP over idle/busy period cycles [1]. It is easily proven that the performance vector is a weighted version of the performance vectors of SP, with the weights equal to the randomization probabilities. Choosing probabilities to achieve a certain performance vector then follows quite easily.

*{jpvlerbe,jw,tmaerten,hb}@telin.UGent.be. SMACS Research Group, Department of Telecommunications & Information Processing, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium.

†This research has been performed while this author was doing an internship at Ghent University.

This research has been co-funded by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office.

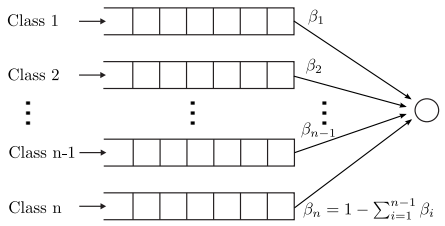


Figure 1: Parekh's GPS scheduling

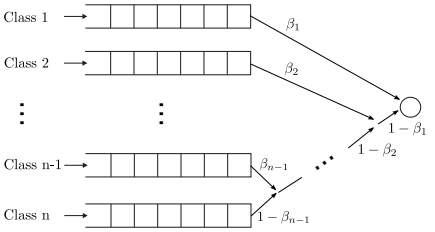


Figure 2: Hierarchical GPS scheduling

However, when the system is highly loaded so that busy periods are long, the desired performance vector is not guaranteed in the short term. A fairer scheduling is Generalized Processor Sharing (GPS). It was developed as an efficient scheduling mechanism providing manageable service differentiation in computer and telecommunication networks [2]. With GPS, each of n traffic classes is given a certain weight β_j, with $\sum_{j=1}^{n} \beta_j = 1$, and the available link capacity is constantly shared according to the weights of the backlogged classes (see Figure 1). GPS assumes that network traffic is fluid. In practice, there are several scheduling mechanisms which attempt to approach the performance of GPS as closely as possible, e.g., Weighted Fair Queueing.

The biggest drawback of all GPS-based scheduling mechanisms is fixing the weights. As opposed to mixing priorities described above, these weights are not equal to the weights in the convex sum of the extreme points. Moreover, easy analytical results for the performance vector in a GPS system are non-existent, even for the simplest of queueing models. Hence, we have to resort to, e.g., Monte-Carlo simulation to estimate the performance vector. Then one way to search for the optimal weights is by means of some iterative procedure, but an extra complexity is that all weights have to be optimized jointly, i.e., changing one weight influences all elements in the performance vector. This all makes the design of an efficient search procedure very hard.

In the current study, we transform the original GPS scheduling to a purely hierarchical version (coined H-GPS) as depicted in Figure 2. In this way, we obtain a scheduling where we can optimize the weights *separately*. We sketch this in the remainder, while we illustrate that the achievable performance region of a union of H-GPS systems equals the achievable performance region of all preemptive work-conserving scheduling policies.

3 Hierarchical GPS

Let us, for the sake of exposition, consider a system with three classes. Define, furthermore, $\bar{w}_j$ as the mean unfinished work of class j in the system. The total unfinished work in the system is independent of the scheduling mechanism as long as this one is work-conserving; so for work-conserving scheduling policies, the mean total unfinished work $\bar{w}_T = \sum_{j=1}^{3} \bar{w}_j$ is a constant. This effectively allows a two-dimensional performance vector $(\bar{w}_1, \bar{w}_2)$. Then we have 6 extreme points in the $(\bar{w}_1, \bar{w}_2)$-plane, corresponding with the 6 priority constellations (black bubbles in Figure 3). The region inside the polygon with these 6 extreme points as vertices is the achievable performance region.

We show that a union of H-GPS systems can achieve all vectors in the achievable region. Assume the H-GPS system as depicted in Figure 2 with three classes and with class 1 at the highest level of the hierarchy. When $\beta_1 = 1$, class 1 has strict priority

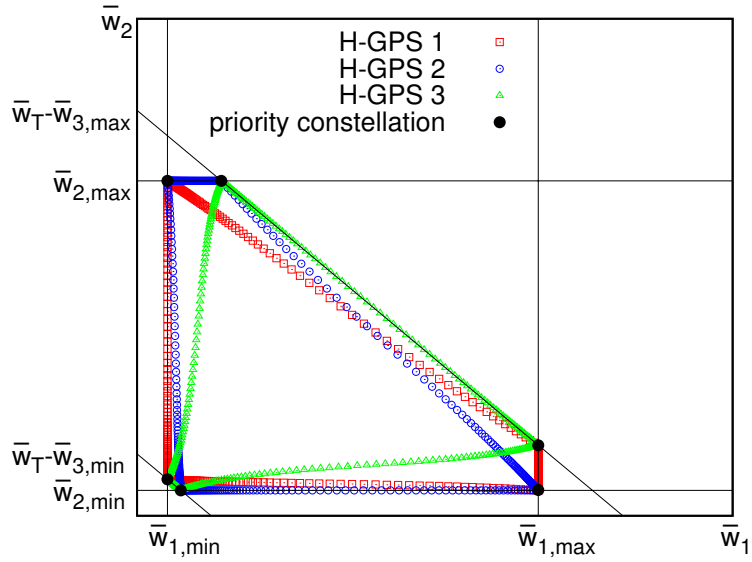


Figure 3: Achievable performance region of three-class GPS scheduling

over classes 2 and 3, and $\bar{w}_1 = \bar{w}_{1,\min}$. It is easy to see that the value of β_2 has no influence on the value of $\bar{w}_1$ when β_1 is kept fixed. By modifying β_2 from 0 to 1 while keeping $\beta_1 = 1$, $\bar{w}_1$ stays $\bar{w}_{1,\min}$ and all points on the line between the extreme points $(1 > 2 > 3)$ and $(1 > 3 > 2)$ are reached. We can do a similar thing for $\beta_1 = 0$ (and $\bar{w}_1 = \bar{w}_{1,\max}$). This gives us 2 edges of the polygon (the vertical lines in Figure 3). Conversely, keeping β_2 constant to 0 or 1 while changing β_1 does not give us straight lines between two extreme points: all performance measures change when β_1 is changed (see Figure 3). With this H-GPS system, all performance vectors in the red surface are achievable. One can easily see that we cannot achieve the complete polygon; specifically, it is not possible to achieve the performance of priority scheduling with class 1 as middle priority. However, we can consider two extra H-GPS systems, one with class 2 and one with class 3 on the highest level of the hierarchy. As can be seen in Figure 3, the union of the achievable performance regions of the 3 systems coincides with the polygon with the 6 extreme points as vertices.

Then a procedure to find the optimal weights follows easily. First, we select a H-GPS system that is able to achieve the desired performance vector. Once this is done, the optimal weights can be searched hierarchically, starting at the highest hierarchy level. This is easy as the value of β_k has no influence on the value of $\bar{w}_j$ for $j < k$ in H-GPS.

References

[1] A. Federgruen, H. Groenevelt, M/G/c queueing systems with multiple customer classes: characterization and control of achievable performance under nonpreemptive priority rules, Management Science 34 (9) (1988) 1121–1138.

[2] A. Parekh, R. Gallager, A generalised processor sharing approach to flow control in integrated services networks: the multi-node case, IEEE/ACM Transactions on Networking 2 (2) (1994) 137–150.

Scheduling of mixed-criticality sporadic task systems with multiple levels

Sanjoy K. Baruah [*] Vincenzo Bonifaci [†]
Gianlorenzo D'Angelo (Speaker) [‡] Haohan Li [§]
Alberto Marchetti-Spaccamela [¶] Suzanne van der Ster [‖] Leen Stougie [‖]

There is an increasing trend in embedded systems towards implementing multiple functionalities upon a single shared computing platform. This can force tasks of different criticality to share a processor and interfere with each other. We focus on the scheduling of sporadic task systems [3] in these mixed-criticality (MC) systems. The mixed-criticality model that we follow has first been proposed and analyzed, for independent collection of jobs, by Baruah et al. [1]. The model has been extended to task systems by Li and Baruah [5]. The results presented here appear in Baruah et al. [2].

We first describe the model and give some notation. Then, we describe an algorithm (called EDF-VD) to preemptively schedule MC task systems on a single machine. We give a sufficient condition for schedulability by EDF-VD and derive a speed-up guarantee.

The model. Given an integer $K \geq 1$, A K-level MC sporadic task system τ consists of a finite collection $(\tau_1, \ldots, \tau_n)$ of MC sporadic tasks. An MC sporadic task τ_i of a K-level system is characterized by a *criticality level* $\chi_i \in \{1, 2, \ldots, K\}$ and a pair $(c_i, d_i) \in \mathbb{Q}_+^{\chi_i} \times \mathbb{Q}_+$, where: $c_i = (c_i(1), c_i(2), \ldots, c_i(K))$ is a vector of *worst-case execution times* (WCET), we assume that $c_i(1) \leq c_i(2) \leq \ldots \leq c_i(\chi_i)$ and $c_i(\chi_i) = c_i(\chi_i + 1) = \ldots = c_i(K)$; d_i is the *relative deadline* of the jobs of τ_i. We consider *impicit-deadline* tasks in which d_i is equal to the *minimum interarrival time* between two jobs of task τ_i. The *utilization* of task τ_i at level k is defined as $u_i(k) := \frac{c_i(k)}{d_i}$, $i = 1, \ldots, n$, $k = 1, \ldots, K$. The total utilization at level k of tasks that are of criticality level l is $U_l(k) := \sum_{1 \leq i \leq n, \chi_i = l} u_i(k)$, $l = 1, \ldots, K$, $k = 1, \ldots, l$. Task τ_i generates a sequence of jobs $(J_{i1}, J_{i2}, \ldots)$. An MC job J_{ij} of task τ_i is characterized by two parameters: $J_{ij} = (a_{ij}, \gamma_{ij})$, where: $a_{ij} \in \mathbb{R}_+$ is the *arrival time* of the job; $\gamma_{ij} \in (0, c_i(\chi_i)]$ is the *execution requirement* of the job; the *(absolute) deadline* of job J_{ij} is $d_{ij} := a_{ij} + d_i$. It is important to notice that neither the arrival times nor the execution requirements are known in advance. In particular, the value γ_{ij} is discovered by executing the job until it *signals* that it has completed execution. A collection of arrival times and execution requirements is called a *scenario* for the task system. The criticality level of a scenario is defined as the smallest integer $\ell \leq K$ such that $\gamma_{ij} \leq c_i(\ell)$, for each job J_{ij} of each task τ_i. An

[*] baruah@cs.unc.edu. University of North Carolina, USA.
[†] vincenzo.bonifaci@iasi.cnr.it. IASI – Consiglio Nazionale delle Ricerche, Italy.
[‡] gianlorenzo.dangelo@gssi.infn.it Gran Sasso Science Institute (GSSI), Italy.
[§] lihaohan@cs.unc.edu Google, USA.
[¶] alberto@dis.uniroma1.it Sapienza Università di Roma, Italy.
[‖] suzanne.vander.ster@vu.nl, leen.stougie@cwi.nl Vrije U. Amsterdam, the Netherlands.

(online) algorithm correctly schedules a sporadic task system τ if it is able to schedule *every* job sequence generated by τ such that, if the criticality level of the corresponding scenario is ℓ, then all jobs of level at least ℓ are completed between their release time and deadline.

Algorithm EDF-VD. We consider a variant of the Earliest Deadline First algorithm, EDF-VD (EDF with virtual deadlines). Algorithm EDF-VD consists of an *offline preprocessing* phase and a *run-time scheduling* phase. The first phase is performed prior to run time and executes a schedulability test to determine whether τ can be successfully scheduled by EDF-VD or not. If τ is deemed schedulable, this phase also provides two output values that will serve as input for the run-time scheduling algorithm: an integer parameter k (with $1 \leq k \leq K$); and, for each task τ_i of τ, a parameter $\hat{d}_i \leq d_i$, called *virtual deadline*. The second phase performs the actual run-time scheduling and consists of K variants, called EDF-VD(1), ..., EDF-VD(K). Each of these is related to a different value of the parameter k that was provided by the first phase; that is, at run time, the variant EDF-VD(k) is applied. If the scenario is exhibiting a level smaller than or equal to k, then jobs are scheduled according to EDF with respect to the virtual deadlines $(\hat{d}_i)_{i=1}^n$. As soon as the scenario exhibits a level greater than k, jobs are scheduled according to EDF with respect to the original deadlines $(d_i)_{i=1}^n$. The preprocessing phase is based on the following sufficient condition for schedulability by EDF-VD.

Theorem 1 *Given an implicit-deadline task system τ, if either $\sum_{l=1}^{K} U_l(l) \leq 1$ or, for some k ($1 \leq k < K$), the following condition holds:*

$$1 - \sum_{l=1}^{k} U_l(l) > 0 \quad \text{and} \quad \frac{\sum_{l=k+1}^{K} U_l(k)}{1 - \sum_{l=1}^{k} U_l(l)} \leq \frac{1 - \sum_{l=k+1}^{K} U_l(l)}{\sum_{l=1}^{k} U_l(l)}, \tag{1}$$

then τ can be correctly scheduled by EDF-VD.

Speedup guarantee. The *speedup factor* of a scheduling algorithm A is the smallest real number f such that any task system τ that is feasible on a unit-speed processor is correctly scheduled by A on a speed-f processor. In the following we determine the minimum speedup factor f_K such that any K-level task system that is feasible on an unit-speed processor is correctly scheduled by EDF-VD on a f_K-speed processor. Such problem can be formulated as follows: Find the *largest* q ($q \leq 1$) such that the following implication holds for all $U_l(k)$, $k = 1, 2, \ldots, K$, $l = k, k+1, \ldots, K$:

$$\sum_{l=k}^{K} U_l(k) \leq q \qquad \forall k = 1, 2, \ldots, K \quad \Rightarrow$$

$$\text{either} \quad \sum_{l=1}^{K} U_l(l) \leq 1 \quad \text{or} \quad \exists k \in \{1, 2, \ldots, K-1\} \text{ s.t. } \begin{cases} 1 - \sum_{l=1}^{k} U_l(l) > 0 \quad \text{and} \\ \frac{\sum_{l=k+1}^{K} U_l(k)}{1 - \sum_{l=1}^{k} U_l(l)} \leq \frac{1 - \sum_{l=k+1}^{K} U_l(l)}{\sum_{l=1}^{k} U_l(l)}. \end{cases}$$

Number of levels K	Speedup factor f_K
2	1.3333
3	2.0000
4	2.6180
5	3.0811
6	3.7321
7	4.2361

Number of levels K	Speedup factor f_K
8	4.7913
9	5.3723
10	5.8551
11	6.4641
12	6.9487
13	7.5311

Table 1: Minimum speedup factor for $K \leq 13$ levels

If the largest such value of q is q^*, the speedup factor is then $f_K = 1/q^*$. Equivalently, we want to find the *smallest* q such that the above implication does not hold, that is, the premise is true but the conclusion is false; in other words, the largest value of the speedup for which one can still construct a counterexample. This leads to a non-linear formulation that involves disjunctions, which are typically disallowed by numerical solvers. We prove that solving such formulation is equivalent to finding $q^* := \min_{j=1,2,\ldots,K-1} q_j^*$, where each q_j^* is the solution to the non-linear program whose constraints are multivariate polynomial inequalities in the variables $U_l(k)$ and q_j. As such, it can be solved by a (numerical) global non-linear continuous optimization solver. In this case we used Couenne [4]. Couenne was able to find the optimum for any $K \leq 13$. The resulting speedup factors are reported in Table 1.

Theorem 2 *Let τ be a K-level task system with $2 \leq K \leq 13$. If τ is feasible on a unit-speed processor, then it is correctly scheduled by* EDF-VD *on a processor of speed f_K, where f_K ($\pm 10^{-4}$) is as in Table 1.*

References

[1] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie (2012). *Scheduling real-time mixed-criticality jobs.* IEEE Transactions on Computers, 61(8):1140–1152.

[2] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. *Preemptive Uniprocessor Scheduling of Mixed-Criticality Sporadic Task Systems.* Journal of the ACM. To appear.

[3] S. K. Baruah and J. Goossens (2003). *Scheduling real-time tasks: Algorithms and complexity.* In J. Y.-T. Leung, editor, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, chapter 28. CRC Press.

[4] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter (2009). *Branching and bounds tightening techniques for non-convex MINLP.* Optimization Methods & Software 24, 4–5, pages 597–634.

[5] H. Li and S. K. Baruah (2010). *An algorithm for scheduling certifiable mixed-criticality sporadic task systems.* In Proc. 16th IEEE Real-Time Systems Symposium, pages 183–192.

Time-relaxed sport scheduling

Dries Goossens (Speaker) * Frits C.R. Spieksma †

1 Introduction

Millions of people, all over the world, are enthralled by sports. Athletes competing against each other, both as individuals or in a team format, form a source of inspiration for young generations. At the same time, almost every sports competition needs a schedule, stating who will play whom, when, and where. A good schedule is important, because it has an effect on the outcome of the competition, public attendance, commercial interests, and even the health of the players.

In general, the sport scheduling problem is to construct a schedule of play such that several constraints are satisfied. Some common constraints are place constraints (e.g. a team cannot play a home game on a given day), separation constraints (e.g. there should be sufficient time between two meetings between the same teams), and fairness constraints (e.g. a team should not face all strong opponents consecutively). We refer to Nurmi et al. [3] for an overview of sport scheduling constraints. Typically, the competition is organized as a k-round robin tournament (each player or team faces each opponent k times). Without any further constraints, generating a schedule for a round robin tournament is easy (de Werra [1]). However, as soon as constraints as *team x cannot play team y at time t* are added, the problem becomes NP-complete (Schaerf [4]).

Sport scheduling problems can be divided into two types: temporally constrained and temporally relaxed problems. In temporally constrained sport scheduling problems, no more than the minimum number of rounds (i.e. time periods) required to schedule all matches is available. For tournaments with an even number of teams, this means that each team will play on each round. In temporally relaxed sport scheduling problems, the number of available rounds is strictly larger than what is required for scheduling all matches. In this case, teams do not play on every round. As most professional round robin tournaments are time constrained, it does not come as a surprise that time-relaxed problems received little academic attention (without overlooking Knust [2] and Schönberger et al. [5]). Time-relaxed sport scheduling problems are however quite common in amateur sport leagues. In this contribution, we focus on a problem derived from an amateur indoor football competition.

*dries.goossens@ugent.be. Management Information Science and Operations Management, Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat , 9000 Ghent, Belgium.

†frits.spieksma@kuleuven.be. Operations Research and Business Statistics, Faculty of Business and Economics, K.U.Leuven, Naamsestraat 69, 3000 Leuven, Belgium.

2 Problem description

We consider a competition between a set of teams T, with $|T| = n$. Each team meets each other team twice: once at its home venue, and once at the opponent's venue (i.e. double round robin). There is a set of rounds $S = \{1, 2, ..., |S|\}$, ranging from the first day of the season till the last. All matches should be played within this time frame.
Each team $i \in T$ provides a list of rounds $H_i \subseteq S$ for which their home ground is available. Home games for a team can only be scheduled on time periods from this list. Obviously, if each match is to be scheduled, each team should at least provide as many rounds as it has opponents, i.e. $|H_i| \geqslant n-1$. This list is called the *home game set*. Some teams may have a time slot on the same weekday every other week; other teams may have a more irregular home game set. Each team can also provide a list of dates $A_i \subseteq S$ on which it doesn't want to play a match; we call this list the *forbidden game set*. We assume that $H_i \cap A_i = \emptyset$. A team is not allowed to play twice on the same day, or more than twice in a period of R_{max} days. Finally, there should also be at least m calendar days between two matches featuring the same teams. Notice that it is allowed to meet an opponent for the second time, before all other opponents have been faced once. In summary, we have the following constraints:

- each team plays a home match against each other team exactly once (1)
- home team availabilities H_i are respected (2)
- away team unavailabilities A_i are respected (3)
- at least m days between two matches with the same teams (4)
- each team plays at most one game per round (5)
- each team plays at most 2 games in a period of R_{max} rounds (6)

The goal is to develop a schedule with for each team a balanced spread of their matches over the season. More in particular, teams wish to avoid having two matches in a period of R_{max} days or less. Having 2 matches in 2 days is considered more unpleasant than having 2 matches in 4 days. The main idea behind this, is that most players prefer not to fully spend their weekend with their sport. Moreover, matches packed together could also lead to injuries. If a team has more than R_{max} days between two consecutive matches, we assume that the league organizers no longer care, and consider any number greater than R_{max} as equally adequate. Constraint (1) is in fact interpreted as a soft constraint, i.e. it is possible not to schedule a match, but only at a high cost. This guarantees feasibility of any instance (e.g. in case some team does not provide enough time slots for which their home ground is available). In practice, if a match cannot be scheduled, the league organizers leave it to the home team to find a suitable date and location to play the match (if the home team fails to organize the match, they lose the game).

3 Results

We develop an integer programming formulation and a tabu search approach, and use them to generate schedules for problem instances from amateur indoor football competitions. The core component of the heuristic consists of solving a transportation problem,

which efficiently schedules (or reschedules) all home games of a team $i \in T$. In a construction phase, we solve the transportation problem sequentially, for each team $i \in T$. The order of the teams is determined by H_i: we start with the team with the lowest number of available home game slots. The end result of the construction phase is a schedule, where some matches possibly remain unscheduled. Next we iteratively pick a team i (while maintaining a tabu list), remove all the home games of this team from the current schedule, and solve the transportation problem for this team. After a number of iterations without improvement, we randomly remove part of the scheduled matches in order to open up some space in the schedule, and continue the tabu search phase. Overall, the quality of solutions obtained with the heuristic is comparable with those resulting from solving the IP formulation with Cplex. For several instances, the heuristic outperforms Cplex both in terms computation time and objective function value. Indeed, Cplex does not manage to solve all instances to (proven) optimality. Evaluating the performance of the heuristic on a number of instances from other (amateur) indoor sports remains interesting future research.

References

[1] D. de Werra (1981) Scheduling in sports. In: Hansen, P. (editor). *Studies on graphs and discrete programming,* Amsterdam:North-Holland, pp.381-395.

[2] S. Knust (2010). Scheduling non-professional table-tennis leagues. *European Journal of Operational Research,* 200:358–367.

[3] K. Nurmi, D. Goossens, T. Bartsch, F. Bonomo, D. Briskorn, G. Duran, J. Kyngas, J. Marenco, C. Ribeiro, F. Spieksma, S. Urrutia and R. Wolf (2010). A framework for a highly constrained sports scheduling problem *Lecture Notes in Engineering and Computer Science,* 2182:1991–1997.

[4] A. Schaerf (1999). Scheduling sport tournaments using constraint logic programming *Constraints,* 4:1991–1997.

[5] J. Schönberger, D. Mattfeld, H. Kopfer (2004). Memetic algorithm timetabling for non-commercial sport leagues. *European Journal of Operational Research,* 153:102–116.

Coordinating Multi-Job Players in Scheduling Games

Fidaa Abed (Speaker) * José R. Correa † Chien-Chung Huang ‡

1 Introduction

In general, scheduling problems can be described as follows. Consider a set $\mathcal{J}$ of n jobs that have to be processed on a set $\mathcal{M}$ of m parallel machines. If processed on machine i, job j requires a certain processing time p_{ij} to be completed. Job j also has a weight w_j. The goal is to find an assignment of jobs to machines, and an ordering within each machine so that a certain objective functions is minimized. Denoting, for any such assignment and ordering, the *completion time* C_j of job j the time at which job j completes, we may write the two most widely studied objectives as $C_{\max} = \max_{j\in\mathcal{J}} C_j$ (the makespan) and $\sum_{j\in\mathcal{J}} w_j C_j$ (the sum of weighted completion times). We will focus on the unrelated machine scheduling problem with the sum of weighted completion times objective, denoted by $R||\sum w_j C_j$.

Since the early work of Smith for the $\sum w_j C_j$ objective, a lot of work has been put in designing *centralized* algorithms providing reasonably close to optimal solutions with limited computational effort for these NP-hard problems. The underlying assumption is that all information is gathered by a single entity which can enforce a particular schedule. However, as distributed environments emerge, understanding scheduling problems where jobs are managed by different selfish agents (players), who are interested in their own completion time, becomes a central question.

Coordination mechanisms. In recent times there has been quite some effort to understand these scheduling games in the special case in which agents control a single job in the system, which we call *single-job games*. In this context, there is a vast amount of work studying existence, uniqueness, the *price of anarchy*, and other characteristics of equilibrium when, given some processing rules, each agent seeks to minimize her own completion time. In the scheduling game each job is a fully informed player wanting to minimize its individual completion time, and its set of strategies correspond to the set of machines. Job j's completion time on a machine depends on the strategies chosen by other players, and on the *policy* (or processing rule) of the chosen machine. While the cost of a job is its weighted completion time, $w_j C_j$. A *coordination mechanism* is then a set of *local policies*, one per machine, specifying how the jobs assigned to that machine are scheduled. In a *local policy* the schedule on a machine depends on the full vector $(p_{1j}, p_{2j}, \ldots, p_{mj})$ and weights w_j of jobs assigned to that machine. In evaluating the efficiency of these policies, one needs a benchmark to compare this social cost against. The definition of the price of anarchy of the induced game considers a social optimum

*fabed@mpi-inf.mpg.de. Max-Planck-Institut für Informatik.
†correa@uchile.cl. Universidad de Chile.
‡huangch@chalmers.se. Chalmers University of Technology.

with respect to the costs specified by the chosen machine policies. However, to measure the quality of a coordination mechanism we consider the worst case ratio of the social cost at an equilibrium to the optimal social cost that could be achieved by the centralized optimization approach. We refer to this as the *coordination ratio* of a mechanism.

In this paper we take a step forward and study *multi-job games*, in which there is a set of agents $\mathcal{A}$ who control arbitrary sets of jobs. Specifically the set of jobs controlled by player $\alpha \in \mathcal{A}$ is denoted by $J(\alpha) \subset \mathcal{J}$ and its cost given a particular schedule is the sum of weighted completion times of its own jobs $\sum_{j \in J(\alpha)} w_j C_j$. As in single-job games, we concentrate on designing coordination mechanisms leading to small coordination ratios, when the social cost is the sum of weighted completion times of all jobs (or equivalently of all agents).

Machine policies. Throughout we assume that policies are prompt: they do not introduce deliberate idle time. In other words, if jobs $j_1, \ldots, j_k$ are assigned to machine i, then by time $\sum_{\ell=1}^{k} p_{ij_\ell}$ all jobs have been completed and released. We distinguish between *nonpreemptive*, *preemptive*, and *randomized* policies. In nonpreemptive policies jobs are processed in some fixed deterministic order that may depend arbitrarily on the set of jobs assigned to the machine (processing time, weight, and ID), and once a job is completed it is released. On the other hand, preemptive policies may suspend a job before it completes in order to execute another job and the suspended job is resumed later. Interestingly, such policies can be considered as nonpreemptive policies, but where jobs may be held back after completion. Finally, randomized policies have the additional power that they can schedule jobs at random according to some distribution depending on the assigned jobs' characteristics. Another usual distinction is between policies that are *anonymous* and *non-anonymous*. In the former jobs with the same characteristics (except for IDs) must be treated equally and thus assigned the same completion time. In the latter, jobs may be distinguished using their IDs.

Equilibrium concepts. For the single-job scheduling game the underlying concept of equilibrium is, quite naturally, that of Nash (NE). However, once we allow players to control many jobs and endow them with the weighted completion time cost, already computing a best response to a given situation may be NP-complete. Therefore, it is rather unlikely that such an equilibrium will be attained. To overcome this difficulty we consider a weaker equilibrium concept, which we call *weak equilibrium* (WE), namely, a schedule of all jobs is a WE if no player $\alpha \in \mathcal{A}$ can find a job $j \in J(\alpha)$ such that moving j to a different machine will strictly decrease her cost $\sum_{j \in J(\alpha)} w_j C_j$. We extend the WE concept to mixed (randomized) strategies by allowing player α to keep the distribution of all but one job $j \in \mathcal{J}(\alpha)$ and move job j to any machine. Observe that in the single-job game NE and WE coincide. Throughout, we provide bounds on the coordination ratio of policies for the weak equilibrium, and since NE are also WE our bounds hold for NE as well.

Our results. We start by considering deterministic policies and prove that the coordination ratio of **sr** under WE is exactly 4. This generalizes the result for single-job games [1] and therefore it is the best possible coordination ratio that can be achieved nonpreemptively. We prove the upper bound of 4 for **sr** with mixed WE. This is relevant since a pure strategy NE may not exist in this setting.

Before designing improved policies we observe that no anonymous policy may obtain a coordination ratio better than 4, and basically no policy, be it preemptive or randomized,

local or strongly local, can achieve a coordination ratio better than 2.618. The latter is in sharp contrast with the case in which players control just one job where better ratios can be achieved with randomized policies [1]. Quite surprisingly we are able to design an "optimal" policy, which we call *externality* (**ex**), that guarantees a coordination ratio of 2.618 for WE. Under this **ex** policy, jobs are processed according to Smith rule but are held back (and not released) for some additional time after completion. This additional time basically equals the negative externality that this particular job imposes over other players. Additionally, we prove that **ex** defines a potential game, so that pure WE exists, and that the convergence time is polynomial. It is worth mentioning that in the single-job game **ex** coincides with the proportional-sharing (**ps**) policy [1], which in turn extends the EQUI policy of the unit-weight case. On the other side, when a single player controls all jobs, **ex** coincides with **sr**.

Interestingly, our result for **ex** in case just one player controls all jobs implies a tight approximation guarantee of 2.618 for local optima under the jump neighborhood for $R||\sum w_j C_j$. This tight guarantee also holds for the *swap* neighborhood, in which one is additionally allowed to swap jobs between machines so long as the objective function value decreases. In addition, our fast convergence result for **ex** implies another new result, namely, that local search with the jump neighborhood, when only maximum gain steps are taken, converges in polynomial time. These facts appear to be quite surprising since, despite the very large amount of work on local search heuristics for scheduling problems, performance guarantees, or polynomial time convergence results are are only known for identical machines.

Methodologically our work is based on the inner product framework of [1], but more is needed to deal with the multi-job environment. Our main contribution is however conceptual: On the one hand, we demonstrate that the natural economic idea of externalities leads to approximately optimal, and in a way best possible, outcomes even in decentralized systems with only partial information (in a full information and centralized setting one can easily design policies leading to optimal outcomes). On the other hand, we provide the first direct application of purely game-theoretic ideas to the analysis of natural and well studied local search heuristics that lead to the currently best known results.

References

[1] R. Cole, J.R. Correa, V. Gkatzelis, V.S. Mirrokni, N. Olver Inner product spaces for MinSum coordination mechanisms. In *STOC* 2011.

Fault tolerant scheduling of non-uniform tasks under resource augmentation

Dariusz R. Kowalski *and* Prudence W.H. Wong (Speaker) * Elli Zavou †

1 Introduction

Dealing with computationally intensive jobs is becoming a necessity rather than an additional advantage of new computational systems. Some of the multiple challenges that appear with the complexity of such systems include the dynamicity of job (or task) arrivals, the diversity of their computational demands (e.g. different processing times), the unpredictable machine failures, as well as the preservation of power consumption.

In this work we focus on the simple model of one single machine prone to unpredictable crashes and restarts, and tasks of sizes $c \in [c_{min}, c_{max}]$ arriving dynamically in the system. Values c_{min} and c_{max} represent the smallest and largest processing times a task may need respectively, when executed by the machine running without additional *resource augmentation*. We consider a parameter s representing *speedup*; the amount of resource augmentation added to the machine, such that the processing time of a task of size c becomes c/s. We apply resource augmentation to overcome the machine failures, as an alternative to using more processing entities (e.g. multiprocessor systems).

Due to the unpredictable nature of the machine and the dynamicity of task arrivals, we consider crash, restart and injection patterns to be controlled by an adversarial entity $\mathcal{A}$, and perform worst-case competitive analysis for the performance of online scheduling algorithms. We focus on two efficiency measures: the *completed time*, which is the aggregate size of all tasks that have been completely executed, and *latency*, which is the longest time a task spends in the system. In some sense, the former corresponds to the *utilization* of the machine, while the latter on the *fairness* of the scheduling algorithm.

In a previous work, Fernández Anta et al. [2] looked at the *pending time* competitiveness of a similar system of multiple machines and showed that in order to achieve competitiveness, it is necessary to use speedup. They proved the NP-hardness of the offline version of the problem and gave lower bounds on speedup, under which no competitiveness can be achieved. These were given by conditions **C1**: $s < \rho$ and **C2**: $s < 1 + \gamma/\rho$, where $\rho = c_{max}/c_{min}$, the ratio of maximum over minimum task sizes, and $\gamma > 0$ a parameter that represents the number of c_{min} tasks that a machine with speedup s can complete in addition to a c_{max} task, in an interval of length $(\gamma + 1)c_{min}$. In a different line of work and environment, Fernández Anta et al. [1] have shown that even with no speedup, an algorithm that gives priority to the shortest tasks can achieve

*D.Kowalski@liverpool.ac.uk and pwong@liverpool.ac.uk. Department of Computer Science, University of Liverpool, L69 3BX Liverpool, UK.

†elli.zavou@imdea.org. Universidad Carlos III de Madrid and IMDEA Networks Institute, 28911 Madrid, Spain. PhD Candidate partially supported by FPU Grant from MECD, Spain.

completed time competitiveness at most $1/(\rho+1)$. Following their line of work, Jurdzinski et al. [3] proposed an algorithm that generalized the results of [1] for a fixed number of different task sizes (more than two), and improved the competitiveness to 1-completed-time-competitiveness, when working with speedup $s = 2$. Another requirement for this algorithm to work, is the divisibility property of the task sizes. We therefore hope to be able to give an algorithm that needs less resource augmentation to achieve 1-completed-time-competitiveness, even if some restrictions apply on the task sizes.

2 Results

Our first result in this work involves the speedup threshold for non-competitiveness. It is summarized in the following theorem, whose proof is based on defining and analyzing two different adversarial strategies (one for each efficiency measure), under which no algorithm can be competitive, either regarding latency or completed time. Roughly speaking, the adversary attempts to force the online algorithm unable to complete the c_{max}-task and hence incurring infinite latency.

Theorem 1 *For any given c_{min}, c_{max} and s, if both conditions* **C1** *and* **C2** *are satisfied, NO deterministic online algorithm is latency competitive, or* 1*-completed-time-competitive when run with speedup s against an adversary that injects tasks of sizes $c \in [c_{min}, c_{max}]$, even in a system with one single machine.*

However, considering the result in [1], we introduce a deterministic scheduling algorithm γ-Burst, for the case of only two task sizes, which achieves both 1-latency-competitiveness and 1-completed-time-competitiveness as soon as condition **C2** does not hold (even if condition **C1** still holds, i.e. $s \in [1+\gamma/\rho, \rho)$). Observe that the speedup required is less than $s = 2$ needed for the algorithm in [3].

Algorithm γ-Burst. It separates the pending tasks in two lists according to their size and sorts them according to their arrival time. This way, the next task to be scheduled from each list, if one of that size is to be scheduled, is the first task (being the one that has been waiting the longest in the system). It then takes its scheduling decisions at the end of each *stage*, which also indicates the beginning of a new one. A stage ends either by being interrupted by a machine crash or by the completion of all the tasks that were decided at the beginning of the stage to be scheduled within the stage. The scheduling decisions are taken based on the following rules:
1. If there are no c_{max} tasks pending, then γ-Burst schedules a c_{min} task.
2. If there are no c_{min} tasks pending, then it schedules a c_{max} task.
3. Else, if there are at least γ tasks of size c_{min} pending, it schedules γ c_{min}-tasks consecutively followed by a c_{max} task.
4. Otherwise, it schedules tasks from the two lists alternatively. In this case, the stage ends after a single task is completed.

Theorem 2 *For any given c_{min}, c_{max} and speedup s satisfying condition* **C1** $\wedge \neg$**C2**, *i.e. $s \in [1+\frac{\gamma}{\rho}, \rho)$, algorithm γ-Burst is* 1*-latency-competitive and* 1*-completed-time-competitive.*

The proof of the results claimed in the theorem above is based on the analysis of latency for each group of task sizes, as well as the exhaustive analysis of the completed time in different types of stages.

References

[1] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R. Kowalski, Joerg Widmer, and Elli Zavou. Measuring the impact of adversarial errors on packet scheduling strategies. In *Structural Information and Communication Complexity - 20th International Colloquium, SIROCCO 2013, Ischia, Italy, July 1-3, 2013, Revised Selected Papers*, pages 261–273, 2013.

[2] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R. Kowalski, and Elli Zavou. Online parallel scheduling of non-uniform tasks: Trading failures for energy. In *Fundamentals of Computation Theory - 19th International Symposium, FCT 2013, Liverpool, UK, August 19-21, 2013. Proceedings*, pages 145–158, 2013.

[3] Tomasz Jurdzinski, and Dariusz R Kowalski, and Krzysztof Lorys. Online packet scheduling under adversarial jamming. In *Approximation and Online Algorithms: 12th International Workshop, WAOA 2014, Wrocław, Poland, September 11-12, 2014, Revised Selected Papers*, volume 8952, page 193. Springer, 2015.

Active and Busy Time Minimization

Jessica Chang * Samir Khuller † Koyel Mukherjee ‡

The active time model involves a collection $\mathcal{J}$ of n jobs that need to be scheduled on a single machine. Each job j has release time r_j, deadline d_j and length p_j. We assume that time is integrally slotted and that job parameters are also integral. The jobs need to be scheduled on the machine in such a way that at most B jobs are running simultaneously. For a job j, we need to schedule p_j units in the window $[r_j, d_j)$ and at most one unit can be scheduled in any time slot. The goal is to minimize the *active time* of the machine, that is, the total duration for which the machine is on. The active time model was first introduced by Chang et al. [2], who showed that in the special case where jobs have unit length, there is a fast optimal algorithm [2], and also developed several other results for other variants. Improved algorithms for non-slotted time was subsequently given by Koehler and Khuller [7].

However, it is not clear how to extend the framework of [2] to the more general case of arbitrary-length jobs. In fact, the existence of efficient active time algorithms depends on the nature of the jobs themselves. If jobs are non-preemptive, it is trivial to show that the packing constraints of this problem make it strongly NP-hard. (Even the feasibility question becomes NP-hard.) The approximability of this setting is open. On the other hand, if preemption at integer boundaries is permitted, then the feasibility question is resolved by a simple network flow computation, as discussed by Chang and Khuller [3] using network flow. Unfortunately, in such a construction, there is no control as to which slot nodes receive flow; to minimize active time suggests a shift from computing a maximum flow to computing a min-edge cost flow, where we wish to send positive flow through as few edges as possible that connect to the sink.

We show that a broad class of natural greedy solutions efficiently approximate the optimal active time to within a factor of 3. In particular, a *feasible solution* is a set of time slots whose activation can support the job set. In other words, there is a way to feasibly assign jobs to just these slots without violating the batch capacity B or job release times or deadlines. W show that a minimal feasible solution gives a solution within thrice the optimal, and this bound is tight. We further improve the approximation ratio by considering a natural IP formulation and rounding its LP relaxation to an integral one that is within twice the integer optimum. Before rounding the fractional solution, we convert a fractional optimal solution to one of equal cost, but with enforced *right-shifted* structure. This will allow us to systematically charge partially open slots to other parts of the fractional solution, without over-charging any single part of the fractional solution by too much. Because the integrality gap is two, the analysis of this algorithm is tight. This presents substantial progress on the problem left open earlier [2]. However, in general the NP-hardness of this problem remains open.

*`jschang@cs.umd.edu`. Department of Computer Science, U. Maryland, College Park MD 20742.
†`samir@cs.umd.edu`. Department of Computer Science, U. Maryland, College Park MD 20742.
‡`koyel.mukherjee@xerox.com`. Xerox Research Centre, Bangalore India.

Busy Time Scheduling: We also consider a related problem that has been studied previously [5, 6], referred to as the *busy time* problem. The main variation from the active time problem is access to an unbounded number machines (e.g., consider the virtual machine setting). We are given a collection $\mathcal{J}$ of n jobs that need to be non-preemptively scheduled on a set of identical machines. Each job j has release time r_j, deadline d_j and length p_j. The jobs need to be partitioned into groups (each group of jobs will be scheduled non-preemptively on a machine) so that at most B jobs are running simultaneously on a given machine. We say that a machine is *busy* at time t if there is at least one job running on the machine at t; otherwise the machine is *idle*. The time intervals during which a machine M is busy is called its *busy time* and we denote its length by *busy(M)*. The objective is to find a feasible schedule assigning jobs to machines (i.e., partitioning jobs into groups) to minimize the cumulative busy time over all machines. Unlike the active time problem, every instance instance is feasible; one can always assign each job to its own machine.

In a well-studied special case of this problem, each job j is "rigid", i.e., $d_j = p_j + r_j$. Since each job's deadline is exactly its release time plus its processing time, there is no question about when it must start. Jobs of this particular form are called *interval jobs*. Interestingly, the busy time problem is still NP-hard even with interval jobs and $B = 2$. What makes the interval jobs case particularly central is that one can convert an instance of the general busy time problem to an instance of interval jobs in polynomial time by solving a dynamic program with unbounded B [6]. The dynamic program "fixes" the positions of the jobs to minimize their *shadow*, i.e., projection onto the time-axis. The magnitude of this shadow lower bounds on the busy time of any feasible solution to the original problem. Then, one can adjust the release times and deadlines to artificially "fix" the position of each job to where it was scheduled in the solution for unbounded B. This creates an instance of interval jobs, on which we can then apply an approximation algorithm for the case of interval jobs and as shown, the proof of FirstFit extends to yield a 4 approximation.

Busy time scheduling in this form was first studied by Flammini et al. [5]. They present a simple greedy algorithm FirstFit for interval jobs and demonstrate that it always produces a solution of busy time at most 4 times that of the optimal solution. The algorithm considers jobs in non-increasing length order, greedily packing each job with the first group in which it fits. In the same paper, they highlight an instance on which the cost of FirstFit is three times that of the optimal solution.

However, unknown to Flammini et al., Alicherry and Bhatia [1] and Kumar and Rudra [8] previously studied a related wavelength assignment problem. Their algorithms yield two different 2-approximations for the busy time problem on interval job instances. We show that the mapping of general jobs to interval jobs [6] can actually increase the cost of the optimal solution for the interval case by a factor of 2 and thus the 2-approximations of Alicherry and Bhatia [1] and Kumar and Rudra [8] for interval jobs can similarly be extended to 4-approximations, and that those bounds are tight.

We develop an improved algorithm with a bound of 3, using a completely different approach. The focus is on the development of a 3-approximation for the interval jobs case. While not the best approximation in the special case, this algorithm implies an improved approximation for the general job case. Given an interval job j, its *span* is the cardinality of the window $[r_j, d_j)$. Then, the span of a subset of interval jobs is the cardinality of the union of job windows. The *central idea* of the GreedyTracking

algorithm is to iteratively identify *subsets of unscheduled jobs whose windows are disjoint.* We are interested in subsets having the maximum span; such subsets can be found easily in polynomial time and are denoted as *tracks.* Then, the set of jobs assigned to a particular machine is the union of B such tracks; we call the set of jobs assigned to the same machine a *bundle* of jobs. The *busy time* of a machine is simply the span of the bundle assigned to it. The goal is to assign jobs to bundles so that at no point on the time axis does a single bundle have more than B jobs, and to do so in a way that minimizes the cumulative busy time.

Intuitively, GreedyTracking succeeds on instances where FirstFit fails because GreedyTracking is less myopic. FirstFit greedily schedules jobs one at a time; GreedyTracking schedules entire subsets of jobs (tracks) at a time. Indeed, GreedyTracking performs optimally on the instance where FirstFit's busy time is thrice the optimal. However, one source of GreedyTracking's suboptimality is in the fact that it ignores alignment when identifying tracks to put into the same bundle. We can construct instances to exploit this weakness, forcing GreedyTracking's busy time to twice the optimal. Closing this gap in GreedyTracking's analysis would be interesting.

One important consequence of GreedyTracking is an improved bound for the busy time problem on general job instances. We first solve the problem assuming unbounded machine capacity B to get a solution that minimizes the projection of the jobs onto the time-axis [6]. This maps the original instance to one of interval jobs, forcing each job to start exactly as it did in the solution for unbounded capacity, and we apply the GreedyTracking algorithm. We prove that in total, this approach has busy time within thrice that of the optimal solution, and the bound is tight.

References

[1] M. Alicherry and R. Bhatia. Line system design and a generalized coloring problem. In *Proceedings of ESA*, pages 19–30, 2003.

[2] J. Chang, H. Gabow, and S. Khuller. A model for minimizing active processor time. In *Proceedings of ESA*, pages 289–300, 2012.

[3] J. Chang and S. Khuller. A min-edge cost framework for capacitated covering problems. In *Proceedings of the 15th Annual ALENEX*, pages 14–25, 2013.

[4] J. Chang, S. Khuller, and K. Mukherjee. LP rounding and combinatorial algorithms for minimizing active and busy time. *SPAA 2014.*

[5] M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. In *Proceedings of the IPDPS*, pages 1–12, 2009.

[6] R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir. Minimizing busy time in multiple machine real-time scheduling. In *Proc. of FST&TCS*, pages 169–180, 2010.

[7] F. Koehler and S. Khuller. Optimal batch schedules for parallel machines. In *Proceedings of WADS*, pages 475–486, 2013.

[8] V. Kumar and A. Rudra. Approximation algorithms for wavelength assignment. In *Proceedings of FST&TCS*, pages 152–163, 2005.

Dominant 1-cycles in circular balanced robotic flow-shops

Florence Thiard (Speaker) Nicolas Catusse Nadia Brauner *

1 Robotic flow-shop

Modern manufacturing systems often include handling resources which need to be taken into account. The robotic cell model addresses this issue. It consists in a flow-shop composed of m machines served by a robotic arm.

As in [2], we consider cells composed of m bufferless machines, denoted by $M_1, M_2, ..., M_m$. The cell is also equipped with an input buffer, which provides the parts to be produced in infinite quantity, and an output buffer, also of infinite capacity. These buffers are modeled by two additional machines, respectively M_0 and M_{m+1}.

Our study is focused on the cyclic production of identical parts with unbounded waiting times at the machines : all parts undergo the same treatment and must be processed successively on machines $M_1, M_2, ..., M_m$. A given part may stay as long as necessary on a machine (unbounded waiting times). Both the robot and the machines can handle only one single part at a time. The travel times between the machines are supposed to be additive.

The objective is to devise the robot move sequence in order to maximize the long run cell's throughput.

2 1-cycles in circular cells

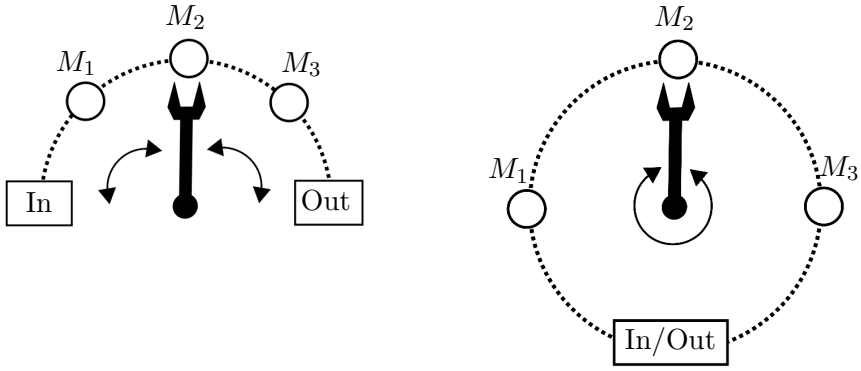
A set of robot move sequences is said to be dominant if, for any value of the parameters, it contains one optimal sequence. Similarly, a subset of sequences is said to be dominant within a set if it contains a sequence at least as performant as all the other sequence of the set. In a cyclic programmation, the robot repeats indefinitely a finite sequence of movements called a cycle, each iteration leaving the cell in the same state. We only consider such sequences, as their dominance has been shown in [6].

1-cycles are production cycles of 1 part exactly : during a single iteration, exactly one part enters the cell, and one processed part leaves the cell. Using the notion of activities, these sequences can easily be described as permutations of $m + 1$ so-called activities [5]. As such, they are easy to express and enumerate. Although 1-cycles are not generally optimal [6, 3], they are easier to apply in practice. Consequently, a common simplification is to limit the set of possible sequences to 1-cycles only. The problem is then to find a set of cycles dominant within 1-cycles.

The answer to this question and its complexity depends heavily on the cell's layout. Two main configurations are studied in the literature : on one hand linear or semi-circular

*<firstname>.<lastname>@g-scop.grenoble-inp.fr. Univ. Grenoble Alpes – CNRS, G-SCOP, F-38000 Grenoble, France

layouts, where the input and output buffer are separated and located respectively at each end of the line [4] (fig 1a), and on the other hand circular layouts, where the machines are arranged in a circle, with the input and output buffers either occupying the same spot, or very close [8, 7] (fig 1b). This potentially changes the travel times between two given machines, as the robot may circle around the cell.



(a) semi-circular layout (b) circular layout

Figure 1: 3 machine robotic cells

For linear layouts, Crama and Van de Klundert [4] have proved the dominance of a family of permutations within 1-cycles, and derived a polynomial algorithm for solving this problem. However, these results do not stand for circular layouts, and Rajapakshe *et al.* [8] showed that in a circular regular cell (with regularly arranged machines), this same problem is NP-hard. The authors also define a family of 1-cycles which provides a $\frac{5}{3}$-approximation.

3 Balanced cells

We study a special case of circular cells, called regular balanced cells. In this configuration, in addition to the machines being regularly disposed, the processing times are machine-independent. An instance of the problem is then specified by only three numbers : the number of machine m, the travel time between two consecutive machines δ and the processing time p. This simpler model allows for an easy representation of the cycle performances depending on the parameters, and highlights the specificity of circular configurations.

In this special case, the complexity of finding the best 1-cycle is still open. The problem is solved for 3 machines in [1]. By computation and cases studies, we show that up to 11-machine cells, the minimum cardinal of a set of cycles dominant over 1-cycles is 3 or 4 (see table 1).

We consider 3 classical 1-cycles, and specifically the so-called odd-even cycle [8] : this cycle takes profit of the circular layout by making the robot circle the cell twice, serving odd machines during the first turn and even machines during the second. We build a new family of 1-cycles using slight perturbations of this sequence, in order to uniformly

m	3	4	5	6	7	8	9	10	11
cardinal	3	4	4	3	3	3	4	4	4

Table 1: Cardinal of a minimum dominant set of cycles

space out subsequent loadings and unloadings of a same machine, which dominates the classical cycles for some parameters. We study the characteristics of this new family and identify some necessary property of dominant 1-cycles. Based on these results and further experimentations, we make the following conjecture :

Conjecture 1 *For circular regular balanced cells, the three classical cycles and this new family define a set of dominant cycles within 1-cycles.*

Proving the validity of this conjecture could lead to a polynomial algorithm for finding the best 1-cycle in a regular balanced cell with circular layout.

References

[1] N. Brauner. *Ordonnancement dans des cellules robotisées.* Thèse de doctorat, Université Joseph Fourier, Grenoble, France, 1999.

[2] N. Brauner. Identical part production in cyclic robotic cells: Concepts, overview and open questions. *Discrete Applied Mathematics*, 156(13):2480–2492, 2008.

[3] N. Brauner and G. Finke. Cycles and permutations in robotic cells. *Mathematical and Computer Modelling*, 34(5-6):565–591, 2001.

[4] Y. Crama and J. van de Klundert. Cyclic scheduling of identical parts in a robotic cell. *Operations Research*, 45(6):952–965, 1997.

[5] Y. Crama and J. van de Klundert. Cyclic scheduling in 3-machine robotic flow shops. *Journal of Scheduling*, 2:35–54, 1999.

[6] M. Dawande, H. N. Geismar, S. P. Sethi, and C. Sriskandarajah. Sequencing and scheduling in robotic cells: Recent developments. *Journal of Scheduling*, 8(5):387–426, 2005.

[7] K. S. Jung, H. N. Geismar, M. Pinedo, and C. Sriskandarajah. Approximations to optimal sequences in single-gripper and dual-gripper robotic cells with circular layouts. *IIE Transactions*, 47(6):634–652, 2015.

[8] T. Rajapakshe, M. Dawande, and C. Sriskandarajah. Quantifying the impact of layout on productivity: An analysis from robotic-cell manufacturing. *Operations Research*, 59(2):440–454, 2011.

Approximability of machine scheduling problems with non-renewable resources

Péter Györgyi (Speaker) * Tamás Kis †

1 Introduction

We study single machine scheduling problems with jobs requiring some non-renewable resources (e.g. raw materials or money). Each resource has an initial stock, which is replenished in known quantities at given dates. A schedule is feasible if no two jobs overlap in time, and when a job is started enough resources are available to cover its requirements. The jobs consume the required resources and the objective is to minimize the makespan.

Scheduling problems with resource consuming jobs were introduced by Carlier [4], and Carlier and Rinnooy Kan [5]. Slowinski [12] and Toker et al. [13] consider different special cases of the problem and provide some polynomial time algorithms for them. Grigoriev at al. [7] derive basic complexity results for several variants. Gafarov et al. [6] developed these results by determining the complexity of other variants. Briskorn et al. [2, 3] studied problems where the objective is to minimize the inventory level.

Györgyi and Kis [9] provide an FPTAS for the problem with one resource and two supplies and a PTAS for the problem with one resource and a constant number of supplies. In [10] some very strong links have been proved between the studied problem and some variants of the knapsack problem. Several approximability results have been obtained as a consequence. In that article a connection to a resource delivery problem (resources are produced rather than consumed) has been established as well. Therefore, some of the approximability results for the problems with resource consuming jobs can be transfered to those with resource producing jobs. More results on the latter problem can be found in Drótos and Kis [8].

When jobs may consume as well as produce non-renewable resources, Kellerer et al. [11] consider the minimization of maximum stock level and propose three different approximation algorithms with relative error 2, 8/5, 3/2. Several complexity results are provided in Briskorn et al. [2], and Boysen et al. [1].

*gyorgyi.peter@sztaki.mta.hu. Department of Operations Research, Loránd Eötvös University, Pázmány Péter sétány 1/C, H1112 Budapest, Hungary and Institute for Computer Science and Control, Kende str. 13-17, H1111 Budapest, Hungary.

†kis.tamas@sztaki.mta.hu. Institute for Computer Science and Control, Kende str. 13-17, H1111 Budapest, Hungary.

2 Achieved results

We have both new inapproximability and approximability results. All of the following results are obtained for the makespan minimization problem with one machine and resource consuming jobs. The negative one is the following:

- If the number of the resources is not a constant (part of the input) then the problem is APX-hard even in case of 2 supply dates. We reduce the APX-complete Vertex Cover Problem in Bounded-degree graphs to our problem.

On the positive side we have shown the following:

- There is a PTAS if the number of the resources, the number of supplies, and the number of distinct job release dates before the last supply are all bounded by a constant. We modeled the problem with an integer program and we reuse some techniques from our previous PTAS ([9]), like dividing the jobs into big and small ones, and schedule them separately. First, we search several, but still a constant number of partial solutions of the IP. Each partial solution essentially gives a schedule of the big jobs. After that we define a residual problem for every partial solution and solve that approximately. We use an LP rounding at this phase. Finally, we piece together the reached results and specify a feasible schedule with a makespan at most $1 + \varepsilon$ times greater than the optimal.

- If the number of the supplies is not bounded by a constant then there is a PTAS for the problem with one resource, if the resource requirement of each job is equal to its processing time. Again we define an IP model of the problem and divide the jobs into big and small ones. Now we examine polynomial number of partial solutions, that essentially meet the big job schedules. We define a new scheduling problem from the residual problem to schedule the small jobs (and may modify some starting times of the big jobs a little) that we solve by a greedy-like algorithm.

  Recall that without the assumption between the resource requirements and processing times, a PTAS is known only if the number of supply dates is a constant.

- We can extend both results by enabling job specific release dates as well.

Acknowledgements

This work has been supported by the OTKA grant K112881. The work of Tamás Kis has been supported by the János Bólyai research grant, no. BO/00412/12/3 of the Hungarian Academy of Sciences.

References

[1] N. Boysen, S. Bock, M. Fliedner (2013). *Scheduling of inventory releasing jobs to satisfy time-varying demand: an analysis of complexity.* Journal of Scheduling, 16, 185–198.

[2] D. Briskorn, B.-C. Choi, K. Lee, J. Leung, M. Pinedo (2010). *Complexity of single machine scheduling subject to nonnegative inventory constraints.* European Journal of Operational Research, 207, 605–619.

[3] D. Briskorn, F. Jaehn, E. Pesch (2013). *Exact algorithms for inventory constrained scheduling on a single machine.* Journal of Scheduling, 16, 105–115.

[4] J. Carlier (1984). *Problèmes d'ordonnancements à contraintes de ressources: Algorithmes et complexité.* Doctorat d'état.

[5] J. Carlier, A.H.G. Rinnooy Kan (1982). *Scheduling subject to nonrenewable resource constraints.* Operational Research Letters, 1, 52–55.

[6] E.R. Gafarov, A.A. Lazarev, F. Werner (2011). *Single machine scheduling problems with financial resource constraints: Some complexity results and properties.* Mathematical Social Sciences, 62, 7–13.

[7] A. Grigoriev, M. Holthuijsen, J. van de Klundert (2005). *Basic scheduling problems with raw material constraints.* Naval Research of Logistics, 52, 527–553.

[8] M. Drótos, T. Kis (2013). *Scheduling of inventory releasing jobs to minimize a regular objective function of delivery times.* Journal of Scheduling, 16, 337–346.

[9] P. Györgyi, T. Kis (2014). *Approximation schemes for single machine scheduling with non-renewable resource constraints.* Journal of Scheduling, 17, 135–144.

[10] P. Györgyi, T. Kis (2015). *Reductions between scheduling problems with non-renewable resources and knapsack problems.* Theoretical Computer Science, Volume 565, 63–76.

[11] H. Kellerer, V. Kotov, F. Rendl, G.J. Woeginger (1998). *The Stock Size Problem.* Operations Research, 46, S1–S12.

[12] R. Slowinski (1984). *Preemptive scheduling of independent jobs on parallel machines subject to financial constraints.* European Journal of Operational Research, 15, 366–373.

[13] A. Toker, S. Kondakci, N. Erkip (1991). *Scheduling under a non-renewable resource constraint.* Journal of the Operational Research Society, 42, 811–814.

University Course Timetabling with Conflict Minimization and Elective Courses

Ernst Althaus * Udo Muttray (Speaker) *

1 Introduction and Problem Definition

Educational timetabling poses a considerable challenge at numerous universities and schools. In addition to the computational complexity of the problem itself, the institutional models vary substantially, as pointed out by McCollum [3].

In this paper, we describe an integer programming approach to a real-life university timetabling case from Germany. In comparison to most timetabling problems presented in the literature, our real-life case differs in four fundamental ways. If the those special features are considered, then it usually does not happen in the context of integer programming techniques.

First, the timetabling instances come from a study program for students who want to become a teacher. As such, they have to choose 2 out of 21 subjects and attend the respective courses. In addition, all students have to take courses from educational sciences. Preassigning courses from educational sciences, this is still roughly equivalent to a timetabling problem with 200 different curricula where each curriculum shares courses with 40 other curricula. In this situation, a conflict-free timetable can rarely be achieved. Instead, the objective must be to minimize the number of conflicts (weighted by the number of affected students). Kiaer and Yellen develop a weighted graph model to describe such problems and use a heuristic algorithm to solve them [2].

Second, courses can be divided into two course types, *compulsory* courses and *elective* courses. Compulsory courses are offered only once, while elective courses are offered at least twice. In our case, the important difference is that students must be able to attend all of the compulsory courses, but only one of at least two offers of each elective course. Müller and Rudová report on the successful implementation of elective courses and course sections into the course timetabling system UniTime, using local search [4].

Third, courses may have special constraints, which might even link courses from different semesters. All special constraints fall into one of the following categories:

(1) *consecutive slots* (only if the course needs more than one slot), (2) *desired conflicts with other courses*, involving courses with the same course type from different subjects or semesters, (3) *no conflicts with other courses*, involving courses with the same course type from the same subject from different semesters, (4) *set of unavailable slots*.

Finally, if not too many courses are scheduled simultaneously, in our case there is always an adequate number of rooms available. Therefore, the assignment of rooms is not considered. Note that this feature actually makes the problem easier to solve.

*{ernst.althaus, udo.muttray}@uni-mainz.de. Institute of Computer Science, Johannes Gutenberg University Mainz, Germany.

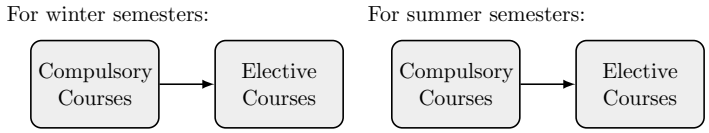


Figure 1: Decomposition based on course type

2 Decomposition-based Integer Programming Model

The main idea is a decomposition, assigning compulsory courses first and elective courses second. Both parts will be formulated and solved as integer programs. As with all decomposition-based models, there is a chance that the first step produces a solution which leads to a (globally) non-optimal solution during the second step. In our case, this seems to be acceptable since there are usually a lot more compulsory courses than elective courses. Furthermore, elective course are sometimes offered more often than it is required by the model, which provides additional flexibility.

As the optimization problems for winter and summer semester are independent, we solve the models for both compulsory and elective courses once for the winter semester and once for the summer semester. On the other hand, the problems for all courses from a winter semester respectively all courses from a summer semester need to be dealt with simultaneously due to the second and third special constraint. As a result, for each timetabling instance we need to perform four computations as shown in Figure 1.

For the compulsory model, we use variables $x_{ck} \in \{0, 1\}$ to describe whether a course c is assigned to slot k or not (the courses determines the semester). A conflict occurs if and only if two courses c and c' from the same curriculum (i.e. a combination of subjects (i, j)) must be taken in the same semester and are assigned to the same slot. Instead of linearizing products $x_{ck} \cdot x_{c'k}$ indicating a conflict, we introduce additional (exponentially many) variables $h^s_{Lk} \in \{0, 1\}$ for each subset of subjects L, slot k and semester s, and link them to the respective course variables x_{ck}. For the set variables, we made the heuristic assumption that no slot will have more than a given number n of different subjects. This assumption was supported by preliminary analysis without taking the special constraints into account, comparing weighted conflicts between both models. It will additionally contribute to the room distribution.

A variable h^s_{Lk} takes value 1 if and only if exactly the subjects in L have a course in slot k in semester s. Denoting the number of induced weighted conflicts for a subset of subjects L by θ_L, this leads to an easy computation of the objective function:

$$\min \sum_{s,k,L} \theta_L \cdot h^s_{Lk}.$$

Analogously, Cacchiani et al. propose models with exponentially many variables, which result in equally elegant expressions for the objective functions [1].

For the elective model, we introduce two *elective blocks* and require that each elective course is offered at least once in each elective block. Technically speaking, for each semester s and combination of subjects (i, j), there must exist a valid choice of elective blocks $(t, u) \in T^2 = \{1, 2\}^2$ such that students can attend the elective courses in block t for subject i and in block u for subject j without further conflicts between either the two chosen elective blocks or a chosen elective block with previously assigned compulsory

courses. Since such a choice may not exist, we allow additional conflicts from the elective courses and minimize their weighted number.

The decomposition and the models were implemented in Java 8, using Gurobi 6.0.0 as the solver. Experiments were performed on a 64-bit Linux system, running on an Intel Core i7-5820K CPU (6×3.3 GHz) with 32 GB of RAM. For the compulsory courses, a size limit of $n = 4$ for the subject sets L and a time limit of 24 hours were laid down.

The model was tested on the real-life instance as well as on 10 randomly generated, artificial instances. Runtimes of the compulsory model ranged from 3 minutes to 14 hours. For the elective courses, runtimes did not exceed 10 seconds.

To reduce variability from the solver, optimizations were executed twice for each planning semester in each instance, using solver seeds 0 and 1. In one case, the model for the compulsory courses could not be solved. Taking the better one of the two results, in 19 out of 21 cases the elective courses did not produce further conflicts, proving optimality of the solution with respect to the decomposition. For all instances, subsequent experiments without the size limitation and without taking some of the special constraints into account were performed. In all cases, the experiments yielded the same number of conflicts as the original models, proving optimality with respect to the size assumption $n = 4$.

3 Conclusions

In this paper, we presented an approach to a real-life timetabling case. The problem incorporates certain peculiarities, uncommon to most timetabling problems from the literature. We proposed a model that minimizes the number of conflicts and is able to cope with elective courses. Decomposition was used to split the problem into a model for compulsory courses and a model for elective courses. Both models were solved by integer programming and for the great majority of cases optimal solutions were obtained.

As there is a real-life application associated to the research project, the obtained timetables are expected to be implemented soon. Although the description of the problem seems to be specific to our university, several other universities in Germany face the same type of challenges. Therefore, we consider our work a significant contribution towards the practical applicability of timetabling.

References

[1] V. Cacchiani, A. Caprara, R. Roberti, P. Toth (2013). *A new lower bound for curriculum-based course timetabling.* Computers & OR, 40, 2466–2477.

[2] L. Kiaer, J. Yellen (1992). *Weighted graphs and university course timetabling.* Computers & OR, 19(1), 59–67.

[3] B. McCollum (2007). *A Perspective an Bridging the GAP Between Theory and Practice in University Timetabling.* In: E.K. Burke, H. Rudová (eds.), PATAT 2006, Lecture Notes in Computer Science, 3867, 3–23. Springer, Berlin Heidelberg.

[4] T. Müller, H. Rudová (2014). *Real-life curriculum-based timetabling with elective courses and course sections.* Annals of Operations Research, 1–18.

Locks, Graphs, and Intervals*

Ward Passchyn (Speaker)† Dirk Briskorn‡ Frits C.R. Spieksma§

Introduction

We consider the following scheduling problem that deals with ships passing through a lock. Consider a single lock that consists of m parallel chambers. Let $M = \{1, \ldots, m\}$ be the set of lock chambers. The chambers operate independently of each other and are each characterized by two numbers: their *lockage time* and their *capacity*, respectively denoted by T_j and C_j ($j \in M$). The term lockage time refers to the time needed to bring a ship from the downstream water level to the upstream water level, or vice versa. The capacity gives an upper bound on the number of ships that may simultaneously be present within the chamber. Ships arrive at the locks at given times $t_1, t_2, \ldots, t_n$. Let $S = \{1, \ldots, n\}$ be the set of ships. A ship can arrive either from the upstream side, or from the downstream side (this is called the *position* of a ship). Our interest in this paper is exclusively on the existence of so-called *no-wait* schedules. A no-wait schedule is a schedule where each ship, upon its arrival, can enter a chamber of the lock immediately. Thus, in a no-wait schedule, each ship $s \in S$ leaves the lock at time $t_s + T_j$, where $j \in M$ is the particular chamber that the ship is assigned to. The question we address is thus: given the arrival times and the position of the ships, does there exist an assignment of each ship to a chamber such that no ship has to wait? More compactly: does there exist a no-wait schedule?

Of course we are aware that, from a practical point of view, this problem description does not include all relevant details. However, in order to be able to solve these practical problems, it is good to understand the behaviour of this more basic problem. Moreover, as we will argue below, the problem can be seen as a particular interval scheduling problem, which is of independent interest, and some of our results have implications for other interval scheduling problems. A special case of the problem that will be of some interest is the case where all ships have the same position, i.e., the special case where all ships arrive from the upstream (or downstream) side. We refer to this special case as the uni-directional case.

*This research has been supported by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office.

†ward.passchyn@kuleuven.be. KU Leuven, Research Group Operations Research and Business Statstics (ORSTAT), Naamsestraat 69, B-3000 Leuven, Belgium.

‡briskorn@wiwi.uni-wuppertal.de. University of Wuppertal, Germany.

§frits.spieksma@kuleuven.be. KU Leuven, Research Group Operations Research and Business Statstics (ORSTAT), Naamsestraat 69, B-3000 Leuven, Belgium.

Practical Motivation

Lock Scheduling

Scheduling locks is a problem that is receiving an increasing amount of attention. We mention the following papers: Hermans (2014) presents an $O(n^4 \log n)$ dynamic programming algorithm that asserts feasibility for a single chamber with $C_1 = 1$ with respect to ship deadlines. Coene et al. (2013) deal with minimizing total waiting time for a single lock chamber. They give an $O(n^4)$ algorithm for the bi-directional single chamber setting, and discuss results for practical features such as ship handling times, draught, etc. See also Verstichel (2013), where a lock scheduling problem is considered that includes the packing of ships inside lock chambers. Another related problem is the scheduling of so-called *sidings*, i.e., designated wide sections of a waterway, required for the overtaking of large ships. We refer to Lübbecke et al. (2014), where this problem is considered in order to optimize traffic along the Kiel Canal.

Interval Scheduling

The problem described above can be phrased in terms of interval scheduling as follows. Let a chamber be a machine, and let a ship be a job. Multiple intervals are associated to each job, one for each chamber (or machine) in the instance. Clearly, the starting time of each of the intervals that correspond to one particular ship s is identical, i.e., equal to t_s. The ending times of these intervals are given by $t_s + T_j$, $j \in M$. Notice that when considering a particular interval, it is associated with a ship, and with a machine. A feasible solution consists of a selection of intervals such that (i) one interval corresponding to each ship is selected, (ii) the selected intervals that correspond to a machine are disjoint, and even more: when two consecutive intervals of a machine correspond to ships with the same position, there must be a difference of T_j between the starting point of the later interval and the ending point of the earlier interval. This latter requirement involving the difference between two intervals is needed because a chamber transporting ships of the same position consecutively needs T_j time-units to return to this position. Notice that in the uni-directional case, this last requirement vanishes since all ships then have the same position.

Interval scheduling is a well-studied subject, see Kolen et al. (2007) for an overview. A recent paper by Krumke et al. (2011) deals with interval scheduling on related machines, which can be formulated as follows. Given are m machines, each with a certain speed, and n intervals specified by a starting point and a processing time. They show that even deciding the existence of a schedule is NP-complete. Clearly, this setting is relevant for the uni-directional variant of our problem: by setting the speed of chamber j equal to $\frac{T_j}{T_{\max}}$ (where $T_{\max} = \max_j T_j$), the two problems are very much related. In fact, our problem is a special case of the problem in Krumke et al. (2011) since, in our case, the lengths of intervals corresponding to a particular machine are identical.

Another interesting paper is the one by Böhmová et al. (2013); they consider a version with machine-dependent intervals. Here, a job corresponds to a set of intervals, one for each machine, and to schedule a job exactly one of its intervals must be selected. A set of selected intervals is then called feasible if the intervals corresponding to the same machine do not overlap. In their paper, they consider different special cases, one of which is of primary importance to our problem: the problem with so-called *cores*. Cores refer

to the property that the set of intervals corresponding to the same job have a point in time in common. More specifically, Böhmová et al. (2013) deal with the problem where all intervals of a job end at the same time, and prove that deciding whether a feasible selection of intervals scheduling all jobs exists is NP-complete (solving an open problem from Sung and Vlach (2005)). As this problem is equivalent to dealing with the problem where all intervals of a job start at the same time, this seems to be identical to our problem. There is one difference however: in our case the set of lengths of the intervals that correspond to a job is the same for all jobs - which is not necessarily the case in Böhmová et al. (2013).

Results

In this paper we consider two special cases of the problem described above:

1. The setting with two distinct chambers, and all ships having the same position (the uni-directional case). We give necessary and sufficient conditions determining the existence of a no-wait schedule, and we show how to find such a schedule in linear time.

2. We prove that the uni-directional case of the problem is NP-complete in case the number of chambers is part of the input. This result strengthens both the result given in Krumke et al. (2011) and a result in Böhmová et al. (2013).

References

Böhmová, K., Disser, Y., Mihalák, M., and Widmayer, P. (2013). Interval selection with machine-dependent intervals. In *WADS'13*, pages 170–181.

Coene, S., Passchyn, W., Spieksma, F., Vanden Berghe, G., Briskorn, D., and Hurink, J. (2015). The lockmaster's problem. *under review.*

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco.

Hermans, J. (2014). Optimization of inland shipping - a polynomial time algorithm for the single ship single lock optimization problem. *Journal of Scheduling*, 17:305–319.

Kolen, A. W., Lenstra, J. K., Papadimitriou, C. H., and Spieksma, F. C. (2007). Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543.

Krumke, S. O., Thielen, C., and Westphal, S. (2011). Interval scheduling on related machines. *Computers & Operations Research*, 38(12):1836–1844.

Lübbecke, E., Lübbecke, M. E., and Möhring, R. H. (2014). Ship traffic optimization for the kiel canal. Technical report, TU Berlin, Inst. Math. and RWTH Aachen, Operations Research.

Sung, S. C. and Vlach, M. (2005). Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *Journal of Scheduling*, 8(5):453–460.

Verstichel, J. (2013). *The Lock Scheduling Problem.* PhD thesis, KU Leuven.

Parallel machine scheduling with conflicting jobs: An exact algorithm

Daniel Kowalczyk (Speaker) * Roel Leus †

1 Introduction

A set $J = \{1, \ldots, n\}$ of n independent jobs is to be scheduled on m identical parallel machines without preemption such that the maximum completion time of the jobs, or makespan, is minimized. Each job j has a processing time $p_j \in \mathbb{N}_0$ and is assigned to a single machine. The machines are gathered in set $M = \{1, \ldots, m\}$ and each machine can process at most one job at a time. An undirected *conflict graph* $G = (J, E)$ entails the following constraints: if $\{j, j'\} \in E$ then jobs j and j' are *conflicting jobs*, and they cannot be assigned to the same machine. We call the resulting problem the parallel machine scheduling problem with conflicts (PMC). This problem is NP-hard, because PMC contains both $P||C_{\max}$ as well as the vertex coloring problem (VCP) as special cases. It can be seen that a feasible schedule exists if and only if the conflict graph can be colored with at most m colors; we will assume $m < n$ to avoid trivial solutions.

PMC is theoretically important because it generalizes two well-known problems in combinatorial optimization, but it also naturally arises as (sub-)problem in a number of practical applications in multiprocessor scheduling, TV advertisement scheduling and audit scheduling. PMC was already studied by Bodlaender et al. [1]. They obtain a number of hardness and approximation results for specific graph types, but they do not develop an exact algorithm. Bodlaender et al. present approximation algorithms for the case where a k-coloring of the conflict graph is known a priori, with $k + 1 \leq m$; the worst-case ratio depends only on k and when $\frac{m}{k}$ tends to infinity then the worst-case ratio tends to 2. They also prove that, unless P = NP, no approximation algorithm can improve upon the worst-case ratio of 2.

Informally, problem $P||C_{\max}$ can be seen as a "dual" to the bin packing problem (BPP), where the bin capacities represent the makespan and the number of bins is the number of parallel machines [3]. A similar relation exists between PMC and the bin packing problem with conflicts (BPPC), where items are packed in a minimum number of bins of limited capacity while avoiding joint assignments of conflicting items (based also on a conflict graph). Clearly, BPPC generalizes both BPP and VCP. Algorithms for BPPC have recently been published by a number of researchers, see [4, 6, 7].

*daniel.kowalczyk@kuleuven.be. ORSTAT, Faculty of Economics and Business, KU Leuven, Leuven, Belgium.

†roel.leus@kuleuven.be. ORSTAT, Faculty of Economics and Business, KU Leuven, Leuven, Belgium.

2 An outline of the algorithm

Consider the decision variant of PMC: we introduce an upper bound C on the value of the objective function, and we denote the resulting decision problem by $P(C, m)$, which is to determine whether there exists a feasible schedule without conflicts and with maximum makespan C on m machines. We use the relationship between PMC and BPPC to develop an exact algorithm for PMC. Problems PMC and BPPC are denoted by $P(\cdot, m)$ and $P(C, \cdot)$, respectively.

Our exact algorithm for PMC $\equiv P(\cdot, m)$ works as follows. First a lower bound $L(\cdot, m)$ and an upper bound (heuristic solution) $U(\cdot, m)$ on the minimum makespan are computed. We use multiple lower bounds for $Pm||C_{\max}$, but we tailor them to PMC. The heuristics are taken from Bodlaender et al. [1] for given colorings; we use coloring heuristics from [2, 5]. When the heuristics do not succeed in finding a feasible solution, we invoke a feasibility test in an attempt to recognize instances with empty solution space. The test consists in lower-bounding the chromatic number of the conflict graph: if this number exceeds m then the instance is infeasible. Conversely, when a feasible solution is found with one of the heuristics then we attempt to improve it by means of a limited local search procedure.

If $L(\cdot, m) = U(\cdot, m)$ then an optimal solution has been found, otherwise we start a binary search to identify the optimal objective function (similar to [3]). In this search procedure, we iteratively verify whether a feasible schedule exists with makespan at most $C^* = \lfloor \frac{L(\cdot,m)+U(\cdot,m)}{2} \rfloor$; this verification is established by a branch-and-price (B&P) algorithm. Let $L_F(C, \cdot)$ denote the optimal objective value of the LP relaxation of a set-covering formulation for $P(C, \cdot)$ (a lower bound). If $L_F(C^*, \cdot) > m$ then we replace the lower bound $L(\cdot, m)$ by $C^* + 1$. Otherwise, if the solution is integral then $U(\cdot, m)$ is replaced by the makespan of this solution (which is at most C^*), and if none of the previous two conditions holds then the B&P algorithm will branch and apply the same tests at lower levels of the search tree.

References

[1] Bodlaender, H.L., K. Jansen, G.J. Woeginger. 1994. Scheduling with incompatible jobs. *Discrete Applied Mathematics* **55** 219–232.

[2] Brélaz, D. 1979. New methods to color the vertices of a graph. *Communications of the ACM* **22** 251–256.

[3] Dell'Amico, M., M. Iori, S. Martello, M. Monaci. 2008. Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing* **20** 333–344.

[4] Elhedhli, S., L. Li, M. Gzara, J. Naoum-Sawaya. 2011. A branch-and-price algorithm for the bin packing problem with conflicts. *INFORMS Journal on Computing* **23** 404–415.

[5] Johnson, D.S., C.R. Aragon, L.A. McGeoch, C. Schevon. 1991. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations Research* **39** 378–406.

[6] Muritiba, A.E.F., M. Iori, E. Malaguti, P. Toth. 2010. Algorithms for the bin packing problem with conflicts. *INFORMS Journal on Computing* **22** 401–415.

[7] Sadykov, R., F. Vanderbeck. 2013. Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing* **25** 244–255.

A Tight Lower Bound for Randomized Preemptive Scheduling with Deadlines

Yossi Azar * Oren Gilon (Speaker) †

1 Introduction

We consider the deadline scheduling problem in its simplest setting. Job requests arrive over time at a server that has K processors (for simplicity, you may think as $K = 1$). Upon arrival a job provides its *duration*, its *deadline* and its *value*. In order to *complete* a job, it must be processed for a total time equal to its duration, but this must be done before its deadline. Otherwise the job is lost. The server may choose to *preempt* jobs currently being processed in favor of processing different jobs. The processing of the preempted job can later be resumed, and a job can possibly migrate between processors if $K > 1$. At most K jobs can be processed at a given time - one by each processor. The value gained by the server from a given input sequence is the sum of values of all completed jobs. Note that no value is gained from partially processed jobs that were not completed before their deadline. We define V as the ratio of the maximum to minimum value, D as the ratio of the maximum to minimum duration, and ρ as the ratio of maximum to minimum density (value by duration) over all arriving jobs. We define $\kappa = min(V, D, \rho)$. The model we discuss is identical to the one described by Canetti and Irani [2]. In their paper, they proved a $\Omega(\sqrt{\frac{log(\kappa)}{loglog(\kappa)}})$ lower bound for any randomized preemptive algorithm. The proof of this well-known result is fairly elaborate and involved. In contrast, we show a significantly (more than quadratic) improved lower bound, accompanied by a simple proof (closing a gap which was supposedly open for 20 years). We state our main result:

Theorem 1 *Any randomized preemptive algorithm for the job scheduling problem with deadlines is $\Omega(log(\kappa))$-competitive, where $\kappa = min(V, D, \rho)$.*

Our proof, given fully below, is surprisingly straightforward. This result is complementary to the upper bound provided by the Classify and Randomly Select algorithm, randomly choosing between possible job values or durations or densities, also presented in [2]. This algorithm is $O(log(\kappa))$-competitive and constitutes an upper bound that matches our lower bound. Note that in our lower bound all jobs are *tight*, meaning that their deadline is precisely the sum of their arrival time and their duration.

The scheduling problem described here has been vastly researched in many different variations. There are many small simplifying assumptions that can be made to the model such that a constant competitive deterministic algorithm exists. One such modification

*Blavatnik School of Computer Science, Tel-Aviv University, Israel. Email: azar@tau.ac.il
†Blavatnik School of Computer Science, Tel-Aviv University, Israel. Email: orengilon@gmail.com

is that where all job's densities are constant. In this case there exists a 4-competitive algorithm, as shown by Koren et al [7]. Alternatively, if all jobs are of unit duration (e.g. packets) a 1.828-competitive algorithm was presented by Englert et al [4]. Another simplifying assumption can be a bound on the tightness of jobs, by defining that jobs have a possible window time that is at least α times longer than their duration. In this model there exists a $(\frac{\alpha}{\alpha-1})$-competitive algorithm [3, 5]. A different approach for relaxing this problem is through resource augmentation. In this relaxation, we give the algorithm some added power in hopes of offsetting the advantage of the optimum algorithm . For instance, by giving the server processors which are faster by a factor of $(1+\epsilon)$, a $(1+\frac{1}{\epsilon})$-competitive algorithm was presented by Kalyanasundaram et al [6]. All these modifications show that there are many variations to the model, all of which result in the logarithmic bound collapsing to a constant competitive algorithm.

2 Proof of the Lower Bound

We will now present the proof of our main result (Theorem 1). For simplicity, we assume the number of processors $K = 1$. The details of the multi-processor proof are omitted. Let ALG be some randomized algorithm solving the job scheduling problem. If we let ALG transmit packets fractionally, clearly a 1-competitive algorithm exists for tight jobs. Nevertheless, in our analysis, we give ALG *some* additional power (semi-fractional): we assume ALG can fractionally transmit jobs that have been continuously processed since their arrival, i.e. for a job with value v of duration d that arrived at time T, after being processed for some $t < d$ time by time $T + t$, ALG gains $\frac{t}{d}v$ value from the job. We define $\frac{log(\kappa)}{2}$ job types, where the i'th job type is of duration 4^i and value 2^i for each $0 \leq i < \frac{log(\kappa)}{2}$. We build an input sequence that is composed of a series of *phases*. At the beginning of each phase, one job of each type is sent to the server. We define p_i to be the probability that ALG accepts the i'th job. Note that since all $\frac{log(\kappa)}{2}$ arrived at the same time, at most one of them can be accepted by ALG. This means that $\sum_i p_i \leq 1$. We define $r_i = \sum_{j \leq i} \frac{p_j}{2^{i-j}} + \sum_{j>i} \frac{p_j}{2^{j-i}}$. We note that

$$\begin{aligned}\sum_i r_i &= \sum_i \sum_{j \leq i} \frac{p_j}{2^{i-j}} + \sum_i \sum_{j>i} \frac{p_j}{2^{j-i}} \\ &= \sum_i (p_i \sum_{j \geq i} 2^{i-j}) + \sum_i (p_i \sum_{j<i} 2^{j-i}) \\ &= \sum_i 2p_i + \sum_i p_i \leq 3\end{aligned}$$

This means that there exists some i such that $r_i \leq \frac{3}{\frac{log(\kappa)}{2}} = \frac{6}{log(\kappa)}$. The optimum algorithm (denoted by OPT) processes the job of type i during this phase. This means that during the phase, OPT has a gain of 2^i. ALG's expected gain during this phase is $\sum_{j \leq i} 2^j p_j + \sum_{j>i} 2^i \frac{2^{j+1}}{4^{j+1}} p_j$. This is due to the fractional nature of ALG. If we denote the k'th phase as σ_k, notice that this means that

$$\begin{aligned}\frac{ALG(\sigma_k)}{OPT(\sigma_k)} &= \sum_{j \leq i} 2^{j-i} p_j + \sum_{j>i} \frac{2^{j+1}}{4^{j+1}} p_j \leq \sum_{j \leq i} 2^{j-i} p_j + \sum_{j>i} 2^{-j-1} p_j \\ &\leq \sum_{j \leq i} 2^{j-i} p_j + \sum_{j>i} 2^{i-j} p_j \leq r_i \leq \frac{6}{log(\kappa)}\end{aligned}$$

We begin a new phase immediately when OPT finishes processing its job. We repeat this process N times, for some large N. Note that as ALG is semi-fractional, it can only improve its situation by replacing a partially processed job of type i with the job of type i that arrives at the beginning of the new phase. This means that we can assume that ALG preempts the currently processed job immediately before the start of a new phase. Thus the analysis holds for all phases but the last one, where ALG has a gain of at most V. We denote the full input sequence by σ. Using the claims above, we see that

$$\frac{OPT(\sigma)}{ALG(\sigma)} = \frac{\sum_i OPT(\sigma_i)}{V + \sum_i ALG(\sigma_i)} \geq \frac{\frac{log(\kappa)}{6}(\sum_i ALG(\sigma_i))}{V + \sum_i ALG(\sigma_i)}$$

Thus as the number of phases N tends to ∞, the ratio tends to $\frac{log(\kappa)}{6}$. This means that the input sequence σ gives the required lower bound.

References

[1] Sanjoy Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis Rosier and Dennis Shasha (1991). *On the competitiveness of on-line task real-time task scheduling.* Proceedings of the 12th Real-Time Systems Symposium, p.106-115 San Antonio, Texas.

[2] Ran Canetti and Sandy Irani (1995). *Bounding the Power of Preemption in Randomized Scheduling.* Proceedings of the twenty-seventh annual ACM symposium on Theory of computing, p.606-615, Las Vegas, Nevada.

[3] Bhaskar DasGupta and Michael A. Palis (2000). *Online real-time preemptive scheduling of jobs with deadlines.* Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization, p.96-107.

[4] Matthias Englert and Matthias Westermann (2007). *Considering suppressed packets improves buffer management in QoS switches.* Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), p.209-218.

[5] Juan A. Garay, Joseph Naor, Bulent Yener, and Peng Zhao (2002). *On-line admission control and packet scheduling with interleaving.* Proceedings of Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies.

[6] Bala Kalyanasundaram and Kirk Pruhs (2000). *Speed Is as Powerful as Clairvoyance.* Journal of the ACM (JACM), v.47 n.4, p.617-643.

[7] Gilad Koren and Dennis Shasha (1992). *D-over: An optimal on-line scheduling algorithm for overloaded real-time systems.* Proceedings of the 13th Real-Time Systems Symposium, p.290-299.

SelfishMigrate: A Scalable Algorithm for Non-clairvoyantly Scheduling Heterogeneous Processors

Sungjin Im [*] Janardhan Kulkarni [†] Kamesh Munagala [‡]
Kirk Pruhs (Speaker) [§]

1 Introduction

Many computer architects believe that architectures consisting of heterogeneous processors will be the dominant architectural design in the future: Simulation studies indicate that, for a given area and power budget, heterogeneous multiprocessors can offer an order of magnitude better performance for typical workloads. Looking at the consequences of Moore's Law even further in the future, some computer architectures are projecting that we will transition from the current era of multiprocessor scaling to an era of "dark silicon", in which switches become so dense that it is not economically feasible to cool the chip if all switches are simultaneously powered. One possible architecture in the dark silicon era would be many specialized processors, each designed for a particular type of job. The processors that are on any point of time should be those that are best suited for the current tasks.

It is recognized by the computer systems community [BSC08] and the algorithms community that scheduling these future heterogeneous multiprocessor architectures is a major challenge. Previously it is known that some of the standard scheduling algorithms for single processors and homogeneous processors can perform quite badly on heterogeneous processors [GIK$^+$12]. A scalable algorithm) was known if somehow the scheduler was clairvoyant (able to know the size of a job when it arrives) [CGKM09]; however, this knowledge is generally not available in general purpose computing settings. A scalable algorithm was also known if all jobs were of equal importance [IKM14]; however, the whole raison d'être for heterogeneous architectures is that there is generally heterogeneity among the jobs, most notably in their *importance/priorities*.

Here we show how to nonclairvoyantly schedule jobs of varying importance on heterogeneous processors. Our two main theorems are:

Theorem 1 *For any $\epsilon > 0$, there is a $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive non-clairvoyant algorithm for the problem of minimizing the total weighted flow-time on unrelated machines. Furthermore, each job migrates at most $O((\log W + \log n)/\epsilon)$ times, where W denotes the ratio of the maximum job weight to the minimum.*

[*] sim3@ucmerced.edu . EECS Department, University of California-Merced.
[†] kulkarni@cs.duke.edu. Department of Computer Science, Duke University.
[‡] kamesh@cs.duke.edu. Department of Computer Science, Duke University.
[§] kirk@cs.pitt.edu. Department of Computer Science, University of Pittsburgh.

Theorem 2 *For any $\epsilon > 0$, there is a $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive non-clairvoyant algorithm for the problem of minimizing the total weighted flow-time plus total energy consumption on unrelated machines. This result holds even when each machine i has an arbitrary strictly-convex power function $f_i : [0, \infty) \to [0, \infty)$ with $f_i(0) = 0$. Further this results holds in the model where at each time instant a processor i can either run at speed s_i consuming a power P_i, or be shutdown and consume no energy.*

We demonstrate a simple framework SelfishMigrate that can be best viewed in a game theoretic setting where jobs are selfish agents, and machines declare their scheduling policies in advance. To elaborate we describe the machine behavior and, the job behavior, and the algorithmic analysis.

Machine Behavior: Each machine maintains a *virtual queue* on the current set of jobs assigned to it; newly arriving jobs are appended to the tail of this queue. Each machine treats a migration of a job to it as an arrival, and a migration out of it as a departure. This means a job migrating to a machine is placed at the tail of the virtual queue.

Each machine runs a scheduling policy that is a modification of Weighted Round Robin (WRR) that smoothly assigns larger speed to jobs in the tail of the queue, taking weights into account. We note that the entire analysis also goes through with WRR, albeit with $(2 + \epsilon)$-speed augmentation. The nice aspect of our smooth policies is that we can approximate the instantaneous delay introduced by this job to jobs ahead of it in its virtual queue, *even without knowing job sizes.*

Job Behavior: Each job j has a virtual utility function, which roughly corresponds to the inverse of the instantaneous weighted delay introduced by j to jobs ahead of it in its virtual queue, and their contribution to j's weighted delay. Using these virtual utilities, jobs perform sequential best response (SBR) dynamics, migrating to machines (and get placed in the tail of their virtual queue) if doing so leads to larger virtual utility. Therefore, at each time instant, the job migration achieves a Nash equilibrium of the SBR dynamics on the virtual utilities. We show that our definition of the virtual utilities implies they never decrease due to migrations, arrivals, or departures, so that at any time instant the Nash equilibrium exists and is computable. (Of course one can also simulate SBR dynamics and migrate each job directly to the machine that is predicted by the Nash equilibrium.)

Algorithmic Analysis: When a job migrates to a machine, the virtual utility starts off being the same as the real speed the job receives. As time goes by, the virtual queue ahead of this job shrinks, and the virtual queue behind the job grows. This lowers the real speed the job receives, but its virtual utility, which measures the inverse of the impact to jobs ahead in the queue and vice versa, does not decrease. Our key contribution is to define the coordination game on the virtual utilities, rather than on the actual speed improvement jobs receive on migration. The analysis then proceeds by setting the dual variable for a job to the increase in overall weighted delay it causes on jobs ahead of it in its virtual queue. A key insight is to show that Nash dynamics on virtual utilities directly corresponds to our setting of dual variables being feasible for the dual constraints, implying the desired competitive ratio. This overall approach requires two key properties from the virtual utility:

- The virtual utility should correspond roughly to the inverse of the instantaneous delay induced by a job on jobs ahead of it in its virtual queue.
- SBR dynamics should monotonically improve virtual utility, leading to a Nash equilibrium that corresponds exactly to satisfying the dual constraints.

A key contribution is to show the existence of such a virtual utility function for WRR and its scalable modifications, when coupled with the right notion of virtual queues. In hindsight, we believe this framework is the right way to generalize the greedy dispatch rules and dual fitting analysis from previous works [AGK12, IKM14], and we hope it finds more applications in complex scheduling settings.

A conference version of these results can be found at [IKMP14], and a full version can be found at http://arxiv.org/abs/1404.1943.

References

[AGK12] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241, 2012.

[BSC08] Fred A. Bower, Daniel J. Sorin, and Landon P. Cox. The impact of dynamically heterogeneous multicore processors on thread scheduling. *IEEE Micro*, 28(3):17–25, May 2008.

[CGKM09] J. S. Chadha, N. Garg, A. Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In *STOC*, 2009.

[GIK+12] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *SODA*, pages 1242–1253, 2012.

[IKM14] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. In *STOC*, 2014.

[IKMP14] Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *FOCS*, pages 531–540, 2014.

A Fully Polynomial-Time Approximation Scheme for Speed Scaling with Sleep State

Antonios Antoniadis [*] Chien-Chung Huang [†] Sebastian Ott [‡]

1 Introduction

As energy-efficiency in computing environments becomes more and more crucial, chip manufacturers are increasingly incorporating energy-saving functionalities to their processors. One of the most common such functionalities is *dynamic speed scaling*, where the processor is capable to dynamically adjust the speed at which it operates. The algorithmic study of dynamic speed scaling was introduced in a seminal paper by Yao, Demers and Shenker [11]. A higher speed implies a higher performance, but this performance comes at the cost of a higher energy consumption. On the other hand, a lower speed results in better energy-efficiency, but at the cost of performance degradation. In practice, it has been observed [7, 4] that the power consumption of the processor is approximately proportional to its speed cubed. However, even when the processor is idling, it consumes a non-negligible amount of energy just for the sake of "being active" (for example because of leakage current). Due to this fact, additional energy-savings can be obtained by further incorporating a *sleep state* to the processor, in addition to the speed-scaling capability. A sleep state is a state of negligible or even zero energy-consumption, to which the processor can transition when it is idle. Some fixed energy-consumption is then required to transition the processor back to the active state in order to continue processing.

We study the offline problem of minimizing energy-consumptions in computational settings that are equipped with both speed scaling and sleep state capabilities. This problem is called *speed scaling with sleep state*.

A more thorough discussion of dynamic speed scaling problems can be found in the surveys [1, 8].

Problem Description: Consider a processor that is equipped with two states: the *active state* during which it can execute jobs while incurring some energy consumption, and the *sleep state* during which no jobs can be executed, but also no energy is consumed. We assume that a *wake-up operation*, that is a transition from the sleep state to the active state, incurs a constant energy cost $C > 0$, whereas transitioning from the active state to the sleep state is free of charge. Further, as in [2, 9], the power required by the processor in the active state is an arbitrary convex and non-decreasing function P of its speed

[*] aantonia@mpi-inf.mpg.de. Max-Planck-Institut für Informatik, Saarbrücken, Germany.
[†] villars@gmail.com. Chalmers University, Göteborg, Sweden.
[‡] ott@mpi-inf.mpg.de. Max-Planck-Institut für Informatik, Saarbrücken, Germany.

s. We assume that $P(0) > 0$, since (i) as already mentioned, real-world processors are known to have leakage current and (ii) otherwise the sleep state would be redundant. Further motivation for considering arbitrary convex power functions for speed scaling can be found, for example, in [5].

The input is a set $\mathcal{J}$ of n jobs. Each job j is associated with a release time r_j, a deadline d_j and a processing volume v_j. One can think of the processing volume as the number of CPU cycles that are required in order to completely process the job, so that if job j is processed at a speed of s, then v_j/s time-units are required to complete the job.

A *schedule* is defined as a mapping of every time point t to the state of the processor, its speed, and the job being processed at t (or *null* if there is no job running at t). Note that the processing speed is zero whenever the processor sleeps, and that a job can only be processed when the speed is strictly positive. A schedule is called *feasible* when the whole processing volume of every job j is completely processed in interval $[r_j, d_j)$. Preemption of jobs is allowed.

The energy consumption incurred by schedule $\mathcal{S}$ while the processor is in the active state, is its power integrated over time, i.e. $\int P(s(t))dt$, where $s(t)$ is the processing speed at time t, and the integral is taken over all time points during which the processor is active under $\mathcal{S}$. Assume that $\mathcal{S}$ performs k transitions from the sleep state to the active state. Then the total energy consumption of $\mathcal{S}$ is $E(\mathcal{S}) := \int P(s(t))dt + kC$, where again the integral is taken over all time points at which $\mathcal{S}$ keeps the processor in the active state. We are seeking a feasible schedule that minimizes the total energy consumption.

2 Previous Work and Our Contribution

The algorithmic study of speed scaling with sleep state was initiated in [9], who also provided a 2-approximation algogrithm for the problem. The exact computational complexity of speed scaling with sleep state has been repeatedly posed as an open question (see e.g. [8, 2, 6]). The currently best known upper and lower bounds are a 4/3-approximation algorithm and NP-hardness due to [2] and [2, 10], respectively.

We close the aforementioned gap between the upper and lower bound on the computational complexity of speed scaling with sleep state by presenting a fully polynomial-time approximation scheme for the problem. At the core of our approach is a transformation of the original preemptive problem into a non-preemptive scheduling problem of the same type. At first sight, this may seem counterintuitive, especially as Bampis et al. [3] showed that (for the problem of speed scaling alone) the ratio between an optimal preemptive and an optimal non-preemptive solution on the same instance can be very high. However, this does not apply in our case, as we consider the non-preemptive problem on a modified instance, where each job is replaced by a polynomial number of *pieces*. Furthermore, in our analysis, we make use of a particular lexicographic ordering that does exploit the advantages of preemption.

In order to compute an optimal schedule for the modified instance via dynamic programming, we require a number of properties that pieces must satisfy in a valid schedule. The definition of these properties is based on a discretization of the time horizon by a polynomial number of time points. Roughly speaking, we focus on those schedules that start and end the processing of each piece at such time points, and satisfy

a certain constraint on the processing order of the pieces. Proving that a near-optimal schedule in this class exists is the most subtle part of our approach. On the one hand, our DP structurally relies on the processing order constraint, but on the other hand, such a property is difficult to establish in an optimal schedule after having introduced indivisible volumes (since pieces of different jobs might have different volumes and cannot easily be interchanged). To get around this, we first ensure the right ordering in an optimal schedule for the preemptive setting, and then perform a series of transformations to a non-preemptive schedule with the above properties. Each of these transformations increases the energy consumption only by a small factor, and maintains the correct ordering among the pieces.

References

[1] Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.

[2] Susanne Albers and Antonios Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. *ACM Transactions on Algorithms*, 10(2):9, 2014.

[3] Evripidis Bampis, Alexander Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Ioannis Nemparis. From preemptive to non-preemptive speed-scaling scheduling. In *COCOON*, pages 134–146. Springer, 2013.

[4] Nikhil Bansal, Ho-Leung Chan, Dmitriy Katz, and Kirk Pruhs. Improved bounds for speed scaling in devices obeying the cube-root rule. *Theory of Computing*, 8(1):209–229, 2012.

[5] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms*, 9(2):18, 2013.

[6] Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial-time algorithms for minimum energy scheduling. *ACM Transactions on Algorithms*, 8(3):26, 2012.

[7] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, November 2000.

[8] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[9] Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3(4), 2007.

[10] Gunjan Kumar and Saswata Shannigrahi. NP-hardness of speed scaling with a sleep state. *CoRR*, abs/1304.7373, 2013.

[11] F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *FOCS*, pages 374–382. IEEE Computer Society, 1995.

On minimizing the number of tardy jobs on the two-machine open shop with common due date

Federico Della Croce * Christos Koulamas † Vincent T'kindt ‡

1 Introduction

We consider the two-machine open shop problem with a constraint on the availability of the shop and the objective of maximizing the number of processed jobs while the shop is available.

There is a set of n jobs $J_j,\ j = 1, ..., n$, all of them available at time zero; each job J_j must be processed non-preemptively on two machines M_1, M_2, with known integer processing times $p_{1,j}, p_{2,j}$, on machines M_1, M_2, respectively. Each machine can process at most one job at a time and the two operations of a job can be processed in any order. The shop is available for only a predetermined integer amount of time d; the objective is to maximize the number of jobs that can be completed while the shop is available.

The natural application of this problem is in shops with preset daily operating hours. Since the jobs cannot be preempted, the shop manager should ensure that an appropriate subset of the available jobs is selected each day in order to maximize the number of jobs completed before the shop closes. The remaining jobs become available for processing on the next day and so on.

The problem has been considered in the scheduling literature as the two-machine open shop problem with a common due date d and the objective of minimizing the number of tardy jobs. Using the three-field notation introduced by Graham et al. [1], the problem is denoted as $O2|d_j = d|n_T$. Jozefowska et al. [2] showed that the $O2|d_j = d|n_T$ problem is ordinary NP-hard and proposed a $O(nd^2)$ pseudo-polynomial dynamic programming (DP) algorithm.

Wagneur and Sriskandarajah [5] introduced a variant of the open shop in which the operations of each job are allowed to be processed concurrently, to be called the concurrent open shop from now on. This variant leads to the $OC2|d_j = d|n_T$ problem (with concurrent operations) which is also ordinary NP-hard and solvable in $O(nd^2)$ time by dynamic programming as stated in [4] and the references there in.

*federico.dellacroce@polito.it. D.A.I., Politecnico di Torino, Italy.

†koulamas@fiu.edu. College of Business, Florida International University, USA.

‡tkindt@univ-tours.fr. Université Francois-Rabelais, CNRS, LI EA 6300, OC ERL CNRS 6305, Tours, France

It is easy to observe that the DP algorithm for the $O2|d_j = d|n_T$ problem can also solve the $OC2|d_j = d|n_T$ problem with the following modification: in the $O2|d_j = d|n_T$ case, all jobs J_j with $p_{1,j} + p_{2,j} > d$ are scheduled tardy beforehand and not considered in the DP implementation while in the $OC2|d_j = d|n_T$ case only the jobs J_j with $\max\{p_{1,j}, p_{2,j}\} > d$ are scheduled tardy beforehand.

An important issue associated with pseudo-polynomial DP algorithms is the possibility of extending them into fully polynomial time approximation schemes (FPTAS). Lin and Kononov [4] showed that this is not possible in the case of the $OC2|d_j = d|n_T$ problem; similar consideration holds for the $O2|d_j = d|n_T$ problem.

Let n_A, n_O denote the number of tardy jobs supplied by a heuristic algorithm A and an optimal algorithm respectively. The non-availability of a FPTAS combined with the integrality of the number of tardy jobs objective implies that a polynomial time algorithm A supplying a solution with $n_A \leq n_O + 1$, or equivalently with approximation ratio $\rho = \frac{n_A}{n_O} \leq \frac{n_O+1}{n_O}$, is a "best possible" polynomial time algorithm unless $P = NP$.

2 Main results

It is well known from [3] that the $O2|d_j = d|n_T$ problem can be expressed by means of the following linear programming model that uses 0/1 variables x_j for each job J_j ($x_j = 1$ if J_j is early, else $x_j = 0$).

$$\max \sum_{j=1}^{n} x_j \tag{1}$$

$$\sum_{j=1}^{n} p_{i,j} x_j \leq d \qquad i = 1, \ldots, 2 \tag{2}$$

$$x_j \in \{0, 1\} \qquad j = 1, \ldots, n \tag{3}$$

$$x_j = 0 \qquad j \in N' = \{j = 1, \ldots, n | p_{1,j} + p_{2,j} > d\} \tag{4}$$

The objective sums up the number of early jobs. Constraints 2 indicate that the sum of the processing times of the early jobs on machines M_1, M_2 does not exceed the common due date d. Constraints 3 indicate that all variables are binary (all jobs are either early or tardy) and constraints 4 indicate that all jobs J_j having sum of processing times $p_{1,j} + p_{2,j} > d$ are tardy.

Model (1-4) is essentially a two-constraint 0/1 knapsack problem with unit profits. Herewith we denote by $z* = \sum_{i=1}^{n} x_i^*$ the optimal solution of the considered model and by $z' = \sum_{i=1}^{n} x_i'$ the optimal solution of the continuous relaxation of the considered model. Notice that, correspondingly, $n_O = n - z^*$. Consider algorithm 1 for the $O2|d_j = d|n_T$ problem.

The following proposition holds (proof omitted).

Proposition 1. *Algorithm $NumTardy-approx$ applied to the $O2|d_j = d|n_T$ problem is a best possible linear time algorithm and provides approximation ratio $\frac{n_{NumTardy-approx}}{n_O} \leq \frac{n_O+1}{n_O}$. Further, this bound is tight.*

Algorithm 1 $NumTardy - approx$

1) Solve the continuous relaxation of model (1-4) and obtain z' and the vector $\overline{x}'$ of the variables $x_j \ \forall j = 1, ..., n$.
2) Set $x''_j = \lfloor x'_j \rfloor \ \forall j = 1, ..., n$.
Output: $n_{NumTardy-approx} = n - z'' = n - \sum_{j=1}^{n} x''_j$

Proposition 1 provides an information on the quality reachable by an approximation algorithm on problem $O2|d_j = d|n_T$ running in polynomial time. Let consider notation $O^*(\cdot)$ to measure complexity of an algorithm ignoring polynomial factors. When we turn to the exact solution of the considered problem, the following proposition holds (proof omitted).

Proposition 2. *The $O2|d_j = d|n_T$ problem can be solved to optimality with $O^*(1.4142^n)$ worst-case time (and space) complexity.*

Proposition 2 is interesting since it indicates that whenever we are not interested in a best polynomial time heuristic we have to pay for at most $O^*(1.4142^n)$ time to get the optimal solution. In a sense, we pay $O(n)$ time to get a solution with $n_A \leq n_O + 1$ and $O^*(1.4142^n)$, in the worst-case, to get a solution with $n_A = n_O$. Besides, notice that the same $O^*(1.4142^n)$ complexity bound currently present on the 0/1 knapsack problem [5] holds and it does not seem to be negligible the effort necessary to improve upon that bound for the $O2|d_j = d|n_T$ problem.

References

[1] Graham, R.L., E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, 1979, Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey, Annals of Discrete Mathematics, Vol. 5, 287-326.

[2] Jozefowska, J., B. Jurisch, and W. Kubiak, 1994, Scheduling shops to minimize the weighted number of late jobs, Operations Research Letters, Vol. 16, 277-283.

[3] Koulamas, C., and Kyparisis G.J., 1998, Open Shop Scheduling To Minimize Late Jobs, Naval Research Logistics, Vol. 45, 525-532.

[4] Lin, B.M.T., and A.V. Kononov, 2007, Customer order scheduling to minimize the number of late jobs, European Journal of Operational Research, Vol. 183, 944-948.

[5] Wagneur, E. and C. Sriskandarajah, 1993, Openshops with jobs overlap, European Journal of Operational Research, Vol. 71, 366-378.

[6] Woeginger, G. J., Exact algorithms for NP-hard problems: a survey. In M. Juenger, G. Reinelt, and G. Rinaldi, (eds.) Combinatorial Optimization - Eureka! You shrink!, volume 2570 of Lecture Notes in Computer Science, 185207. Springer-Verlag (2003).

Complexity results for robust storage loading problems

Thanh Le Xuan (Speaker) * Sigrid Knust †

1 Introduction

Storage loading problems appear in practical applications such as container terminals, container ships or warehouses, see [3] and references therein. In such problems, arriving items from trains, trucks or vessels have to be assigned to stacks respecting certain constraints. We study problems where n items have to be loaded into a storage area which is arranged in m stacks, each stack has its own fixed position and consists of b levels.

The items are partitioned into three different sets: set I^{fix} consists of items that are already fixed in the storage area, set I^1 consists of items that have to be stored now (i.e. assigned to feasible positions described by tuples (stack, level)), set I^2 contains items that will arrive later. No stack is fully filled by I^{fix}-items. The items must be stored in the sequence $I^{fix} \to I^1 \to I^2$, more precisely, no I^1-item can be stacked below an I^{fix}-item, and no I^2-item can be stacked below an item of $I^{fix} \cup I^1$.

We assume that the items may have different lengths. The actual length of each item in $I^{fix} \cup I^1$ is known exactly. For each item $i \in I^2$ a nominal length $\bar{\ell}_i$ is known but its actual length may vary in an interval $[\ell_i^{min}, \ell_i^{max}]$. For stability reasons, a hard stacking constraint on lengths must be regarded, i.e., item i may only be stacked on top of item j if $\ell_i \leq \ell_j$. This stacking constraint is encoded by a matrix $S = (s_{ij}) \in \{0,1\}^{n \times n}$, where $s_{ij} = 1$ if and only if item i can be stacked onto item j. We denote this stacking constraint by $s_{ij}(\ell)$.

We focus on finding feasible stacking solutions for all items. To guarantee the feasibility of the solutions in different scenarios of the lengths of I^2-items, we apply the concepts of strict and adjustable robustness (see [1]). In strictly robust storage loading problems, we have to calculate positions for all items in $I^1 \cup I^2$ satisfying the stacking constraints in all scenarios of lengths of I^2-items. In adjustable robust storage loading problems, we only have to calculate positions for the items in I^1 such that for each scenario of lengths of I^2-items we can find feasible assignments of I^2-items to positions in the storage area.

To the best of our knowledge, no literature on robust optimization for storage loading problems exists. For the deterministic version of the storage loading problems, Bruns et al. [2] provided complexity results for some special cases. In this paper we present

*xuanthanh.le@uni-osnabrueck.de. Department of Mathematics and Computer Science, University of Osnabrueck, Albrechtstrasse 28, 49076 Osnabrueck, Germany, supported by the Deutsche Forschungsgemeinschaft, GRK 1916/1.

†sigrid@informatik.uni-osnabrueck.de. Department of Mathematics and Computer Science, University of Osnabrueck, Albrechtstrasse 28, 49076 Osnabrueck, Germany.

complexity results for some particular cases of the robust versions of the storage loading problems.

2 Main results

To shortly describe our considered problems, we follow the idea of using the three-field notation $\alpha \mid \beta \mid \gamma$ proposed in [3]. The first field α contains the problem type (L for loading) and information about the common height limit b of stacks (this subfield is omitted if b is given as part of the input). The second field β contains information about stacking sequence and stacking constraint on lengths of items $s_{ij}(\tilde{\ell})$, in which the tilde symbol above ℓ means that the lengths of some items are uncertain. We use the notation $\tilde{I}^2$ to mean that only items in I^2 have uncertain lengths. The last field γ consists of the notation '–' to mean that we are looking for feasible solutions.

Theorem 1 *The problem of finding a strictly robust solution to*

$$(P_1) \qquad L, b = 2 \mid I^{fix} \rightarrow I^1 \rightarrow \tilde{I}^2, s_{ij}(\tilde{\ell}) \mid -$$

can be solved in $\mathcal{O}(n^{2.5})$.

Theorem 1 extends a result in [2] which was obtained for the deterministic version of (P_1). This theorem can be proved by introducing an undirected graph representing the certain stackability of each pair of items, then by calculating a maximum cardinality matching in this graph.

Theorem 2 *An adjustable robust solution to*

$$L, b = 2 \mid I^{fix} \rightarrow I^1 \rightarrow \tilde{I}^2, s_{ij}(\tilde{\ell}) \mid -$$

can be calculated in $\mathcal{O}(n^2)$.

Theorem 3 *An adjustable robust solution to*

$$L \mid I^1 \rightarrow \tilde{I}^2, s_{ij}(\tilde{\ell}) \mid -$$

can be calculated in $\mathcal{O}(n \log n)$ *(if it exists).*

The key ideas for proving Theorem 2 and Theorem 3 are as follows.

- The stacking constraint on lengths defines a total order on the set of all items.
- Thanks to the interval data on lengths of I^2-items, it is sufficient to consider only one dominant scenario in which each I^2-item has maximum length (i.e., $\ell_i = \ell_i^{max}$ for all $i \in I^2$).
- Apply a greedy algorithm in appropriate ways.

References

[1] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski (2009). *Robust Optimization.* Princeton University Press, Princeton and Oxford.

[2] F. Bruns, S. Knust, and N. V. Shakhlevich (2015). Complexity results for storage loading problems with stacking constraints. *submitted.*

[3] J. Lehnfeld and S. Knust (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research* 239:297-312.