Proceedings of the 13th Workshop on Models and Algorithms for Planning and Scheduling Problems

MAPSP 2017
Seeon Abbey, Germany
June 12–16, 2017

Editors: Susanne Albers, Nicole Megow, Andreas S. Schulz and Leen Stougie

Sponsors

† Sadly passed away on January 5, 2017.

Preface

This volume contains abstracts of talks presented at the 13th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2017), held from June 12 to June 16, 2017, in Seeon Abbey in Seeon-Seebruck, Germany.

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 and 2003), Siena, Italy (2005), Istanbul, Turkey (2007), Kerkrade, Netherlands (2009), Nymburk, Czech Republic (2011), Pont á Mousson, France (2013) and La Roche-en-Ardenne, Belgium (2015).

The abstracts in this volume include 5 invited talks by Nikhil Bansal, Bernhard Häupler, Monika Henzinger, Jochen Könemann and Rolf H. Möhring plus 85 contributed talks. Each submission was reviewed by at least two program committee members.

Submitted papers are presented in three parallel tracks. We adopted the policy of MAPSP 2015 and surveyed participants' preferences for attending talks. Frits Spieksma (organizer of MAPSP 2015) and Bart Vangerven kindly constructed a schedule that maximizes total attendance and satisfaction according to the given preferences.

We thank all sponsors of MAPSP 2017 for their generous funding. The conference is supported by the German Research Foundation (DFG), the Association of European Operational Research Societies (EURO), the German Operations Research Society (GOR), the company Optineon, the Technical University of Munich, and the University of Bremen.

Special thanks also to Marek Adamczyk for maintaining the webpage, Franziska Eberle for producing the proceedings, Eva Simon and Jamila Tazit for organization and to the helping hands Daniel Schmidt gen. Waldschmidt and Benedikt Plank.

We thank the referees, and the members of the organizing committee for making MAPSP 2017 possible.

On behalf of the Program Committee of MAPSP 2017,

Leen Stougie (Chair).

Contents

Invited Talks

Contributed Talks

Some Open Problems in Scheduling

Nikhil Bansal *

The survey "Polynomial time approximation algorithms for machine scheduling: Ten open problems" by P. Schuurman and G.J. Woeginger from 1999 has had an important impact on scheduling research, and many prominent open problems mentioned there have now been solved. I will revisit some of these problems, highlight the recent progress and the main ideas involved, and describe some new problems and directions of my own.

*n.bansal@tue.nl. Department of Mathematics and Computer Science, Eindhoven University of Technology.

Low-Congestion Shortcuts: Routing for Distributed Optimization Algorithms

Bernhard Häupler *

How fast a distributed optimization problem can be solved in a given network depends in a highly non-trivial manner on the topology of the network. This talk will introduce a simple routing problem and a related graph structure, called low-congestion shortcuts, which often tightly characterize and capture this dependency.

*haeupler@cs.cmu.edu. School of Computer Science, Carnegie Mellon University.

Local Flow Partitioning for Faster Edge Connectivity or Flow Beats PageRank

Monika Henzinger *

We present a new deterministic algorithm for computing the unweighted minimum cut, aka edge connectivity, of an undirected graph with n nodes and m edges in time $\mathcal{O}(m \log^2 n \log\log^2 n)$. It improves the previous break-through result by Kawarabayashi and Thorup, which uses a PageRank-based subroutine and takes time $\mathcal{O}(m \log^1 2n)$. To achieve the running time improvement we replace the PageRank-based computation by a flow-based computation. Note that our algorithm is also faster than the fastest randomized algorithm for this problem, which is Karger's 1996 $\mathcal{O}(m \log^3 n)$-time algorithm.

This is joint work with Satish Rao and Di Wang and appeared at SODA 2017.

*monika.henzinger@univie.ac.at. Faculty of Computer Science, Universität Wien.

Improved Approximation for Tree Augmentation via Chvatal Gomory Cuts

Jochen Könemann *

In the weighted tree augmentation problem (WTAP) we are given a tree T in a graph $G = (V, E)$. Edges in $E \setminus T$ are called links and carry non-negative weights. The goal is to find a minimum-weight set L of links such that $T + L$ is 2-edge-connected.

WTAP is a classical NP- and APX-hard network design problem (even when all links have weight 1; call this TAP). The best known approximation algorithm achieves a performance guarantee of 2 (due to Fredrickson & Jaja). Kortsarz and Nutov improved this to 3/2 in the special case of TAP. Very recently Adjashvili gave a 1.96-approximation for WTAP whenever link weights are bounded, and a 5/3-approximation for TAP.

We show how to improve the result of Adjashvili and obtain a 3/2-approximation for both weighted and unweighted tree augmentation in the bounded link weight setting. The key ingredient is a strengthened linear programming formulation for WTAP.

This is joint work with Martin Gross, Samuel Fiorini and Laura Sanita; see `https://arxiv.org/abs/1702.05567`.

*`jochen@uwaterloo.ca`. Department of Combinatorics and Optimization, University of Waterloo.

Online Scheduling of Bidirectional Traffic

Rolf Möhring *

We introduce, discuss, and solve a hard practical optimization problem that deals with routing bidirectional traffic on a line. This situation occurs in train traffic on a single track with sidings, in ship traffic in a canal, or in bidirectional data communication.

We illustrate our methods and algorithms on the Kiel Canal, which is the world '92s busiest artificial waterway with more passages than the Panama and Suez Canal together. The scheduling problem arises from scarce resources (sidings) that are the only locations where large ships can pass each other in opposing directions. This requires decisions on who should wait for whom (scheduling), in which siding to wait (packing) and when and how far to steer a ship between sidings (routing), and all this for online arriving ships at both sides of the canal. We have developed a combinatorial algorithm that provides a unified view of routing and scheduling that combines simultaneous (global) and sequential (local) solution approaches to allocate scarce network resources to a stream of online arriving vehicles in a collision-free manner. Computational experiments on real traffic data with results obtained by human expert planners show that our algorithm improves upon manual planning by 25%. This combination of routing and scheduling (without the packing) leads to a new class of scheduling problems, and we will also address recent complexity and approximation results for this class.

The lecture is based on joint work with Elisabeth Lübbecke and Marco Lübbecke.

*Rolf.Moehring@tu-berlin.de. Institut für Mathematik, Technische Universität Berlin.

On Scheduling with a Sharable Resource

Peter Kling (Speaker) [*] Alexander Mäcker [†] Sören Riechers [‡]
Alexander Skopalik [§]

1 Introduction

Consider a system of $m \in \mathbb{N}$ processors and $n \in \mathbb{N}$ jobs from the set $J \coloneqq \{1, 2, \ldots, n\}$. There is a *common resource* that is to be shared by the processors. In each round $t \in \mathbb{N}_0$ each processor i is assigned a share $R_i(t) \in [0, 1]$ of the resource, with the requirement that $\sum_{i \in [m]} R_i(t) \leq 1$ (i.e., the resource is not overused). Each processor can process at most one job per round and each job can be processed by at most one processor. A job j has a *processing volume* $p_j \in \mathbb{N}$ and a *resource requirement* $r_j \in [0, 1]$. Without loss of generality (w.l.o.g.), $r_1 \leq r_2 \leq \cdots \leq r_n$. The resource requirement specifies what portion of the common resource is needed to process one unit of the job's processing volume during a round. More exactly, assume job j is processed by processor i during round t. Then exactly $\min(R_i(t)/r_j, 1)$ units of j's processing volume are processed during that round. A job is finished once all p_j units of its processing volume have been processed. Preemption and migration are not allowed. The objective is to find a schedule S (i.e., a resource and job assignment) that minimizes the *makespan*. We refer to this problem as *Linear Shared Resource Job-Scheduling* (LinSRJS).

LinSRJS is a variant (and arguably a more realistic version) of [1]. There, the authors proposed and studied this type shared resource, but for the case when all jobs are unit-size, have already been assigned to the different processors, and are to be finished in a fixed order on each processor. Their results for this setting include NP-hardness (if m is variable), an optimal algorithm for 2 processors, and a greedy algorithm achieving an approximation ratio of $2 - 1/m$. Our model assumes no a priori assignment of jobs nor any order constraints. Moreover, jobs of arbitrary sizes. This variant turns (unsurprisingly) out to be NP-hard, even for unit size jobs. One of our main results is the design and analysis of a simple polynomial time approximation algorithm with the following guarantee:

Theorem 1 *Our algorithm for Linear Shared Resource Job-Scheduling generates a valid schedule with approximation ratio* $2 + \frac{2}{m-2} + \mathrm{o}(1)$. *If jobs have unit size, the approximation ratio is* $1 + \frac{1}{m-1} + \mathrm{o}(1)$.

[*] peter.kling@uni-hamburg.de. University of Hamburg, Vog-Kölln-Str. 30, 22527 Hamburg, Germany.
[†] alexander.maecker@upb.de. Paderborn University, Fürstenallee 11, 33102 Paderborn, Germany.
[‡] soeren.riechers@upb.de. Paderborn University, Fürstenallee 11, 33102 Paderborn, Germany.
[§] alexander.skopalik@upb.de. Paderborn University, Fürstenallee 11, 33102 Paderborn, Germany.

Additionally, we further generalize this model to a *composed services* setting: Here, a number of services (tasks) $T_1, T_2, \ldots, T_N$ is to be scheduled, where each task T_i is composed of a number of (unit size) jobs (i.e., $T_i = \{ j_{i,1}, j_{i,2}, \ldots, j_{i,n_i} \}$). The objective is to minimize the average completion time of the tasks (where a task is completed if all its jobs are completed). We refer to this problem as *Linear Shared Resource Task-Scheduling* (LinSRTS). Using Theorem 1 as a building block, we are able to derive an algorithm with the following guarantee:

Theorem 2 *Our algorithm for Linear Shared Resource Task-Scheduling generates a valid schedule with approximation ratio* $2 + \frac{4}{m-3} + \mathrm{o}(1)$.

The rest of this abstract gives a brief overlook of the deisgn and analysis of our algorithm for LinSRJS.

2 Overview: Algorithm & Analysis of LinSRJS

Given a schedule S for LinSRJS, consider a job j on processor i in round t. Assume (w.l.o.g.) $R_i(t) \leq r_j$ (otherwise set $R_i(t)$ to r_j to get a schedule with the same makespan). Let t_1 and t_2 be the rounds when j was started and finished, respectively. We have $\sum_{t=t_1}^{t_2} R_i(t)/r_j \geq p_j$ or, equivalently, $\sum_{t=t_1}^{t_2} R_i(t) \geq r_j \cdot p_j$. Thus, with $s_j \coloneqq r_j \cdot p_j$ as the *total resource requirement* of job j, one can think of j of being finished once the total resource shares it received while being processed sum up to s_j. We define $s_j(t) \coloneqq s_j - \sum_{t'=t_1}^{t} R_i(t')$ as the total resource requirement remaining after round t. Note that job j is finished in the first round t for which $s_j(t) = 0$. We use $J(t) \coloneqq \{ j \in J \mid s_j(t) > 0 \}$ to denote the set of jobs that are not finished after round t.

An interesting observation is that, with this formulation, one can think of LinSRJS as a bin packing variant. There are n items that are to be packed in bins of capcity 1. Item j has size s_j and can be split into parts, but each part cannot be larger than r_j. Moreover, each bin cannot contain more than m item parts. The goal is to minimize the number of bins (corresponding to number of rounds in LinSRJS) necessary to pack all items. If we would allow preemption in LinSRJS, this bin packing variant would be equivalent to LinSRJS. Similar bin packing problems (without the constraint of a maximal part size) have been considered in [2, 4, 3].

Our algorithm builds upon the central definition of *maximal good job windows*. To define these, let $U \subseteq J$ and define $r(U) \coloneqq \sum_{j \in U} r_j$. Given $t \in \mathbb{N}$, let $s_t(U) \coloneqq \sum_{j \in U} s_j(t)$. We say a job $j \in J(t)$ is *fractured* at time t if $s_j(t) = k \cdot r_j + q_j(t)$ for some $k \in \mathbb{N}_0$ and $q_j(t) \in (0, r_j)$.

Definition 3 (Job Window) *A subset of unfinished jobs $W \subseteq J(t-1)$ is called a* good window *(for round t) if it has the following properties:*

(a) (consecutive) $j_1, j_2 \in W \Rightarrow J(t-1) \cap \{ j_1, j_1 + 1, \ldots, j_2 \} \subseteq W$,
(b) (window size) $|W| \leq m - 1$,
(c) (at most one fractured) $|\{ j \in W \mid q_j(t-1) > 0 \}| \leq 1$,
(d) $r(W \setminus \{ \max W \}) < 1$, *and*
(e) (contains all started jobs) $j \in J(t-1) \setminus W \Rightarrow s_j(t-1) = s_j$.

We say W is maximal *if, additionally, the following properties hold:*

(f) $|W| < m - 1 \Rightarrow \min W = \min J(t-1)$ *and*

(g) $r(W) < 1 \Rightarrow \max W = \max J(t-1)$.

We design our algorithm such that it has three key properties:

- During any round, it processes jobs from a maximal good window W.
- If during round t the window W is at the "left border" of the remaining jobs (i.e., $\min W = \min J(t-1)$), then this holds for all $t' > t$.
- If during round t the window W is at the "right border" of the remaining jobs (i.e., $\max W = \max J(t-1)$), then this holds for all $t' > t$.

Note that if W is *not* at the left border of the remaining jobs, Properties (d) and (f) imply that we can assign the resource such that at least $m-2$ jobs receive their full resource requirement r_j. Similarly, if W is not at the right border of the remaining jobs, Properties (b) and (g) imply that we can utilize the full resource. Now consider the first round T such that the window W contains both $\min J(T-1)$ and $\max J(T-1)$. For simplicity, assume the remaining (by Property (b) at most $m-1$) jobs can be finished during round $T+1$. By the above observations, either the schedule assigns at least $m-2$ jobs their full resource requirement in all but the last rounds, or the schedule utilizes the full resource in all but the last rounds. In the former case we get an approximation ratio of $\frac{m}{m-2} = 1 + \Theta\left(\frac{1}{m}\right)$ (in each round, Opt can assign at most m jobs their full resource requirement). In the latter case the computed schedule is optimal (we always use the full resource). The bulk of the analysis goes towards proving that we can can always find such a maximal good window.

References

[1] André Brinkmann, Peter Kling, Friedhelm Meyer auf der Heide, Lars Nagel, Sören Riechers, and Tim Süß. Scheduling shared continuous resources on many-cores. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pages 128–137. ACM, 2014.

[2] Fan Chung, Ronald Graham, Jia Mao, and George Varghese. Parallelism versus memory allocation in pipelined router forwarding engines. *Theory of Computing Systems*, 39(6):829–849, 2006.

[3] Leah Epstein, Asaf Levin, and Rob Van Stee. Approximation Schemes for Packing Splittable Items with Cardinality Constraints. In *Algorithmica*, volume 62, pages 102–129. Springer, 2012.

[4] Leah Epstein and Rob van Stee. Improved results for a memory allocation problem. *Theory of Computing Systems*, 48(1):79–92, 2011.

Restricted Assignment Scheduling with Resource Constraints

Gyorgy Dosa* Hans Kellerer (Speaker) † Zsolt Tuza‡

1 Introduction

We are given a set $N = \{1, \ldots, n\}$ n of independent jobs that are to be scheduled on m parallel machines $M_1, \ldots, M_m$. In the *Restricted Assignment problem* (RA-problem, for short) each job j can be executed on a specific subset $\mathcal{M}(j)$ of the machines, and on those machines the processing time of job j is p_j. The objective is to minimize the makespan. In the three field notation, we abbreviate this problem by $R|p_{ij} \in \{p_j, \infty\}|C_{\max}$.

Assume that additionally there are μ renewable resources $R_1, \ldots, R_\mu$. Let Λ_k be the set of jobs which require resource R_k, and let λ_k denote the cardinality of set Λ_k, $k = 1, \ldots, \mu$. Job j requires simultaneous availability of all resources in the set $\mathcal{R}(j) \subseteq \{R_1, \ldots, R_\mu\}$ for processing; we denote by ρ_j the cardinality of $\mathcal{R}(j)$, $j = 1, \ldots, n$. Any resource can be used by only one job at any time. It means that two jobs which require the same resource cannot be processed simultaneously. This problem is abbreviated by $R|p_{ij} \in \{p_j, \infty\}, res\mu 11|C_{\max}$. In the following, we will call it *Restricted Assignment with Resources problem*, briefly RAR-problem. The *degree* of the problem is defined as the quantity $B = \max_{j=1,\ldots,n} \rho_j$, that is the maximum number of resources required by a job.

To the best knowledge of the authors, restricted assignment and resources were not considered previously together. The two types of conditions, however, have been studied separately.

The restricted assignment problem can be considered as a special case of the classical unrelated machine problem $R|\cdot|C_{\max}$, where job j on machine M_i has processing time p_{ij}. The paper of Lenstra et al. [3] contains a polynomial-time 2-approximation algorithm for $R|\cdot|C_{\max}$. On the negative side it is proven that one cannot get any worst-case ratio better than 3/2 for $R|\cdot|C_{\max}$ unless $\mathsf{P} = \mathsf{NP}$ holds.

There are many papers considering Multiprocessor Scheduling With Resources, MPSR for short. If each task requires at most one resource, i.e. $B = 1$, then MPSR with unit-time jobs admits a polynomial-time algorithm for an arbitrary number of processors, even with prescribed release times and deadlines [1]. On the other hand, for a variable number of resources the problem of minimizing maximum lateness becomes NP-hard, still with unit processing times of all the tasks, and already on just two processors [2].

*dosagy@almos.vein.hu. Department of Mathematics, University of Pannonia, H-8200 Veszprém, Egyetem u. 10, Hungary.

†hans.kellerer@uni-graz.at. Institut für Statistik und Operations Research, Universität Graz, Universitätsstraße 15, 8010 Graz, Austria.

‡tuza@dcs.uni-pannon.hu. Alfréd Rényi Institute of Mathematics, Hungarian Academy of Sciences, H-1053 Budapest, Reáltanoda u. 13–15, Hungary.

2 Our Results

We prove inapproximability results and design approximation algorithms for the RAR-problem. Our main negative result is that the problem with unit-time jobs is APX-hard, already on three machines. In the case that each job requires only a bounded number of resources, we design approximation algorithms with constant worst-case bound, without any restrictions on processing times. For some special cases (e.g., unit-time jobs with degree $B = 1$) we design optimal algorithms with polynomial running time.

To derive the main negative result, we prove a theorem on graph coloring, which seems to be of interest on its own right, too. It states APX-hardness of the chromatic number on a restricted class of graphs.

Our results can be stated as follows:

Theorem 1 *The RAR-problem with unit-time jobs and two machines can be solved in polynomial time.*

Theorem 2 *The RAR-problem with n unit-time jobs and m machines can be solved in $O((m^3 + m^2 n) \log n)$ time for $B = 1$.*

Theorem 3 *There is a polynomial-time $(2 - 1/m + B)$-approximation algorithm for the RAR-problem on m machines with arbitrary processing times.*

Theorem 4 *There is a PTAS for the RAR-problem with a fixed number of machines and $B = 1$.*

Theorem 5 *The following optimization problems are APX-complete:*

(*i*) *the* Chromatic Number *problem restricted to graphs of independence number 3,*

(*ii*) *the* Minimum Clique Cover *problem restricted to graphs whose clique number is 3,*

(*iii*) *even more restrictively the* Minimum Clique Cover *problem on graphs whose clique number is 3 and maximum degree is 4.*

Theorem 6 *The RAR-problem is APX-hard, even when it is restricted to the following type of instances: there are only three machines, all jobs have unit time, any job can be processed on any machine, and each resource is required only for two jobs.*

References

[1] J. Blazewicz (1979). *Deadline scheduling of tasks with ready times and resource constraints*, Inform. Process. Lett., 8 (2), 60–63.

[2] J. Blazewicz, J. Barcelo, W. Kubiak and H. Rock (1986). *Scheduling tasks on two processors with deadlines and additional resources*, Europ. J. Oper. Res., 26, 364–370.

[3] J. K. Lenstra, D. Shmoys and E. Tardos (1990). *Approximation algorithms for scheduling unrelated parallel machines*, Math. Progr., 46, 259–271.

Minimizing the Sum of Completion Times Over Scenarios

Thomas Bosman (Speaker) [*] Martijn van Ee [†] Csanád Imreh [‡]
Alberto Marchetti-Spaccamela [§] Martin Skutella [¶] Leen Stougie [‖]

1 Introduction

We consider scheduling over scenarios, driven by our curiosity how the mere existence of scenarios affects the complexity of problems compared to their traditional single scenario counterparts. In the sparse literature on optimization under scenarios, which has appeared under various names, problems have been identified whose multiple scenario versions are essentially harder than their (traditional) single scenario ones. An example is the shortest path problem with a scenario specified by the destination [2].

In the context of scheduling under scenarios we consider a scenario defined as a subset of a predefined and fully specified set of jobs. Specifically, we are given a set J of jobs and a set of K scenarios $\mathcal{S} = \{S_1, \ldots, S_K\}$, with $S_k \subseteq J$, $k = 1, \ldots, K$. The aim is to find a schedule of the whole set of jobs on machines, i.e. an assignment of all jobs to machines and on each machine an order of the jobs assigned to it, such that the schedule, obtained for the scenarios by simply skipping the jobs not in the scenario, optimizes a function of the scheduling objective over all scenarios.

Feuerstein et al. [1] introduced this model for scheduling problems, and obtained some intriguing results when considering the makespan as objective. In the "*robust*" version of the scenario scheduling problem the *maximum* makespan over all scenarios is minimized. This version we will refer to as the *minmax* version. In the "*stochastic*" version the *average* or, equivalently, the *sum* over all scenarios is minimized, which we will refer to as the *minsum* version.

The most natural next step is then to investigate the behaviour under scenarios of the next most commonly studied objective: the sum of completion times, often called *total completion time*. It is well known that minimizing total completion time on identical parallel machines is done in polynomial time by the *Shortest Processing Time first* (SPT) rule.

It is rather easy to adapt the proofs of the negative complexity results in [1] to obtain similar results for the multiple scenario total completion time problem, even if all jobs have unit processing time. However, for the minmax version the situation turns out to be more dramatic: for the problem with general processing times, even if there are

[*] t.n.bosman@vu.nl. Vrije Universiteit, Amsterdam, The Netherlands.
[†] m.van.ee@vu.nl. Vrije Universiteit, Amsterdam, The Netherlands.
[‡] Our co-author Csanád Imreh (Szeged) passed away with tragic suddenness on January 5th, 2017.
[§] alberto@diag.uniroma1.it. Università di Roma "La Sapienza", Italy.
[¶] martin.skutella@tu-berlin.de. Technische Universität Berlin, Germany.
[‖] l.stougie@vu.nl. Vrije Universiteit and CWI, Amsterdam, The Netherlands and Erable, France

only 3 scenarios, the multiple scenario total completion time problem is already NP-hard on 2 machines. If there are only 2 scenarios, the problem remains easy. Somewhat surprisingly, for the minsum version of the problem with general processing times the structure of optimal solutions can be exploited to arrive at a polynomial time dynamic programming algorithm for a fixed number of scenarios and a fixed number of machines. We conjecture that this is true in fact for any number of machines.

This raises the question if minsum scheduling problems that are easy in their single scenario version remain easy under a fixed number of scenarios. To show that even such a general statement does not hold, we introduce release dates and reservation costs in the total completion time scheduling problem, allow preemption and use only a single machine. Reservation costs means that we need to reserve a time unit if we wish to use that time unit for scheduling jobs in any of the scenarios. We show that the minsum version of this problem becomes NP-hard under 3 scenarios.

In the remainder we refer to the minmax version as MinMaxSTC (MinMax Scenario scheduling with Total Completion time objective) and to the minsum version as MinSumSTC.

2 Any number of scenarios makes it hard

As mentioned in the introduction, it is rather easy to adapt the proofs of the negative complexity results mentioned above in [1] to obtain similar results for the multiple scenario total completion time problem, even if all jobs have unit processing time. Notice that their single scenario versions are trivial.

Proposition 1 *For two machines and all jobs having unit processing times it is NP-hard to approximate MinMaxSTC within a factor* $2 - \varepsilon$.

Proposition 2 *For two machines and all jobs having unit processing times it is NP-hard to approximate MinSumSTC within ratio 1.011 even if all scenarios contain only two jobs.*

3 Minimizing the maximum over scenarios

In case of two scenarios, MinMaxSTC is polynomial time solvable on any number of machines. Actually, we will prove an even stronger result: we can find, in polynomial time, a schedule that minimizes the total completion time under each of the two scenarios. On the other hand, we show that the problem is already hard for 3 scenarios and 2 machines. This basically settles the complexity of MinMaxSTC.

Theorem 3 *For two scenarios a schedule that is optimal for both scenarios can be found in polynomial time.*

Theorem 4 *MinMaxSTC is NP-hard for 3 scenarios and* 2 *machines.*

4 Minimizing the sum over the scenarios

Theorem 5 *MinSumSTC can be solved in polynomial time for any fixed number of scenarios and any fixed number of machines.*

The algorithm is dynamic programming. Since we know the problem is hard for any number of scenarios (see Section 2), it only leaves open the question if the problem can be solved in polynomial time if we allow the number of machines to be part of the input. The following conjecture, which we strongly believe to hold, would give an affirmative answer to the question.

Conjecture 6 *MinSumSTC has an optimal solution such that for every scenario $k = 1, \ldots, K$ and each $j = 1, \ldots, n$, the j largest jobs are assigned to the machines in such a way that the difference in number of jobs assigned to each pair of machines is at most $k - 1$, or more formally, using A_i as the set of jobs assigned to machine i,*

$$\max_{i=\ldots,m} |\{h \in A_i \cap S_k : h \leq j\}| - \min_{i=\ldots,m} |\{h \in A_i \cap S_k : h \leq j\}| \leq k - 1.$$

5 Hardness of a minsum problem with small number of scenarios

Finally, we show that it is not generally true that minsum versions of scheduling problems that are easy in their single scenario versions, remain easy for a fixed number of scenarios. To this end, we consider a single machine problem with a release time r_j for each job j and allow jobs to be preempted. Less common is that we also introduce a reservation cost. We pay c for each time unit that we use for scheduling jobs in any of the scenarios; in other words, if we pay c for a time unit it can be used in all scenarios. The objective is to minimize the sum of the flow times over all scenarios and the total reservation cost. A little thought should make it clear that in the single scenario version the problem is solved by scheduling jobs according to the *Shortest Remaining Processing Time first* (SRPT) rule and reserving exactly $\sum_j p_j$ non-idle time units in that schedule.

Theorem 7 *The scheduling problem described above with 3 scenarios is NP-hard.*

6 Conclusion and open questions

We see as a main challenge to derive structural insights why multiple (fixed number of) scenario versions are sometimes as easy as their single scenario versions, like the scheduling problems that we have studied here, and for other problems become harder or even NP- or APX-hard.

References

[1] Esteban Feuerstein, Alberto Marchetti-Spaccamela, Frans Schalekamp, René Sitters, Suzanne van der Ster, Leen Stougie, and Anke van Zuylen. Minimizing worst-case and average-case makespan over scenarios. *Journal of Scheduling*, pages 1–11, 2016.

[2] Nicole Immorlica, David R. Karger, Maria Minkoff, and Vahab S. Mirrokni. On the costs and benefits of procrastination: approximation algorithms for stochastic combinatorial optimization problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 691–700, 2004.

Routing Open Shop: A Hierarchy of Superproblems

Ilya Chernykh (Speaker) * Ekaterina Lgotina †

1 Introduction

The routing open shop problem being a generalization of the two classic discrete optimization problems (*open shop* $Om||C_{\max}$ and *metric travelling salesman problem* ΔTSP) is considered. Note that both problems are strongly NP-hard in general case (see [9] and [6] respectively.

In the **open shop problem** [7] given a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ and a set of machines $\mathcal{M} = \{M_1, \ldots, M_m\}$, one has to schedule operations of each job for each machine in an arbitrary sequence in such manner that operations of each job are processed in separate time intervals. Processing times $p_{ji} \geqslant 0$ of operation of job J_j on machine M_i are given. Preemptions are not allowed. The goal is to obtain a feasible schedule minimizing the *makespan* (which in this case coincides with the maximum completion time $C_{\max}$).

The **metric TSP** can be described by a complete edge-weighted graph $G = \langle V, E\rangle$, where weight $\tau_{pq} \geqslant 0$ of edge $[p, q]$ represents the distance between nodes p and q. Distances satisfy the triangle inequality. The goal is to find a hamiltonian tour H in G with the minimal total weight $|H| \doteq \sum\limits_{[p,q]\in U} \tau_{pq}$.

The **routing open shop** problem was introduced in [1, 2]. It combines inputs of two problems mentioned in the following manner. Graph $G = \langle V, E\rangle$ (not necessary a complete one) represents some transportation network. Jobs from $\mathcal{J}$ are distributed between the nodes of graph G and have to be processed in an arbitrary order by each of machines from $\mathcal{M}$. The machines are considered to be mobile and are initially located at the specified node referred to as *the depot*. (The depot is the only node allowed to be job-free, although it may contain several jobs as any other node.) The machines travel with unit speed using the shortest routes between nodes and have to arrive at correspondent node to process the jobs from that node in open shop environment. After performing all the operations machines have to return back to the depot. As soon as each node with the possible exception of the depot contains at least one job it is necessary for each machine to visit every node of the graph. The goal is to minimize the *makespan*: the latest *release moment* over all machines, i.e. the moment when the machine has returned to the depot after processing all of its operations. We denote that makespan by $R_{\max}$ and using the classic three-field notation (see for example [8]) we denote the routing open shop problem as $ROm||R_{\max}$.

*idchern@math.nsc.ru. Sobolev Institute of Mathematics, 4 Koptyug ave., Novosibirsk, 630090, Russian Federation.

†kate.lgotina@gmail.com. Novosibirsk State University, 2 Pirogova Str, Novosibirsk, 630090, Russian Federation.

Being a generalization of two NP-hard problems $ROm||R_{\max}$ remains NP-hard in the case of a single machine and in the case of a single node with $m \geqslant 3$ machines. Surprisingly, it is also NP-hard in the simplest case $RO2|G = K_2|R_{\max}$, when we have only two machines and only two nodes of graph G (one of which being the depot) [1]. Detailed review can be found in [4] and references wherein.

In this paper we consider even further generalizations of the routing open shop problem introducing a hierarchy of superproblems. We will discuss conditions of reducibility between those supercases and review several known as well as new results reformulated for new supercases as general as possible.

2 Superproblems and main results

One possible line of generalization was introduced in [3]. It concerns individual travel times of machines between nodes of G. That can be achieved either by assigning different travel speed to each machine (*uniform travel time* or *Qtt*) or even considering *unrelated travel times* (or *Rtt*) for each pair (machine, edge). Another branch of generalization concerns replacing underlying open shop $Om||C_{\max}$ with so called *general open shop* problem $\bar{O}m||C_{\max}$, in which each job instead of having a single operation for each machine has a set of operations (possibly empty) to be performed by that machine. In correspondent generalization of the routing open shop $R\bar{O}m||R_{\max}$ each machine has specific set of nodes to visit.

These two branches can both be generalized by the superproblem in which each machine M_i has its own graph G_i with different edge weighting functions.

Results we are willing to discuss and generalize include (but not limited to) the following list:

1. Wide class of polynomially solvable subcases of $RO2|G = tree|R_{\max}$.
2. Linear time $\frac{6}{5}$-approximation algorithms for $RO2|G = K_2|R_{\max}$ [2] and $RO2|G = K_3|R_{\max}$ [5].
3. Linear time exact algorithm for $RO2|G = tree, variable - depot, Rtt|R_{\max}$ [3] (in that setting the location of the depot is not predefined and has to be chosen by a scheduler).

References

[1] I. Averbakh and O. Berman and I. Chernykh (2006). *The routing open-shop problem on a network: complexity and approximation.* European Journal of Operational Research, Vol. 173(2), pp. 521–539.

[2] I. Averbakh and O. Berman and I. Chernykh (2005). *A 6/5-approximation algorithm for the two-machine routing open shop problem on a 2-node network.* European Journal of Operational Research, Vol. 166(1), pp. 3–24.

[3] I. Chernykh (2016). *Routing open shop with unrelated travel times.* In: 9th International Conference on Discrete Optimization and Operations Research, DOOR 2016; Lecture Notes in Computer Science, Vol. 9869, pp. 272–283.

[4] I. Chernykh and A. Kononov and S. Sevastyanov (2013). *Efficient approximation algorithms for the routing open shop problem*. Comput. Oper. Res., Vol 40(3), pp. 841–847.

[5] I. Chernykh and E. Lgotina (2016). *The 2-machine routing open shop on a triangular transportation network*. In: 9th International Conference on Discrete Optimization and Operations Research, DOOR 2016; Lecture Notes in Computer Science, Vol. 9869, pp. 284–297.

[6] M.R. Garey and D.S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freemann and Company, San Francisco, CA.

[7] T. Gonzalez and S. Sahni (1976). *Open shop scheduling to minimize finish time*. J. Assoc. Comput. Mach., Vol. 23, pp. 665–679.

[8] E. L. Lawler and J. K. Lenstra and A. H. G. Rinnooy Kan and G. B. Shmoys (1993). *Sequencing and scheduling: algorithms and complexity*. Logistics of Production and Inventory, Elsevier, Amsterdam, pp. 445–522.

[9] D.P. Williamson et al (1997). *Short shop schedules*. Oper. Res., Vol. 45(2), pp. 288–294.

Scheduling Data Gathering in 2-Level Tree Networks

Joanna Berlińska *

1 Introduction

Gathering data from remote processors is an important step of many contemporary applications. The data may be obtained as a result of computations or by sensing the environment. They have to be collected together for analysis, processing and storing. Scheduling algorithms for data gathering networks were proposed, e.g., in [1, 2, 5, 7].

In this work we analyze data gathering in a 2-level tree network. The leaf nodes of the network transfer data to intermediate nodes, which preprocess and merge them, constructing new, intermediate datasets. These datasets have to be sent to a single base station. The base station processes them and stores final results. Our goal is to schedule communication and computations in the network so that the whole data gathering process takes the shortest possible time.

2 Problem formulation

The data gathering network consists of a single base station P_0, n intermediate nodes $P_1, \ldots, P_n$ and m leaf nodes P_{jk}. An intermediate node P_j gathers data from leaf nodes P_{jk} for $k = 1, \ldots, m_j$, where $m_1 + \ldots + m_n = m$. A subnetwork consisting of node P_j and all nodes P_{jk} for given j will be denoted by N_j. It is assumed that all network nodes have identical communication and computation capabilities, described by communication rate (inverse of speed) C and computation rate A.

At time $t = 0$ each node P_{jk} holds dataset D_{jk} of size α_{jk}, which should be sent to P_j (in time $C\alpha_{jk}$). At most one leaf node can communicate with intermediate node P_j at a time. After the transfer of dataset D_{jk} is completed, this dataset can be preprocessed by P_j, in time $A\alpha_{jk}$. The results produced by P_j for all datasets D_{jk} are concatenated and constitute a new dataset D_j of size $\alpha_j = \gamma \sum_{k=1}^{m_j} \alpha_{jk}$, where γ is a parameter describing the relationship between the sizes of initial and intermediate data. Node P_j needs time $C\alpha_j$ to transfer this dataset to the base station, which processes it in time $A\alpha_j$. At most one intermediate node can communicate with the base station at a time. It is assumed that each dataset can be sent in many separate pieces, i.e. communication preemptions are allowed. The scheduling problem is to minimize the total data gathering time T.

*Joanna.Berlinska@amu.edu.pl. Faculty of Mathematics and Computer Science, Adam Mickiewicz University in Poznań, Umultowska 87, 61-614 Poznań, Poland.

3 Results

Let us first note that solving our scheduling problem can be divided in two steps. In the first step a schedule for data gathering in each subnetwork N_j will be constructed separately. As communication in a subnetwork is sequential, the analyzed problem consists in makespan minimization in a 2-machine flow shop, where the communication network is the first machine, and node P_j is the second machine. The two operations of job k ($k = 1, \ldots, m_j$) are sending and preprocessing dataset D_{jk}. Therefore, the problem can be solved in $O(m_j \log m_j)$ time using Johnson's algorithm [6]. The minimum makespan computed for subnetwork N_j will be denoted by r_j. Thus, intermediate node P_j is ready to start transferring data to the base station at time r_j. The second step is constructing a schedule for sending and processing the intermediate results. As the communication between intermediate nodes and the base station is sequential, we have to solve a special case of another flow shop scheduling problem, $F2|r_j, pmtn|C_{max}$, which is known to be strongly NP-hard in general [4]. Let us remind that in our problem the execution times of two operations of job j are equal to $p_{1j} = C\alpha_j$ and $p_{2j} = A\alpha_j$, i.e. we consider a proportionate flow shop with different machine speeds.

We first show that if $\alpha_j = \alpha$ for $j = 1, \ldots, n$, i.e. all subnetworks N_j deliver the same amounts of data, then the shortest schedule can be computed in $O(n \log n)$ time. Namely, every time the communication network becomes idle or a new dataset is released, an available dataset with the shortest remaining transfer time should be chosen for sending. Note that $\alpha_j = \alpha$ does not mean that release times r_j have to be equal, since the subnetworks may contain different numbers of leaf nodes holding datasets of different sizes α_{jk} which sum up to α/γ.

For the general case with different α_j we analyze the following communication scheduling algorithm.

1. Let $J = \{1, 2, \ldots, n\}$ and $t = \min_{j \in J}\{r_j\}$.

2. Find the set of available jobs $A = \{j | j \in J, r_j \leq t\}$ and $t' = \min\{r_j | r_j > t, j \in J\}$. Choose a job j in A according to Johnson's rule [6].

3. Let $L = \min\{p_{1j}, t' - t\}$. If $p_{1j} \leq L$, schedule the transfer of dataset D_j in interval $[t, t + p_{1j})$ and set $J = J \setminus \{j\}$. In the opposite case schedule the transfer of dataset D_j in interval $[t, t + L)$ and set $p_{1j} = p_{1j} - L$.

4. If $J \neq \emptyset$, set $t = t + L$ and go to step 2.

Thus, every time the communication network becomes idle or a new dataset is released, the dataset to be transferred is chosen according to Johnson's rule. The datasets are processed by the base station in the order in which they are received. We show that

- If $C \leq A$, the above algorithm computes the shortest possible schedule, since our problem boils down to a special case of $F2|r_j, 1\text{-}min, pmtn|C_{max}$ [3].

- If $C > A$, the algorithm does not always produce the optimum solution.

The case with $C > A$ is further analyzed in order to identify additional conditions under which the considered algorithm is guaranteed to deliver optimum results.

4 Conclusions and future work

We showed that the analyzed data gathering scheduling problem can be solved in polynomial time if $C \leq A$ or $\alpha_j = \alpha$ for $j = 1, \ldots, n$. However, the complexity status of the case with $C > A$ and different α_j remains open and should be investigated in the future. A promising direction in constructing (exact or approximation) algorithms for this problem may be analyzing the relationship between intermediate dataset sizes α_j and their release times r_j.

References

[1] J. Berlińska, Communication scheduling in data gathering networks with limited memory, Applied Mathematics and Computation 235 (2014) 530-537.

[2] J. Berlińska, Scheduling for data gathering networks with data compression, European Journal of Operational Research 246 (2015) 744-749.

[3] J. Cheng, G. Steiner, P. Stephenson, A computational study with a new algorithm for the three-machine permutation flow-shop problem with release times, European Journal of Operational Research 130 (2001) 559-575.

[4] Y. Cho, S. Sahni, Preemptive Scheduling of Independent Jobs with Release and Due Times on Open, Flow and Job Shops, Operations Research 29 (1981) 511-522.

[5] K. Choi, T.G. Robertazzi, Divisible Load Scheduling in Wireless Sensor Networks with Information Utility, IEEE International Performance Computing and Communications Conference 2008: IPCCC 2008, 9-17.

[6] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, Naval Research Logistics Quarterly 1 (1954) 61-68.

[7] M. Moges, T.G. Robertazzi, Wireless Sensor Networks: Scheduling for Measurement and Data Reporting, IEEE Transactions on Aerospace and Electronic Systems 42 (2006) 327-340.

When Shop Scheduling Meets Dominoes, Eulerian and Hamiltonian Paths

Jean-Charles Billaut [*] Federico Della Croce (Speaker) [†] Fabio Salassa [‡]
Vincent T'kindt [§]

1 Introduction

In this paper, we consider no-idle and no-wait shop scheduling problems with some standard configurations namely flow shop, job shop and open shop. We focus on the makespan, referred to as $C_{\max}$, as a performance measure. More precisely, we tackle four distinct problems which, using the general three-field notation [4], can be denoted as $F2|no-idle, no-wait|C_{\max}$ for the 2-machine flow shop, $J2|no-idle, no-wait|C_{\max}$ for the for 2-machine job shop and $O2|no-idle, no-wait|C_{\max}$ for the 2-machine open shop. The m-machine flow shop problem is denoted as $F|no-idle, no-wait|C_{\max}$. We refer first to problem $F2|no-idle, no-wait|C_{\max}$: n jobs are available at time zero; each job j must be processed non-preemptively on two continuously available machines M_1, M_2 with integer processing times $p_{1,j}, p_{2,j}$, respectively. The processing order is $M_1 \rightarrow M_2$ for all jobs. Each machine processes at most one job at a time and operations of each job cannot overlap. For any sequence, $[j]$ denotes the job in position j. The objective is the minimization of the makespan.

With respect to the literature, to the best or our knowledge, few works have dealt with the no-idle and no-wait constraints simultaneously. In [1], it is mentioned that both problems $F2|no-idle|\sum C_j$ and $F2|no-wait|\sum C_j$ are NP-hard. Similar consideration holds for problem $F2|no-idle, no-wait|\sum C_j$. The relevant literature includes [3] where it is shown that minimizing the number of interruptions on the last machine is solvable in $O(n^2)$ time on two machines (the problem is denoted as $F2|no-wait|\mathcal{G}$) while it is NP-hard on three machines or more. We remark that problems $F2|no-wait|\mathcal{G}$ and $F2|no-idle, no-wait|C_{\max}$, even though close are not equivalent and an optimal solution with no interruption of problem $F2|no-wait|\mathcal{G}$ may be non-optimal for problem $F2|no-idle, no-wait|C_{\max}$. Consider a $2-job$ instance with processing times $p_{1,1} = b$, $p_{2,1} = a$, $p_{1,2} = a$, $p_{2,2} = b$, with $b > a$. Then, sequence $1-2$ is no-idle, no-wait, has makespan $C_{\max}^{1-2} = 2b + a$ and is optimal for problem $F2|no-wait|\mathcal{G}$ as it has no

[*] billaut@univ-tours.fr. Université Francois-Rabelais de Tours, Laboratoire d'Informatique (EA 6300), ERL CNRS OC 6305, 64 avenue Jean Portalis, 37200 Tours, France.
[†] federico.dellacroce@polito.it. Corso Duca degli Abruzzi, 24, 10129 Torino, Italy.
[‡] fabio.salassa@polito.it. Corso Duca degli Abruzzi, 24, 10129 Torino, Italy.
[§] tkindt@univ-tours.fr. Université Francois-Rabelais de Tours, Laboratoire d'Informatique (EA 6300), ERL CNRS OC 6305, 64 avenue Jean Portalis, 37200 Tours, France.

interruption. However, it is not optimal for problem $F2|no-idle, no-wait|C_{\max}$ as sequence $2-1$ is also no-idle, no-wait and has makespan $C_{\max}^{2-1} = 2a+b < 2b+a$.

In the next section we present some complexity results related to the four tackled shop scheduling problems.

2 Main results

We first point out that the $no-idle, no-wait$ constraint is very strong as it forces consecutive jobs to share common processing times, namely, any feasible solution for $F2|no-idle, no-wait|C_{\max}$, requires that $\forall j \in ..., n-1$, $p_{2,[j]} = p_{1,[j+1]}$. Figure 1 provides an example of a feasible no-idle, no-wait schedule for a 2-machine flow shop with four jobs.

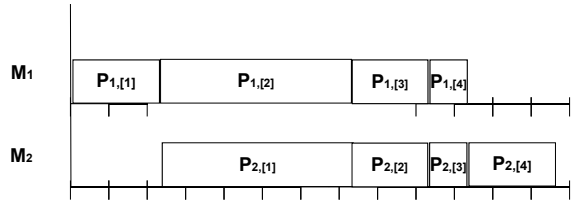


Figure 1: A no-idle no-wait schedule for a 2-machine flow shop

The peculiarity of the $no-idle, no-wait$ effect strictly links the above mentioned flow shop problem to the game of dominoes. Dominoes are 1 x 2 rectangular tiles with each 1 x 1 square marked with spots indicating a number. A traditional set of dominoes consists of all 28 unordered pairs of numbers between 0 and 6. We refer here to the generalization of dominoes presented in [2] in which n tiles are present, each of the tiles can have any integer (or symbol) on each end and not necessarily all pairs of numbers are present.

In [2], it is shown that the Single Player Dominoes (SPD) problem, where a single player tries to lay down all dominoes in a chain with the numbers matching at each adjacency, is polynomially solvable as it can be seen as the solution of an eulerian path problem on an undirected multigraph. Figure 2 shows the solution of an SPD problem with 12 tiles with numbers included between 0 and 6.

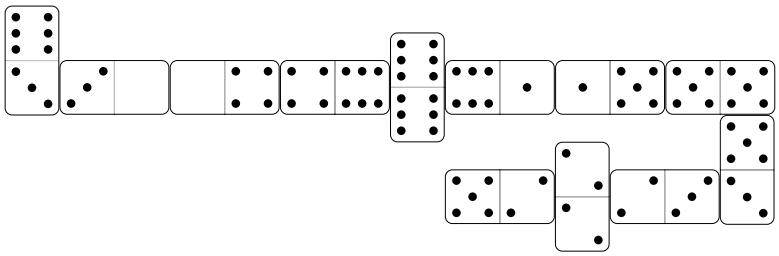


Figure 2: Solution of an SPD problem with 12 dominoes

We refer to the oriented version of SPD called $OSPD$ where all dominoes have an orientation, e.g. if the numbers are a and b, only the orientation $a \rightarrow b$ is allowed but

not viceversa. It is easy to show that also problem OSPD is polynomially solvable as it can be seen as the solution of an eulerian path problem on a directed multigraph.

The following proposition holds.

Proposition 1 *$F2|no-idle, no-wait|C_{\max} \propto OSPD$ and can be solved in $O(n)$ time.*

Problem $F2|no-idle, no-wait|C_{\max}$ is also linked to a special case of the Hamiltonian Path problem on a connected digraph. Consider a digraph $G(V, A)$ that has the following property: $\forall v_i, v_j \in V$, either $S_i \cap S_j = \emptyset$, or $S_i = S_j$ where we denote by S_i the set of successors of vertex v_i. In other words, each pair of vertices either has no common successor or has all successors in common. Let indicate the Hamiltonian path problem in that graph as the Common/Distinct Successors Hamiltonian Directed Path (CDSHDP) problem.

The following proposition holds.

Proposition 2 *$CDSHDP \propto F2|no-idle, no-wait|C_{\max}$. Correspondingly, problem $CDSHDP$ is polynomially solvable.*

The analysis leading to Proposition 1 can be extended also to the m-machine case. The following proposition holds.

Proposition 3 *Problem $F|no-idle, no-wait|C_{\max}$ is polynomially solvable.*

Finally, by reduction from the Numerical Matching with Target Sums (NMTS) problem, the following proposition holds.

Proposition 4 *Problems $J2|no-idle, no-wait|C_{\max}$ and $O2|no-idle, no-wait|C_{\max}$ are NP-Hard in the strong sense.*

References

[1] I. Adiri and D. Pohoryles. Flowshop / no-idle or no-wait scheduling to minimize the sum of completion times. *Naval Research Logistics*, 29: 495–504, 1982.

[2] Erik D. Demaine, Fermi Ma, and Erik Waingarten. Playing Dominoes Is Hard, Except by Yourself. *FUN 2014, LNCS*, 8496: 137–146, 2014.

[3] Wiebke Hohn, Tobias Jacobs, Nicole Megow. On Eulerian extensions and their application to no-wait flowshop scheduling. *Journal of Scheduling*, 15: 295–309, 2012.

[4] Lawler E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (1993), "Sequencing and Scheduling: Algorithms and Complexity" in *S.C. Graves, A.H.G. Rinnooy Kan and P. Zipkin (Eds.): Handbooks in Operations Research and Management Science vol 4: Logistics of Production and inventory, North-Holland, Amsterdam.*

Operating Room Scheduling with Variable Procedure Times

Macarena Azar (Speaker) Javiera Barrera Rodrigo A. Carrasco
Susana Mondschein *

1 Introduction

The operating theatre is one of the most expensive unit in hospitals, representing up to 40% of the total expenses [3]. Therefore, since 1960, a vast amount of research has addressed the management of operating rooms (see [8] and historical references therein). Two main problems have been the focus [5]: the reduction of overtime costs and, the under utilization of ORs .

The OR planning problem can be studied at different levels: strategic, tactical, and operational. We focus on the latter, where the randomness of surgery times have a significant impact on the efficiency of the system. At the operational level, stochastic programming models have been proposed to deal with the uncertainty of the surgery duration. For example, in [7] the schedule provided by the medical centre for one week is used as a base solution, which is improved using a local search heuristic that aims to increase the capacity utilization and reduce the risk of overtime. Similarly, in [10], the authors use an approximation method to add probabilistic capacity constraints, but requires a large dataset to tune. In [1] a two-stage stochastic MIP is formulated to assign a patient, a surgeon, and a block time to an operating room. The formulation also allows to decide the number of operating rooms that should be opened. Although these articles deal directly with the variability of perioperative times, the distribution only depends on the diagnosis and not on the physician performing the surgery.

On the other hand, the prediction of the surgical procedure times has been addressed in many articles (see [4] and reference therein). In [4] the authors mention that the main two factors, besides the diagnosis, to predict the perioperative times are the primary surgeon and the type of anesthetic. Similar conclusions are obtained in [11] and [2]. In all these articles the authors mention that more accurate prediction of the perioperative time should be used to improve the OR management, but recent surveys do not mention any work using this information. Even more, in [9] the authors point out that most studies, which assume certain variability of the surgery time, use the historical data to calibrate the stochastic models instead of using historical data to reduce the uncertainty. To the best of our knowledge, the only work that programs ORs using surgeons' information to predict surgery duration is [6]. They compare three classic data mining techniques to estimate the surgery duration, and propose an IP model to schedule surgeries in one OR during one week, assuming a deterministic perioperative time, that also depends on the surgeon assigned to the operation. This leaves one important open question: is

*{mazar;javiera.barrera;rax;susana.mondschein}@uai.cl. Faculty of Engineering and Sciences, Universidad Adolfo Ibáñez, Santiago, Chile.

it better to deal with the variability duration with a stochastic model without using surgeon information, or is it better to invest in that and reduce variability?

Our work has been motivated by the Instituto de Neurocirugía Dr Asenjo, in Santiago, Chile. The medical centre has provided historical data of their four ORs, dedicated exclusively to elective surgeries. Our data shows that surgeons can have significantly different surgery times, with differences up to 60% for the same surgery. In this article, we propose a time-indexed scheduling formulation with distribution related chance constrains to improve the scheduling of an operating theatre, using the information of each surgeon. Through simulations and the use of real instances, we report the performances for three different metrics for all the solutions, showing the importance of using historical data.

2 Problem Formulation using Chance Constraints

The problem setting is as follows. There are J ORs, and we discretize the time the ORs are open into T time-steps of size Δt. We have I patients, where each patient i has a set K_i of physicians that can perform the surgery, an average time of surgery p_{ik} when performed by physician k, and a real (random) time of surgery S_{ik}. Each physician $k \in \{1, \ldots, K\}$ is available only within time-steps $[a_k, b_k]$. The objective is to program surgeries on all J ORs, assigning patients and physicians in a way such that the probability that any OR has overtime is controlled.

Let x_{ijkt} be 1 if patient i is assigned to have her surgery in OR j, which physician k, starting at time-step t; and 0 otherwise. To tackle this problem we develop a time-index formulation using these decision variables and average operation times, such that only one procedure can be done in each OR at each time; each physician can only perform one procedure at the time; each patient can be operated by one physician at most; surgeries are assigned to physicians only in the time-window where each physician is available; and that surgeries cannot finish after the ORs are closed.

The problem with the previous formulation is that ,even when overtime is not allowed, because we are using average surgery times, overtime might occur in the actual schedule. In order to reduce this occurrence, we propose using chance constraints to account for the probability distribution of the surgery times. In particular, we would like to control the probability of an overtime occurring. Let S_j denote the real (random) total surgery times for the day in OR j. We would like to add a constraint of the form $\mathbb{P}(S_j \geq T) \leq \epsilon$. In the general case, where we do not have enough information of the distribution of each surgery, we can use Hoeffding's inequality which is of the form $\mathbb{P}(S_j - \mathbb{E}[S_j] \geq t) \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$, where $[a_i, b_i]$ represents the support of the i-th random variable with $i = 1, \ldots, n$, such that they sum up to S_j. Hence, the following constraint for each OR j will guarantee the required condition:

$$\exp\left(-\frac{2\left(T - \mathbb{E}[S_j]\right)^2}{\sum_{i=1}^{I}\sum_{k=1}^{K}\sum_{t=1}^{T} x_{ijkt}(b_{ik} - a_{ik})^2}\right) \leq \epsilon. \tag{1}$$

Through a series of transformations and adding auxiliary variables, we write (1) as

$$\sum_{i=1}^{I}\sum_{k=1}^{K}\sum_{t=1}^{T}\left(\frac{1}{2}\ln(\epsilon)(b_{ik} - a_{ik})^2 + p_{ik}^2 - 2Tp_{ik}\right)x_{ijkt} + 2\sum_{i=1}^{I}\sum_{l=i+1}^{I}\sum_{k=1}^{K} p_{ik}p_{lk}y_{kjil} \geq -T^2,$$

plus some additional constraints that link x_{ijkt} and y_{kjil}.

In the setting where we applied our methodology, surgeries show a uniform distribution, in which case we can further improve Hoeffding's bound with a simpler and faster formulation. Additionally, we can also deal with lognormal distributions, which are the ones generally reported in several articles. In that case, we use a Laplace transform to compute the bounds numerically, and later add constraints to the optimization problem.

Finally, we analyse the practical performance of our method, showing that there is an important improvement in the performance of the operative schedules once the random surgery times are considered.

References

[1] Batun, S., and Begen, M. A. Optimization in healthcare delivery modeling: Methods and applications. In *Handbook of Healthcare Operations Management*. Springer, 2013, pp. 75–119.

[2] Batun, S., Denton, B. T., Huschka, T. R., and Schaefer, A. J. Operating room pooling and parallel surgery processing under uncertainty. *INFORMS Journal on Computing 23*, 2 (2011), 220–237.

[3] Denton, B., Viapiano, J., and Vogl, A. Optimization of surgery sequencing and scheduling decisions under uncertainty. *Health care management science 10*, 1 (2007), 13–24.

[4] Dexter, F., Dexter, E. U., Masursky, D., and Nussmeier, N. A. Systematic review of general thoracic surgery articles to identify predictors of operating room case durations. *Anesthesia & Analgesia 106*, 4 (2008), 1232–1241.

[5] Erdogan, S. A., Denton, B., and Fitts, E. Surgery planning and scheduling. *Wiley Encyclopedia of Operations Research and Management Science* (2010), 1–13.

[6] Gomes, C., Almada-Lobo, B., Borges, J., and Soares, C. Integrating data mining and optimization techniques on surgery scheduling. In *International Conference on Advanced Data Mining and Applications* (2012), Springer, pp. 589–602.

[7] Hans, E., Wullink, G., Van Houdenhoven, M., and Kazemier, G. Robust surgery loading. *European Journal of Operational Research 185*, 3 (2008), 1038–1050.

[8] Magerlein, J. M., and Martin, J. B. Surgical demand scheduling: a review. *Health services research 13*, 4 (1978), 418.

[9] Samudra, M., Van Riet, C., Demeulemeester, E., Cardoen, B., Vansteenkiste, N., and Rademakers, F. E. Scheduling operating rooms: achievements, challenges and pitfalls. *Challenges and Pitfalls (March 2016)* (2016).

[10] Shylo, O. V., Prokopyev, O. A., and Schaefer, A. J. Stochastic operating room scheduling for high-volume specialties under block booking. *INFORMS Journal on Computing 25*, 4 (2013), 682–692.

[11] Stepaniak, P. S., Heij, C., and De Vries, G. Modeling and prediction of surgical procedure times. *Statistica Neerlandica 64*, 1 (2010), 1–18.

A Constraint Generation Procedure for Pre-Runtime Scheduling of Integrated Modular Avionic Systems

Mathias Blikstad [*] Emil Karlsson [†] Tomas Lööw [‡]
Elina Rönnberg (Speaker) [§]

1 Background

A modern aircraft hosts lots of advanced electronics in the form of sensors that gather information, units where the information is processed, actuators that control the aircraft, and equipment that presents information to the pilot. During operation, this information is updated repeatedly, giving rise to a complex flow of data between different units and thus putting requirements on when different activities are allowed to be executed.

Electronics in an aircraft is called avionics. During the last two decades, the majority of the avionics industry has gone from using federated systems to using an integrated architecture called Integrated Modular Avionics (IMA) where applications share hardware resources on a common avionic platform. Such an architecture necessitates strict requirements on the spatial and temporal partitioning of the system to achieve fault containment, and a common standard for this partitioning is ARINC 653, see [1]. One way to establish a temporal partitioning is through pre-run time scheduling of the system, which involves creating a schedule for both tasks and a communication network. The introductory parts of the PhD-thesis [2] provide an extensive introduction to the area of scheduling of avionic systems.

In the process of designing an avionic system, new software functionality is developed and added to the system iteratively during a project of several years. Whenever a change is made in a software component, the scheduling tool has to provide a new schedule for the system or, if it fails, preferably produce a proof of infeasibility. If no feasible schedule exist, either changes of the software or upgrades of the avionics platform are needed. Due to the rigid certification processes in the aircraft industry it is extremely costly to upgrade the platform, and therefore it is important to utilise the existing platform in an efficient way and make upgrades only when necessary. The frequency of use and the importance of the outcome of the scheduling gives the scheduling tool a vital role in the process of designing an avionic system.

Most approaches for pre-runtime scheduling of large-scale real-time systems are of a heuristic nature, see references in [3]. For many types of real-time systems, a primal

[*] mathias.blikstad@saabgroup.com. Saab AB, SE-581 88 Linköping, Sweden.
[†] emil.karlsson@liu.se. Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden and Saab AB, SE-581 88 Linköping, Sweden.
[‡] Saab AB, SE-581 88 Linköping, Sweden.
[§] elina.ronnberg@liu.se. Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden and Saab AB, SE-581 88 Linköping, Sweden.

heuristic might be an efficient and sound way to provide a schedule. This does however not hold in our setting when the scheduling also involves the challenge to determine whether a desired software functionality can be implemented with the existing platform or not. If only a primal heuristic is applied and it fails to provide a solution, one does not know if this depends on shortcomings of the heuristic or if it is due to the fact that no feasible solution exists.

Our work contributes in the direction of developing optimisation based approaches for scheduling of large and complex future avionics systems, and we address the problem from a Mixed Integer Programming (MIP) point of view. The research is carried out in collaboration between Linköping University and the Swedish defence and security company Saab. Even for small sized instances of practical relevance, a straightforward MIP-model for the full scheduling problem contains a hundred million binary variables, which makes it computationally intractable. Our solution strategy is a constraint generation procedure that is tailored to exploit known characteristics of our problem.

2 A sketch of the system design and problem formulation

The system under consideration is distributed and at each node there is a set of modules. One of these, the *Communication module* (CM), is responsible for both the inter-node and the intra-node communication as well as the communication with external systems. The tasks on these modules are many and execute once per system period. The responsibility of the other modules, the *Application Modules* (AMs), is to run applications (software processes). The tasks on these modules are relatively few and execute 64 times during a system period. Each task has an interval in which to execute and the system is partitioned in the sense that all tasks are beforehand assigned to modules. Precedence dependencies are used to restrict the distance from one task instance to another task instance, within a system period or between two succeeding periods. Chains are used to prescribe that task instances shall execute in a specified order within the duration of one system period.

The nodes communicate over a *Communication network* (CN) which is a switched Ethernet that supports multicast. It is designed such that messages are assigned to, and sent in, discrete time slots in which they are allowed to use the full bandwidth. To send a message involves a chain of task instances that needs to execute on the CMs involved. There are two types of scheduling decisions to be made; tasks are sequenced and assigned a start time while communication is scheduled by assigning messages to time slots.

3 Solution strategy and computational results

For the industrially relevant instances considered, the computational challenge stems from the large number of tasks to be sequenced. It is also known that a large portion of the tasks have a fixed start time and that the other requirements, like task intervals and precedence dependencies, are not particularly tight or difficult to manage from a computational point of view. This knowledge is exploited in the solution strategy where a reformulation of the sequencing part of the model is made in a way which gives a large amount of constraints instead of a large amount of variables. Most of these constraints are redundant and the problem can therefore be solved without explicitly including all of them. Instead we apply a constraint generation procedure where we use

the Gurobi Optimizer Version 6.5.1 both for solving the relaxed problem and for solving the subproblem that detects constraints to add to the relaxed problem.

For the computational evaluation we have considered three instances, named A, B, and C. Instance A has two nodes and a total of about 6500 task instances. Instance B has 5 nodes and a total of about 15 000 task instances and Instance C is the largest with 7 nodes and a total of about 21 000 task instances. For these, the portion of fixed task instances is 64%, 54%, and 53%, respectively.

As benchmark for the results to be presented, we have tried to solve Instance A by applying Gurobi to our original formulation of the problem (after applying all our customised pre-processing components). Within a time limit of one week, no feasible solution was found. When using our constraint generation procedure, a schedule is obtained for Instance A within about two minutes and the results are not very sensitive to the parameter settings of the method. This exceptionally good outcome is likely to depend on that there exists many feasible solutions for this instance and that none of the relaxed constraints need to be generated in order for the relaxed problem to produce a feasible solution.

Instance B and C are significantly more challenging than Instance A. The computational times reported ranges between 14 minutes and 24 hours for Instance B and between 19 minutes and 12 hours for Instance C, depending on parameter settings of the method. The range in result is due to the fact that our approach for generating constraints is expensive, making the method sensitive to the number of constraint generation iterations needed. We find that these results merely verify that we can solve industrially relevant instances within reasonable time and our conclusion is that the approach is viable for this type of problem. Our continued research aims at improving the components used in this solution strategy to further enhance the computational performance.

Acknowledgements

This work was supported by the Swedish Armed Forces, the Swedish Defence Materiel Administration and the Swedish Governmental Agency for Innovation Systems under grant number NFFP6-2014-00917. The work is also part of the project *Operations research methods for large scale scheduling and resource allocation problems* funded by the Center for Industrial Information Technology (CENIIT) at Linköping University. The work of Emil Karlsson is supported by the Research School in Interdisciplinary Mathematics at Linköping University.

References

[1] Airlines electronic engineering committee (AEEC). Avionics application software standard interface, *ARINC specification 653, (part 1)*. 2006.

[2] A. Al-Sheikh. *Resource allocation in hard real-time avionic systems. Scheduling and routing problems*. PhD thesis, INSA de Toulouse, 2011.

[3] M. Blikstad, E. Karlsson, T. Lööw, E. Rönnberg. *An optimisation approach for pre-runtime scheduling of tasks an communication in an integrated modular avionic system*. Unpublished manuscript, 2016.

Single Machine Projections

Alexander Tesch *

1 Introduction

Given a set of jobs $\mathcal{J} = \{1, \dots, n\}$, we consider any scheduling problem of the form:

$$\min f(C),\ C \in P$$

where $C \in \mathbf{R}^n_{>0}$ is a vector of job completion times and $f(C) : \mathbf{R}^n_{>0} \mapsto \mathbf{R}_{>0}$ is a monotone cost function with $f(C') \leq f(C)$ for all $C' \leq C$. The set of feasible job completion times P can be any subset of $\mathbf{R}^n_{>0}$. In this article, we aim to construct valid inequalities for P that are of the form $\sum_{j\in\mathcal{J}} a_j C_j \geq b$. For this, we assume knowledge on the values $C_{max}(S) = \min_{C\in P} \max_{j\in S} C_j$ for every subset $S \subseteq \mathcal{J}$. For many scheduling problems, these values can be computed efficiently, either exactly or approximately. Except for single machine [2] and parallel machine scheduling [3], general valid inequalities of the form $\sum_{j\in\mathcal{J}} a_j C_j \geq b$ are rather unknown for other scheduling problems. The inequalities studied in this paper can be applied to many different scheduling problems such as unrelated parallel machines, resource-constrained project scheduling, flow-shop and others including (non-)preemption and generalizations to release dates and precedence constraints. For some scheduling problems we are able to state the first non-trivial inequalities of this kind. Our general idea is based on a generalization of the classical single machine characterization of Queyranne [2]. In particular, we show that any scheduling problem of the above form can be interpreted as many individual single machine problems whose associated inequalities are valid for the original problem P.

2 Single Machine Projections

Assume the values $C_{max}(S) = \min_{C\in P} \max_{j\in S} C_j$ for all $S \subseteq \mathcal{J}$ with $S \neq \emptyset$ which can be interpreted as optimal makespan values, if only jobs in the subset S are considered. Notably, the computation of $C_{max}(S)$ can already be NP-hard for many scheduling problems. However, there exists a couple of examples where $C_{max}(S)$ can be computed efficiently, especially compared to other cost functions, see for example $P|pmtn|C_{max}$ and $P|pmtn|\sum w_j C_j$, or $1|r_j, pmtn|C_{max}$ and $1|r_j, pmtn|\sum w_j C_j$. From the viewpoint of complexity theory, it suggests that $C_{max}(S)$ is an easier problem in general.

In the following write $x(S) = \sum_{i\in S} x_i$ for some vector $x \in \mathbf{R}^n$. We call a vector $\pi \in \mathbf{R}^n_{\geq 0}$ of job processing times *admissible*, if $\pi(S) \leq C_{max}(S)$ holds for every job subset $S \subseteq \mathcal{J}$. Thus, the set of all admissible processing times is given by

$$P_A = \{\pi \in \mathbf{R}^n_{\geq 0} : \pi(S) \leq C_{max}(S)\ \ \forall S \subseteq \mathcal{J}, S \neq \emptyset\} \tag{1}$$

*tesch@zib.de. Zuse Institute Berlin (ZIB), Takustraße 7, 14195 Berlin, Germany

which defines an n-dimensional polytope, since $0 \notin P$ and therefore $C_{max}(S) > 0$ for all $S \subseteq \mathcal{J}$ with $S \neq \emptyset$. The polytope P_A is *down-monotone*, that means for every vector $0 \leq \pi' \leq \pi$ with $\pi \in P_A$ we also have that $\pi' \in P_A$. In the presence of release dates, it is possible to modify P_A to exploit supermodularity as given in [1, 3]. Similar to the single machine inequalities of Queyranne [2], define the function

$$g(\pi) = \frac{1}{2}\left(\sum_{j\in\mathcal{J}} \pi_j\right)^2 + \frac{1}{2}\sum_{j\in\mathcal{J}} \pi_j^2 \tag{2}$$

which, in contrast to the original inequalities, takes a vector as argument instead of a subset $S \subseteq \mathcal{J}$. Our results are mainly built on the following theorem.

Theorem 1 *For every $\pi \in P_A$ the inequality $\sum_{j\in\mathcal{J}} \pi_j C_j \geq g(\pi)$ is valid for P.*

In particular, every $\pi \in P_A$ induces one separate single machine instance for which the single machine inequality $\sum_{j\in\mathcal{J}} \pi_j C_j \geq g(\pi)$ holds. Theorem 1 says that the inequality is not only valid for this single machine problem but also for our original problem, that is P. Hence, we translate a certain part of our problem into one single machine problem, generate the corresponding inequality and transfer it back to our original problem. In contrast to the original single machine inequalities, the inequality $\sum_{j\in\mathcal{J}} \pi_j C_j \geq g(\pi)$ does not need to be stated for every subset $S \subseteq \mathcal{J}$ because, by the down-monotonicity of P_A, we may set $\pi_j = 0$ for every $j \in \mathcal{J} \setminus S$, which yields the desired inequality $\sum_{j\in S} \pi_j C_j \geq \frac{1}{2}\pi(S)^2 + \frac{1}{2}\pi^2(S) = g(\pi)$. Therefore, any subset inequality is covered by a separate admissible processing time vector $\pi \in P_A$.

For a more visual interpretation in a Gantt-chart setting, the vectors $\pi \in P_A$ can be seen as follows. For any feasible completion time vector $C \in P$ consider any projection onto the one-dimensional time line, that means every time point gets assigned exactly one job that is processed at this time or a dummy job if no job is processed. Any such projection defines a single machine instance with projected processing time vector $\tilde{\pi} \in \mathbf{R}^n_{\geq 0}$. Then it holds $\pi \leq \tilde{\pi}$ for all $\pi \in P_A$, meaning that every admissible processing time vector yields a single machine lower bound for every single machine projection. This justifies the term of $\pi \in P_A$ being a valid *single machine projection*. In particular, define the set of completion times that are valid for all $\pi \in P_A$ by

$$P_C = \{C \in \mathbf{R}^n_{\geq 0} : \sum_{j\in\mathcal{J}} \pi_j C_j \geq g(\pi) \ \ \forall \pi \in P_A\}. \tag{3}$$

From Theorem 1 it follows that P_C is a relaxation of P, that is $P \subseteq P_C$. The other inclusion does not hold in general. Since $g(\pi) = \pi^T B \pi$ where $B \in \mathbf{R}^{n\times n}$ is a positive semidefinite matrix, one can show that P_C is indeed a polyhedron and the vertices $\pi \in V(P_A)$ of the polytope P_A are sufficient to describe P_C.

3 Separation

Given a completion time vector $C \in \mathbf{R}^n_{>0}$, the separation problem of P_C is to compute a vector $\pi \in P_A$ such that $\Gamma(\pi) = g(\pi) - \sum_{j\in\mathcal{J}} \pi_j C_j > 0$. Since $g(\pi)$ is convex, this problem corresponds to a convex maximization problem over a polytope, which is known to be NP-hard in general. However, for the special case of $\Gamma(\pi)$ we are able to state

necessary optimality conditions. From that, we derive a discrete optimization problem that computes a separating inequality for P_C. In order to do this efficiently, one also requires to perform separation for P_A efficiently. But since P_A depends on the $C_{max}(S)$ values, the separation of P_A is also NP-hard in general. Therefore, let $\tilde{P}_C$ be defined as in (3) but according to the set $\tilde{P}_A \subseteq \mathbf{R}^n_{\geq 0}$. We verify that for every $\tilde{P}_A \subseteq P_A$ it holds $P_C \subseteq \tilde{P}_C$ which means that $\tilde{P}_C$ is also a valid relaxation for $P \subseteq P_C$. Therefore, one approach is to consider lower bound values $L(S) \leq C_{max}(S)$ for every $S \subseteq \mathcal{J}$ in order to obtain the relaxation $\tilde{P}_A = \{\pi \in \mathbf{R}^n_{\geq 0} : \pi(S) \leq L(S)\ \forall S \subseteq \mathcal{J}, S \neq \emptyset\} \subseteq P_A$. Preferably, the values $L(S)$ are chosen such that $\tilde{P}_A$ can be separated in polynomial time, for example by a simpler polytope. Depending on the original problem P, this is often given by linear programming relaxations for $C_{max}(S)$ that function as value oracles to separate $\tilde{P}_A$. Another approach is to focus on discrete relaxations $P^*_A = \{\pi^1, \ldots, \pi^m\} \subseteq P_A$. If $P^*_A = V(P_A)$ this yields an exact separation method for P_C but since one cannot expect to compute $V(P_A)$ efficiently, another possibility is vertex enumeration of the relaxation $\tilde{P}_A$, that is $P^*_A = V(\tilde{P}_A)$. For the special case of $f(C) = \sum_{j \in \mathcal{J}} w_j C_j$, we propose a reduction to a set of non-dominated points π of $\tilde{P}_A$ because all induced points π_S with $S \subseteq \mathcal{J}$ and $\pi_{S,j} = \pi_j$ for all $j \in S$ and $\pi_{S,j} = 0$ for all $j \in \mathcal{J} \setminus S$ can be considered implicitly because the associated inequalities define a linear program with coupling constraints and m polymatroidal blocks. This can be solved, for example, by *Dantzig-Wolfe Decomposition* and the greedy algorithm as subroutine.

4 Application to Scheduling Problems

The proposed methods can be applied generically to a broad range of scheduling problems because they solely depend on the values $C_{max}(S)$ that can be computed, or approximated respectively, more or less efficiently for each individual scheduling problem. We show that certain dual LP-relaxations for unrelated parallel machines and the resource-constrained project scheduling problem with makespan objective yield suitable relaxations $\tilde{P}_A \subseteq P_A$ for their respective completion time polyhedra. Moreover, the associated inequalities subsume some previously proposed parallel machine inequalities [3]. A further interesting application is single machine scheduling with sequence dependent setup times which is known to be equivalent to the famous traveling salesman problem. There we obtain new valid inequalities in the arrival time variables by using dual information from a linear programming relaxation of this problem.

References

[1] M. X. Goemans (1996). *A supermodular relaxation for scheduling with release dates.* In International Conference on Integer Programming and Combinatorial Optimization (pp. 288-300). Springer Berlin Heidelberg.

[2] M. Queyranne (1993). *Structure of a simple scheduling polyhedron.* Mathematical Programming, 58(1), 263-285.

[3] M. Queyranne, A.S. Schulz (1994). *Polyhedral approaches to machine scheduling.* TU-Berlin, Fachbereich 3.

Minimizing Maximum Flow Time on Related Machines via Immediate Dispatch and Dynamic Pricing

Sungjin Im* Benjamin Moseley† Kirk Pruhs‡ Clifford Stein (Speaker) §

1 Introduction

In this paper we give an online algorithm for minimizing the maximum flow time on related machines using dynamic posted prices. In doing so, we prove two main results. First, we give an immediate dispatch algorithm for minimizing maximum flow time on related machines that has a better competetive ratio than previous best result for this problem [BC15]. Then, we show a general result about when dynamic posted prices can be used in an on-line scheduling algorithm. Applying both results to the problem of minimizing maximum flow time on related machines yields the final result.

In the well-known related machines environment, we are given a set of m machines with speeds, $s_1, \ldots, s_m$. We assume, without loss of generality that $s_1 < s_2 < \ldots s_m$. We are also given a set of n jobs with release dates r_j and processing times p_j. A non-preemptive schedule chooses a machine i on which to run each job j, the job is processed for p_j/s_i time units and ultimately has some completion time C_j. The *flow time* of the job is $F_j = C_j - r_j$, and the objective is to minimize the maximum flow time $F_{\max} = \max_j F_j$. Maximum flow time is a generalization of makespan, and if all jobs have identical release dates then it just reduces to makespan. Maximum flow time is also releated to deadline scheduling problems, as a maximum flow time of T means that each job completes by time $r_j + T$, so one is putting a bound on the amount of time that any job spends in the system.

Until recently, there was no $O(1)$-competetive algorithm for minimizing maximum flow time on related machines. Using resource augmentation, Anand et. al. [ABF$^+$13] gave a $(1+\epsilon)$-speed $O(1/\epsilon)$ algorithm for the more general problem of scheduling to minimize maximum flow time on *unrelated machines*. (Related machines are a special case of unrelated machines in which p_{ij} the processing time of job j on machine i is $p_{ij} = p_j/s_i$). They also showed that, for unrelated machines, any algorithm without resource augmentation is $\Omega(m)$-competetive. For the weighted maximum flow time objective, they also gave a $(1+\epsilon)$-speed $O(1/\epsilon^3)$ competetive algorithm for related machines and showed that no $O(1)$-speed, $O(1)$-competetive algorithm exists in the unrelated setting. Finally, Bansal and Cloostermans[BC15] recently gave a 13.5-competetive algorithm for

*sim3@ucmerced.edu Electrical Engineering and Computer Science, University of California at Merced, Merced, CA 95344, USA

†bmoseley@wustl.edu Washington University in St. Louis, St. Louis, MO 63130, USA

‡kirk@cs.pitt.edu Dept. of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA.

§cliff@ieor.columbia.edu. Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027, USA.

related machines. This result is also the first $O(1)$-approximation even for the offline version of minimizing maximum flow time on related machines.

An Immediate Dispatch Algorithm An online algorithm is called *immediate dispatch* if, when a job j arrives at time r_j, we immediately decide which machine j will run on. We do not need to start job j at time r_j, but we must make an irrevocable decision about which machine it will eventually run on. Immediate dispatch algorithms have some practical advantages. For example, we do not need to maintain a global queue of unprocessed work, as we immediately send a job to a machine; the machines can then operate independently in their scheduling decisions. The algorithm of Bansal and Cloostermans is not immediate dispatch, and they mention in their paper that it is not clear that such an algorithm exists.

In this work, we give an immediate dispatch algortihm for miminimizing maximum flow time on related machines. Moreover, our immediate dispatch algorithm has a better competetive ratio, as we prove the following theorem:

Theorem 1 *There is an immediate dispatch algorithm for minimizing maximum flow time on related machines with a competetive ratio of* $25/3$.

We briefly describe the algorithm and omit the analysis. Assume that the algorithm knows Opt, the value of the optimum flow time. If not, we can use a standard doubling trick. For a constant ϵ, we define *epochs* to be of length $\epsilon\mathsf{Opt}$ and we divide time into a series of epochs. We also will have constants α and β to be chosen later. At the start of an epoch we assign jobs that arrived in the last epoch, and assign them in non-increasing order of processing time. Let j be a job that is to be assigned. Let $[i_j, m]$ be the (indices of the) set of machines on which p_j/s_i is at most Opt. (Observe that because we have sorted machines by speed, and because we assume we know Opt, the set of machines must form an interval and must be non-empty.) If there is a machine in $[i_j, m]$ with load less than $\alpha\mathsf{Opt}$, then schedule j on the slowest such machine. Otherwise, if there is a machine in $[i_j, m]$ with load less than $\beta\mathsf{Opt}$ then schedule j on the slowest such machine. Otherwise the algorithm has a proof that the current value of Opt is too small. We then choose α and β appropriately, and are able to obtain the bound in the theorem.

Posted Pricing Schemes There has been a great deal of work on mechanism design for scheduling. One natural class of simple mechanisms are *posted pricing schemes*. (see e.g. [CEFJ15, FGL15].) In a dynamic posted pricing scheme for an online scheduling problem, before the jth job arrives, a vector $g_j = (g_j^1, \ldots, g_j^m)$ is published, where g_j^i is the price that the jth job must pay to run on the machine i. If L_{ij} is load on the ith machine when the jth job arrives and, p_{ij} is the processing time of the jth job on machine i, then job j is assigned to the machine that minimizes $L_{ij} + p_{ij} + c_{ij}$, that is, the job selfishly assigns itself to the machine that minimizes its flow time plus cost. It is important that the prices for the jth job only depend on the first $j-1$ jobs and the scheduling decision made up to that point; they do not depend on any characteristics of the jth job or any knowledge (even distributional) about any future jobs.

We now define a monotone algorithm. Let $A_t(p)$ the speed of the machine that an algorithm A would assign a job of size p to if it was released at the current time t. Then an algorithm is monotone if for all possible instances, and for all possible t, $A_t(p)$ is monotone nondecreasing in p.

We show a connection between immediate dispatch, monotone algorithms and dynamic pricing algorithms.

Lemma 2 *An immediate dispatch, monotone algorithm A for a scheduling problem can be converted into a dynamic posted pricing algorithm.*

This lemma shows a general connection that goes beyond the particular problem that is the subject of this paper.

Combining these results, we obtain a dynamic pricing, $O(1)$-competetive algorithm for maximum flow on related machines.

References

[ABF+13] 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 - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 13–24, 2013.

[BC15] Nikhil Bansal and Bouke Cloostermans. Minimizing maximum flow-time on related machines. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, pages 85–95, 2015.

[CEFJ15] Ilan Reuven Cohen, Alon Eden, Amos Fiat, and Lukasz Jez. Pricing online decisions: Beyond auctions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 73–91, 2015.

[FGL15] Michal Feldman, Nick Gravin, and Brendan Lucier. Combinatorial auctions via posted prices. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 123–135, 2015.

Better Unrelated Machine Scheduling for Weighted Completion Time via Random Offsets from Non-Uniform Distributions

Sungjin Im (Speaker) * Shi Li †

1 Introduction

Minimizing total (weighted) completion time is one of the most popular scheduling objectives that have been extensively studied. The scheduler must assign each job j to a machine i and complete it. We consider two settings, preemptive and non-preemptive schedules. In the non-preemptive setting, each job must be completed without interruption once it starts getting processed. On the other hand, in the preemptive setting, each job's processing can be interrupted to process other jobs and be resumed later. In both cases, job j's completion time is, if j is assigned to machine i, defined as the first time when the job gets processed for $p_{i,j}$ units of time. Then, the objective is to minimize $\sum_{j \in J} w_j C_j$. These two non-preemptive and preemptive versions can be described as $R|r_j|\sum_j w_j C_j$ and $R|r_j, pmtn|\sum_j w_j C_j$ respectively, using the popular three-field notation in scheduling literature. Both versions of the problem are strongly NP-hard even in the single machine setting [5], and are APX-hard even when all jobs are available for schedule at time 0 [4], in which case preemption does not help.

For the non-preemptive case, Skutella gave a 2-approximation based on a novel convex programming [9], which improved upon the $(2+\epsilon)$-approximation based on linear programming [6]. It has been an outstanding open problem if there exists a better than 2-approximation [9, 7, 6, 10]. In particular, it is listed in [7] as one of the top 10 opens problems in the field of approximate scheduling algorithms; see the Open Problem 8. When jobs have no arrival times, i.e. $r_{i,j} = 0$ for all i, j, very recently Bansal et al. [2] gave a better than 1.5-approximation in a breakthrough result, improving upon the previous best 1.5-approximations due to Skutella [9] and Sethuraman and Squillante [8]. In fact, the Open Problem 8 consists of two parts depending on whether jobs have release times or not. Bansal et al. [2] solved the first part of Open Problem 8, and the second part still remained open.

1.1 Our Results

In this paper, we answer the second part of the open problem in the affirmative by giving a better than 2-approximation.

*sim3@ucmerced.edu. Electrical Engineering and Computer Science, University of California, Merced, 5200 N. Lake Road, Merced, CA 95344, USA.

†shil@buffalo.edu. Department of Computer Science and Engineering, University at Buffalo, 1 White Road, Buffalo, NY 14260, USA.

Theorem 1 *For a constant $\alpha < 1.8786$, there exists an α-approximation for $R|r_j|\sum_j w_j C_j$.*

Surprisingly, we give this result by rounding a very simple and natural LP that has not been studied in previous works. Our LP can be viewed as a stronger version of the time-indexed LP in [6], by taking the non-preemption requirement into consideration. However, even with this stronger LP, the rounding algorithm in [6] does not yield a better than 2-approximation, and we believe this is why the previous works overlooked this simple LP. Improving the 2-approximation ratio requires not only the stronger LP, but also novel rounding algorithm and analysis.

Our result also gives a positive answer to the conjecture made by Sviridenko and Wiese [10]. They considered a configuration LP where there is a variable for every machine $i \in M$ and subset of jobs $S \subseteq J$. The variable is associated with the optimal total weighted completion time of the jobs in S on machine i. They showed that one can solve their LP within a factor of $1+\epsilon$, but could not give a better than 2-approximation, conjecturing that their LP have an integrality gap strictly less than 2.

Indeed, one can show that the configuration LP of [10] is the strongest among all convex programmings of the following form: minimize $\sum_{i\in M} f_i(x_i)$ subject to $\sum_{i\in M} x_{i,j} = 1$ for every $j \in J$ and $x_{i,j} \geq 0$ for every $i \in M, j \in J$, where $x_i = (x_{i,j})_{j\in J} \in [0,1]^J$ and f_i is some convex function over $[0,1]^J$ such that if $x_i \in \{0,1\}^J$, then $f_i(x_i)$ is at most the total weighted completion time of scheduling jobs $\{j : x_{i,j} = 1\}$ optimally on machine i. All results mentioned in this paper (including our results) are based on programmings of this form and thus the configuration LP is the strongest among them. Hence, our result gives a 1.8786 upper bound on the integrality gap of the configuration LP.

With a solution to the configuration LP, one can derive a natural independent rounding algorithm. For each job j, independently assign j to a machine i with probability $x_{i,j}$. Then for every machine i, we schedule all jobs assigned to i; this can be done optimally if all release times are 0, and nearly optimally (within $(1+\epsilon)$ factor) in general [1, 3]. When all jobs have release time 0, the algorithm gives a 1.5-approximation. However, [2] showed this independent rounding algorithm can not give a better than 1.5-approximation, which motivated them to develop a clever dependent rounding algorithm.

For $R|r_j|\sum_j w_j C_j$, the independent rounding algorithm is known to give a 2-approximation [6, 9]. In contrast to the status for $R||\sum_j w_j C_j$, no matching lower bound was known for this algorithm. Our result indirectly shows that the independent rounding can achieve 1.8786-approximation. Thus we do not need to apply the sophisticated dependence rounding scheme of [2], which only led to a tiny improvement on the approximation ratio for $R||\sum_j w_j C_j$. We complement our positive result by showing that the independent rounding algorithm can not give an approximation ratio better than $e/(e-1) \approx 1.581$.

Theorem 2 *There is an instance for which the independent rounding gives an approximation ratio worse than $e/(e-1) - \epsilon \geq 1.581 - \epsilon$ for any $\epsilon > 0$.*

We continue to study the preemptive case. In the preemptive case, two variants were considered in the literature depending on whether jobs can migrate across machines or must be completed scheduled on one of the machines. If migration is not allowed, the work in [6] still gives a $(2+\epsilon)$-approximation since the LP therein is a relaxation for

preemptive schedules but the rounding outputs a non-preemptive schedule. If migration is allowed, [9] gives a 3-approximation. Our main result for the preemptive case is the first better than 2-approximation when migration is not allowed.

Theorem 3 *For a constant $\alpha < 1.99971$, there exists an α-approximation for $R|r_j, pmtn| \sum_j w_j C_j$.*

We note that our algorithm is based on a stronger linear programming relaxation. The configuration LP of [10] is for non-preemptive schedules hence not usable for preemptive schedules. Our LP is a different type of configuration LP where there are variables for each job's complete schedules. While we use an LP for preemptive schedules, we output a non-preemptive schedule.

References

[1] Foto Afrati, Evripidis Bampis, Chandra Chekuri, David Karger, Claire Kenyon, Sanjeev Khanna, Ionnis Milis, Maurice Queyranne, Martin Skutella, Cliff Stein, et al. Approximation schemes for minimizing average weighted completion time with release dates. In *FOCS*, 1999.

[2] Nikhil Bansal, Ola Svensson, and Aravind Srinivasan. Lift-and-round to improve weighted completion time on unrelated machines. In *STOC*, 2016.

[3] Leslie A Hall, Andreas S Schulz, David B Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of operations research*, 22(3):513–544, 1997.

[4] Han Hoogeveen, Petra Schuurman, and Gerhard J Woeginger. Non-approximability results for scheduling problems with minsum criteria. *INFORMS Journal on Computing*, 13(2):157–168, 2001.

[5] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Annals of discrete mathematics*, 1:343–362, 1977.

[6] Andreas S Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15(4):450–469, 2002.

[7] Petra Schuurman and Gerhard J Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *J. of Sched.*, 2(5):203–213, 1999.

[8] Jay Sethuraman and Mark S Squillante. Optimal scheduling of multiclass parallel machines. In *SODA*, pages 963–964, 1999.

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

[10] Maxim Sviridenko and Andreas Wiese. Approximating the configuration-lp for minimizing weighted sum of completion times on unrelated machines. In *Integer Programming and Combinatorial Optimization*, pages 387–398. Springer, 2013.

Online Min-Sum Flow Scheduling with Rejections

Giorgio Lucarelli (Speaker) * Nguyen Kim Thang † Abhinav Srivastav ‡
Denis Trystram *

1 Introduction

A well-identified issue in online computation is the weakness of the worst case paradigm which underestimates the performance of an online algorithm. Many algorithms, which perform well in real-world, are known to admit a mediocre theoretical guarantee. Conversely, many theoretical sound algorithms behave poorly even in simple practical settings. The need of more accurate models is considered as a high importance in algorithmic community. We are interested in resource augmentation models in the context of scheduling. Kalyanasundaram and Pruhs [5] proposed the *speed augmentation* model, where an online algorithm is compared against an adversary with slower processing speed, while Phillips et al. [7] proposed the *machine augmentation* model in which the algorithm uses more machines than the adversary. Choudhury et al. [3] introduced the *rejection* model where an online algorithm is allowed to discard a small fraction of jobs.

In this paper, we study the problems of preemptive and non-preemptive online scheduling of jobs on unrelated machines in order to minimize the average time a job remains in the system. Both problems are known to be non-approximable by a constant factor [4]. The preemptive variant has been extensively studied under the different resource augmentation models [1, 2]. The non-preemptive variant is much less explored. An $O(\frac{1}{\epsilon})$-competitive algorithm has been presented in [6] if both an ϵ-speed augmentation is used and an ϵ-fraction of jobs is rejected. We are interested here in exploring the power of the rejection model and in eliminating the need for speed augmentation in the latter result. On the road to this, we show how to replace speed augmentation with rejection in the preemptive variant. Our analysis is based on the dual-fitting paradigm.

Problem definition. We are given a set $\mathcal{M}$ of m unrelated machines where jobs arrive *online*, that is we learn about the existence and the characteristics of a job only after its release. Let $\mathcal{J}$ denote the set of all jobs of our instance, which is not known a priori. Each job $j \in \mathcal{J}$ is characterized by its *release time* r_j, while if j is executed on machine $i \in \mathcal{M}$ then it has a *processing time* p_{ij}. Moreover, each job has to be dispatched to one machine at its arrival and migration is not allowed. Given a schedule, let C_j denote the *completion time* of the job j. The *flow-time* of j is defined as $F_j = C_j - r_j$, that is the total time that j remains in the system. Our objective is to create a schedule that minimizes the total flow-times of all jobs, i.e., $\sum_{j \in \mathcal{J}} F_j$.

*{giorgio.lucarelli,denis.trystram}@imag.fr. Univ. Grenoble Alpes, LIG, INRIA, France
†thang@ibisc.fr. IBISC, University of Evry, France
‡abhinav.srivastav@univ-grenoble-alpes.fr. Verimag, Univ. Grenoble Alpes, France

2 Linear Programming Formulation

For each machine $i \in \mathcal{M}$, job $j \in \mathcal{J}$ and time $t \geq r_j$, we introduce a binary variable $x_{ij}(t)$ which indicates if j is processed on i at time t. We consider the following linear programming relaxation and the corresponding dual program.

$$\min \sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{J}}\int_{r_j}^{\infty}\frac{1}{p_{ij}}(t-r_j+p_{ij})x_{ij}(t)dt$$
$$\sum_{i\in\mathcal{M}}\int_{r_j}^{\infty}\frac{x_{ij}(t)}{p_{ij}}dt \geq 1 \quad \forall j\in\mathcal{J}$$
$$\sum_{j\in\mathcal{J}} x_{ij}(t) \leq 1 \quad \forall i\in\mathcal{M}, t$$
$$x_{ij}(t)\in[0,1]$$

$$\max \sum_{j\in\mathcal{J}}\lambda_j - \sum_{i\in\mathcal{M}}\int_0^{\infty}\gamma_i(t)dt$$
$$\frac{\lambda_j}{p_{ij}} - \gamma_i(t) \leq (t-r_j+p_{ij}) \quad \forall i\in\mathcal{M}, j\in\mathcal{J}, t\geq r_j$$
$$\lambda_j, \gamma_i(t) \geq 0$$

We will interpret the rejection model in the above primal and dual programs as follows. We assume that the algorithm is allowed to reject a set $\mathcal{R}$ of jobs. This corresponds to sum up in the primal objective only on the set of the non-rejected jobs.

3 Preemptive Scheduling

The algorithm. Each job is immediately dispatched to a machine upon its arrival. We denote by $Q_i(t)$ the set of pending jobs at time t dispatched to machine i, i.e., the set of jobs dispatched to i that have been released but not yet completed or rejected at t. Let $q_{ij}(t)$ be the *remaining processing time* at time t of a job j that is dispatched to machine i. For each machine i, our *scheduling policy* is the following: at each time t we execute on i the job $j \in Q_i(t)$ with the smallest remaining processing time in $Q_i(t)$. In case of ties, we select the job that arrived the earliest. Moreover, we maintain a counter c_i (initialized to 0) for each machine i. Every time a job is dispatched to i, c_i is increased by 1. Then, the *rejection policy* is the following: given an arbitrary small constant $\epsilon \in (0,1)$, whenever c_i reaches $1/\epsilon + 1$, we reject the job with the largest remaining processing time from $Q_i(t)$ and reset c_i to 0. Note that the rejected job does not immediately disappear from the system. We say that a job ℓ is *definitively rejected* at time $t + \sum_{j\in Q_i(t)} q_{ij}(t) + q_{i\ell}(t)$, that is at the time that it supposed to be completed. We denote by $R_i(t)$ the set of jobs dispatched to i that are rejected but not yet definitively at time t. Let Δ_{ij} be the increase in the total flow-time occurred in the schedule of our algorithm following the above scheduling and rejection policies if we decide to dispatch a new job j to machine i. Then, the *dispatching policy* is the following: we dispatch j to the machine where Δ_{ij} is minimum.

Dual variables. Based on the dispatching policy, we set $\lambda_j = \min_i \Delta_{ij}$. Let $W_i(t)$ be the total number of jobs dispatched to machine i that are either pending or not yet definitively rejected until t, i.e., $W_i(t) = |Q_i(t)| + |R_i(t)|$. We set $\gamma_i(t) = W_i(t)/(1+\epsilon)$. Based on this definition, we can guarantee that, given any fixed time t, $\gamma_i(t)$ does not decrease due to rejections since the jobs remain in $R_i(t)$ for sufficient time after their rejection. Then, the following theorem holds.

Theorem 1 *Given any $\epsilon \in (0,1)$, there is an $O(\frac{1}{\epsilon})$-competitive algorithm that rejects at most an ϵ-fraction of jobs.*

4 Non-preemptive Scheduling

The algorithm. We enhance the previous algorithm in order to adapt it to a non-preemptive environment where a job is considered to be successfully executed only if its execution is performed without any interruption. The *scheduling policy* for each machine i is the following: at each time t when i is idle, we start executing on i the job that has the smallest processing time in $Q_i(t)$; in case of ties, we select the job that arrived the earliest. We use two rejection rules. The *first rejection policy* is identical with the preemptive case. For the second rejection rule, we maintain another counter v_j for each job j which is initialized to 0 when the execution of j begins and it is increased by one each time a new job is dispatched to machine i while j is executing. The *second rejection policy* is the following: we reject the job j when $v_j = 1/\epsilon$. The *dispatching policy* is based again on the increase in the total flow-time in the same vein as for the preemptive case.

Dual variables. We set $\lambda_j = \epsilon \cdot \min_i \Delta_{ij}$. Let $Q_i(t)$ be the set of pending jobs a time t dispatched to i. We define the set $R_i(t)$ of jobs that have been rejected due to the first rejection policy but not yet definitively. For each job j, let D_j be the set of jobs that are rejected due to the second rejection policy after r_j and before its completion or rejection. Let j_k denote the job released when the job k is rejected due to the second rejection policy. A job j dispatched to machine i is *definitively finished* $\sum_{k \in D_j} q_{ik}(r_{j_k})$ time after its completion or rejection. Let $U_i(t)$ be the set of jobs that are dispatched to machine i and are already completed or rejected due to the second rejection policy but not yet definitively finished at time t. We set $\gamma_i(t) = \frac{\epsilon}{1+\epsilon}(|Q_i(t)| + |R_i(t)| + |U_i(t)|)$.

Theorem 2 *Given any $\epsilon \in (0,1)$, there is an $O(\frac{1}{\epsilon^2})$-competitive algorithm that rejects at most a 2ϵ-fraction of jobs.*

References

[1] S. Anand, N. Garg and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA* , pages 1228–1241, 2012.

[2] A. R. Choudhury, S. Das and A. Kumar. Minimizing weighted L_p-norm of flow-time in the rejection model. In *FSTTCS*, vol. 45 of LIPIcs, pages 25–37, 2015.

[3] A. R. Choudhury, S. Das, N. Garg and A. Kumar. Rejecting jobs to minimize load and maximum flow-time. In *SODA*, pages 1114–1133, 2015.

[4] N. Garg and A. Kumar. Minimizing average flow-time : Upper and lower bounds. In *FOCS*, pages 603–613, 2007.

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

[6] G. Lucarelli, N. K. Thang, A. Srivastav and D. Trystram. Online non-preemptive scheduling in a resource augmentation model based on duality. In *ESA*, vol. 57 of LIPIcs 2016.

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

Integration of Vehicle Maintenance Scheduling and Single Dead-End Track Parking on a Multi-Week Planning Horizon

Murat Elhüseyni (Speaker) * Ali Tamer Ünal †

1 Introduction

We handle the problem of vehicle maintenance scheduling under service level agreement (SLA) and single dead-end track on a multi-week planning horizon. We show that even only the hangar parking scheduling at the track is NP-Hard in the strong sense.

We build a MILP model to solve the problem. Thereafter, we derive some optimality conditions for scheduling of vehicle in the track. We create a discrete-event simulation environment to test the model. In this environment, we determine critical jobs, namely maintenance job list at each week. We employ the MILP model to schedule the jobs in the list. The list is updated with respect to the schedule and carried to next week.

In the literature, there are studies on vehicle maintenance scheduling problems in airway ([4], [2], [5]) and bus operations [1]. Our model introduces a maintenance scheduling problem where there are dead-end tracks in the hangar.

2 Formulation

In the formulation we define the concept of a block which models the movement restriction of a set of vehicles due to the dead-end track. Constraints 2 - 16 account for block assignment of jobs and their start and finish times at the parking schedule. Constraints 17 - 19 deal with tardiness of critical jobs and earliness of preventive jobs. Continuous time variables are linked to discrete time variables via 20 and 21. 22 - 26 model maintenance begin time, active maintenance times and end time. 27 is a soft constraint for SLA demand satisfaction. 28 and 29 render preventive and corrective jobs as out of service if their upkeep operation has not begun till a given threshold.

*murat.elhuseyni@boun.edu.tr. Industrial Engineering Department, Bogaziçi University, 3432, Istanbul, Turkey.

†unaltam@boun.edu.tr. Industrial Engineering Department, Bogaziçi University, 3432, Istanbul, Turkey.

$$\min \sum_{j \in PM} \left(T_j + E_j + penymax_+ \triangle Y_j^{max+} \right) + pencor_+ \sum_{j \in U} T_j + \sum_{t \in T} pen_- \triangle SLA_t^- \quad (1)$$

s.t.

$$\sum_{b \in B} x_{bj} + np_j = 1 \qquad j \in CR \quad (2)$$

$$\sum_{j \in CR} x_{bj} \leq 1 \qquad b \in B \quad (3)$$

$$\sum_{j \in CR} x_{bj} p_j \leq P_b \qquad b \in B \quad (4)$$

$$\sum_{b \in B_i} y_{ab} \leq 1 \qquad a \in B_{i+1}, i<|C| \quad (5)$$

$$y_{ab} \leq \sum_{j \in CR} x_{bj} \qquad b \in B_i, a \in B_{i+1}, i<|C| \quad (6)$$

$$y_{ab} \leq \sum_{j \in CR} x_{aj} \qquad b \in B_i, a \in B_{i+1}, i<|C| \quad (7)$$

$$x_{aj} \leq \sum_{b \in B_i} y_{ab} \qquad j \in CR, a \in B_{i+1}, i<|C| \quad (8)$$

$$F_b \geq S_b + P_b \qquad b \in B \quad (9)$$

$$S_b \geq \sum_{j \in CR} x_{bj} Y_j^{min} \qquad b \in B \quad (10)$$

$$F_a \leq F_b + M(1 - y_{ab}) \qquad b \in B_i, a \in B_{i+1}, i<|C| \quad (11)$$

$$S_b \leq S_a + M(1 - y_{ab}) \qquad b \in B_i, a \in B_{i+1}, i<|C| \quad (12)$$

$$S_{b+1} \geq F_b \qquad b \in B_i, i \in C \quad (13)$$

$$S_j \geq Y_j^{min}(1 - np_j) \qquad j \in CR \quad (14)$$

$$S_j = \sum_{b \in B} B_{bj} \qquad j \in CR \quad (15)$$

$$C_j = \sum_{b \in B} L_{bj} \qquad j \in CR \quad (16)$$

$$T_j \geq S_j + |T| - d_j - |T|(1 - np_j) \qquad j \in CR \quad (17)$$

$$T_j - \triangle Y_j^{max+} \leq Y_j^{max} - d_j \qquad j \in PM \quad (18)$$

$$E_j \geq d_j - S_j - |T| np_j \qquad j \in PM \quad (19)$$

$$\sum_{t \in T} te_{jt} = C_j \qquad j \in CR \quad (20)$$

$$\sum_{t \in T} tb_{jt} = S_j \qquad j \in CR \quad (21)$$

$$\sum_{t \in T} a_{jt} = C_j - S_j \qquad j \in CR \quad (22)$$

$$\sum_{t\in T} b_{jt} + np_j = 1 \qquad j \in CR \qquad (23)$$

$$\sum_{t\in T} e_{jt} + np_j = 1 \qquad j \in CR \qquad (24)$$

$$a_{jt} \leq 1 - \sum_{l=1}^{t} e_{jl} \qquad j \in CR, t \in T \qquad (25)$$

$$a_{jt} \leq \sum_{l=1}^{t} b_{jl} \qquad j \in CR, t \in T \qquad (26)$$

$$\sum_{j\in CR} 1 - (a_{jt} + o_{jt}) + \triangle SLA_t^- \geq SLA_t - |NC| \qquad t \in T \qquad (27)$$

$$o_{jt} = 1 - \sum_{i=r_j^{min}}^{t} b_{ji} \qquad j \in PM, t \geq r_j^{max} \qquad (28)$$

$$o_{jt} = 1 - \sum_{i=r_j^{min}}^{t} b_{ji} \qquad j \in U, t \geq r_j^{min} \qquad (29)$$

$$x_{bj}, y_{ab}, a_{jt}, e_{jt}, b_{jt}, np_j, o_{jt} = \{0, 1\} \qquad a, b \in B, j \in CR, t \in T \qquad (30)$$

$$P_b, S_b, F_b, B_{bj}, L_{bj}, T_j, C_j, S_j \geq 0 \qquad b \in B, j \in CR \qquad (31)$$

3 Results

We introduce a number of valid inequalities to strengthen the formulation. We also propose a heuristic to generate an initial feasible solution for the MIP solver. We report results of our numerical experiments that measure for performance of our model under various shop conditions and solution parameters.

References

[1] Ali Haghani and Yousef Shafahi (2002). *Bus maintenance systems and maintenance scheduling: Model formulations and solutions.* Transportation Research Part A: Policy and Practice, 36:453–482

[2] Andreas Gavranis and George Kozanidis (2015). *An Exact Solution Algorithm for Maximizing the Fleet Availability of an Aircraft Unit Subject to Flight and Maintenance Requirements.* European Journal of Operational Research, 242:631–643

[3] G. Keysan, G.L. Nemhauser and M. W. P. Savelsbergh (2010). *Tactical and Operational Planning of Scheduled Maintenance for Per-Seat, On-Demand Air Transportation.* Transportation Science, 44:291–306

[4] Philip Cho (2011). *Optimal Scheduling of Fighter Aircraft Maintenance.* Massachussets Institute of Technology, Msc thesis

[5] N. Safaei, D. Banjevic and A. K. S. Jardine (2011). *Workforce-constrained maintenance scheduling for military aircraft fleet: a case study.* Annals of Operations Research, 186:291–306

Three Models and a Set of Dominance Rules for the Speed Meeting Problem

Benoit Cantais (Speaker) * Antoine Jouglet * David Savourey *

1 Introduction

In a speed meeting problem, people are gathered in a place where tables are disposed to meet eachother. The set of persons that each person wishes to meet is known. At regular intervals, the persons are asked to get up and are redistributed among the tables. A distribution of persons among the tables is called a *round*.

We consider the problem with M tables of C seats, T meeting rounds and a set a persons $X = \{1, \ldots, N\}$. $G = (X, U \subseteq X^2)$ is an oriented graph such as $(i, j) \in U$ means that person i wishes to meet person j. A meeting $(i, j) \in U$ is considered as realized if persons i and j are seated at the same table during at least one meeting round. At round $t \in \{1, \ldots, T\}$, each person i can be seated at only one table and at most C persons can be seated at table $m \in \{1, \ldots, M\}$. Given the number of rounds T, the goal is to distribute the persons around the tables at every round to maximise the total number of wished meetings realized.

As far as we know, this problem has not been treated yet. However, some particular cases of this problem are close to well known problems of the literature. In the speed dating problem [2], the special case where the capacity of the tables is 2 is considered. In the fully social golfer problem [1], every person has to meet every other person exactly once.

In this talk, we present three models for the speed meeting problem that can be used in a branch and bound algorithm and a set of dominance rules.

2 Three models for the speed meeting problem to use in a branch and bound algorithm

Three models used in depth first branch and bound algorithms are presented to find an optimal solution to the speed meeting problem. In the *person* model, at every branch of the search tree, a person i is assigned with a table m at a round t. In the *couple* model, at every branch of the search tree, a wished meeting between two persons $(i, j) \in U$ is assigned with a table m at a round t. Consequently, i and j are added to table m and others wished meetings between i, j and the persons already seating at the table are done. In both cases, a partial distribution of the persons among the tables for each round is associated to a node of the search tree (see example Figure 1).

*{benoit.cantais,antoine.jouglet,david.savourey}@hds.utc.fr. Sorbonne Universités, Université de Technologie de Compiègne, France, Heudiasyc UMR CNRS 7253.

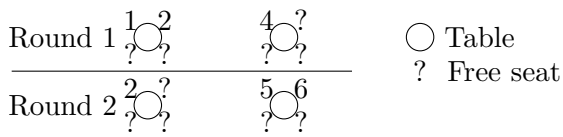


Figure 1: A partial solution P^{table} for the *couple* or *person* model.

In the *round* model, at every branch of the search tree, a round number t is assigned with a wished meeting (i, j). The table number where the meeting takes place is not specified.

Let $G_t^{round} = (X, U_t^{round})$ a graph such as $(i, j) \in U_t^{round} \iff (i, j)$ is associated with t. Two connected vertices i and j mean that persons i and j are seated at the same table during round t. For each round t, X can be partitionned into $\{X_{t,1}, \ldots, X_{t,k}\}$, the k connected components in G_t^{round}. $|X_{t,i}| = 1$ means that the unique person of the singleton is not seated during round t. $|X_{t,i}| > 1$ means that the persons described in the subset are seated at the same table during round t (see example Figure 2).

Round	Meetings assigned	Partition
1	$(2,4),(2,7),(3,5),(6,8)$	$\{\{1\},\{2,4,7\},\{3,5\},\{6,8\}\}$
2	$(1,3),(2,6),(4,7),(5,7),(5,8)$	$\{\{1,3\},\{2,6\},\{4,5,7,8\}\}$

Figure 2: A partial solution P^{round} and the associated partitions for the *round* model.

The distribution of the persons between M tables of C seats for each round t can be transformed into a 1D-BinPacking problem with M boxes of capacity C and a set of k objects such as the size of object $i, i \in \{1, \ldots, k\}$ is $|X_{t,i}|$. The sub-problem associated with every round must be solved to check if a partial solution for the *round* model is feasible (see example Figure 3).

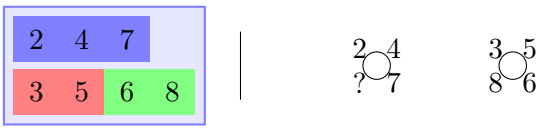


Figure 3: The BinPacking solution associated with round 1 of P^{round}.

3 A set of dominance rules for the speed meeting problem

A unique identifier number $i \in \{1, \ldots, X\}$ is used to identify each person. The first set of rules introduces an order based on the identifier numbers associated with the persons between tables and between rounds.

Theorem 1 *Let $T_{m,t}(P)$ be the smallest identifier number associated with a person who is seated at table m, round t in P. $\forall t \in \{1, \ldots, T\}, \forall (i, j) \in \{1, \ldots, M\}^2$ such as $i < j$, there is an optimal solution P^{opt} with $T_{i,t}(P^{opt}) < T_{j,t}(P^{opt})$.*

Theorem 2 *Let $R_{t,m,i}(P)$ be the i^e smallest identifier number associated with a person who is seated at table m, round t in P. $\forall t \in \{1, \ldots, T-1\}, \forall i \in \{2, \ldots, min(c_{t,1}, c_{t+1,1})\}$*

there is an optimal solution P^{opt} *with* $R_{t,1,i}(P^{opt}) > R_{t+1,1,i}(P^{opt}) => R_{t,1,i-1}(P^{opt}) < R_{t+1,1,i-1}(P^{opt})$.

Figure 4 gives an example of a partial solution P_1 that dominates P_2 and P_3 according to these dominance rules.

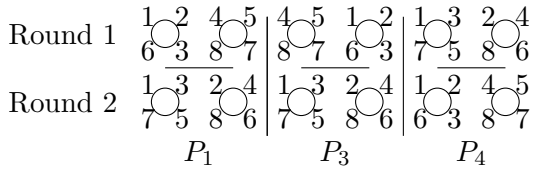


Figure 4: P_1 dominates P_2 and P_3.

Theorem 3 *Let* $M_{m,t}(P)$ *be the smallest number of meetings involving a person who is seated at table* m*, round* t *in* P*. There is an optimal solution* P^{opt} *such as* $\forall m \in \{1,\ldots,M\}, \forall t \in \{1,\ldots,T\}, M_{m,t}(P^{opt}) > 0$.

There is an interest to seat a person at a table only if this person wishes to meet someone seating at the table or someone seating at the table wishes to meet this person.

Theorem 4 *Let* P_1 *be a partial solution with* t *a finished round. Let* (i,j) *be a wished meeting feasible during round* t *in* P_1*. If* (i,j) *is not realized in* P_1 *or* (i,j) *is realized at round* $t' > t$ *then there is a solution* P_2 *at least as good as* P_1 *such as* (i,j) *is realized during round* t.

A round is considered as *finished* if no more assignment will be done. In Figure 5, the first round is considered as finished in P_1 but meetings $(2,9)$ (realized later) and $(1,4)$ (not realized) could have been realized during this round.

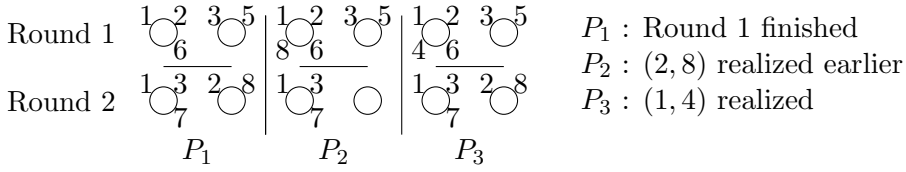


Figure 5: $\{(1,4),(2,8)\} \subseteq U$. P_2 and P_3 dominate P_1.

In this talk, we present preliminary results based on a set of crafted instances. We compare the branch and bound algorithms associated with the *person*, *couple* and *round* models and we discuss the efficiency of the dominance rules.

References

[1] Warwick Harvey. The fully social golfer problem. In *SymCon*, volume 3, pages 75–85, 2003.

[2] Agnès Le Roux. *Ordonnancement de rendez-vous en tête à tête.* PhD thesis, Ecole des Mines de Nantes, 2014.

k-letter Problem: Application, Approximation and Generalization

Anfal ALGharabally * Bala Kalyanasundaram (Speaker) †
Mahe Velauthapillai ‡

In order to explore an unknown *natural* language and find a meaningful structure, we considered the following simple combinatorial idea. Any long physical structure must have support elements to maintain its shape. The weight of the structure is distributed among the support elements. Therefore, support elements must be either strong enough to carry the weight or frequent enough to reduce the load they carry. We view a long string from a language as a long physical entity and seek for appropriate support elements. This approach has led us to find the vowels of any unknown spoken language from random valid texts of the language[1]. The underlying combinatorial problem that helped us in this process tuned out to be a generalization of the classical k-center problem on a line with unit distance between adjacent nodes. We call this problem the *k-ℓetter* problem. We remove the restriction that the underlying structure is a line and present a generalized version called the *k-provider* problem. We present combinatorial bounds on solving this problem exactly and approximately. This problem has two parameters to optimize. We show how optimization of these parameters is related to the classical *Hitting Set* problem[2]. However, our motive is not about the wizardry of establishing bounds on some combinatorial problems. It is about showing how a simple combinatorial *algorithmic* idea can explain a natural phenomenon and more.

***k-ℓetter* Problem:** Given alphabet Σ, $k \leq |\Sigma|$, and a string $\sigma \in \Sigma^*$, find a set $S \subseteq \Sigma$ such that $|S| \leq k$ and the length of the longest substring in σ without any letter in S is minimized. The decision version of *k-letter* Problem takes additional integer input η and answers yes if and only if there exists an $S \subseteq \Sigma$ such that $|S| \leq k$ and the length of the longest substring in σ without any letter in S is at most η.

Unfortunately this problem is NP-Complete and it is strongly connected to the classical *Hitting Set* problem. In addition to the size of input $|\sigma|$, there are three additional parameters, namely $|\Sigma|, k$ and η. As we will see later that the parameter $|\Sigma|$ is critical but we cannot do any approximation on it. However, the other two parameters are subject to minimization requirements. Can we find a good approximation algorithm for this problem?

We first consider the possibility of approximation algorithm for the problem where the condition on k is met while we try to minimize η.

*anfalqw8@gmail.com. Public Authority for Applied Education and Training, Kuwait University, Kuwait.

†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.

Theorem 1 *Let Σ and σ be the alphabet and the input string respectively for the* k-ℓ etter *Problem. There is no polynomial (in $|\sigma|$ and $|\Sigma|$) time poly($|\Sigma|$) approximation (on η) algorithm for the optimization version unless P=NP.*

In order to establish the proof of this theorem, we show that the gapped decision version of *k-letter* problem is NP-Complete where the established gap on η is *poly*($|\Sigma|$).

α-Gap-*k-letter* Promise Problem: Given integers $k, \eta \geq 1$, an alphabet Σ, and a string $\sigma \in \Sigma^*$, it is a *promise problem* where either there exists an $S \subseteq \Sigma$ such that $|S| \leq k$ and the length of the longest substring in σ without any letter in S is at most η (output "yes") or for every $S \subseteq \Sigma$ such that $|S| \leq k$ and the length of the longest substring in σ without any letter in S is at least $\alpha\eta$ (output "no").

Theorem 2 *For any polynomial p, p($|\Sigma|$)-Gap-k-letter promise problem is NP-Complete.*

We now consider the possibility of approximation algorithm for the problem where the condition on η is met while we try to minimize k. It turns out that our problem is exactly equal to the *Hitting Set* problem.

Finally, we consider the possibility of optimizing some combination of both k and η, say their product. Any polynomial (or less) approximation bound for our problem can be extended to the same approximation bound for the *Hitting Set* problem.

Theorem 3 *If there is an α-approximation algorithm for the k-letter problem on minimizing the product of k and η, then there is a α-approximation algorithm for the* Hitting Set *problem.*

Unfortunately approximation algorithms do not help much for our motivating problem of finding an underlying structure for natural language from a collection of texts. We need to solve the optimization problem exactly. More precisely, our algorithm for finding vowels rely on exactly solving *k-letter* problem on many instances where each input instance is sufficiently large. Fortunately our problem has a fixed parameter tractable solution. We use a simple kernalization technique to reduce the input size of the problem to a reasonably small size. This process has considerably reduced the required time to solve the problem exactly. We successfully tested our algorithm on eight different natural languages from six different families [1].

Based on the notions introduced in *k-letter* problem, we now define a generalization of the classical k-center problem that we call the k-provider problem. Let $G = (V, E)$ be a directed edge-weighted complete graph. Let C be a set of colors. Each vertex $v \in V$ is either colored with a color in C or not colored at all. Given $k, d > 0$, can we find at most k colors C' in C such that every uncolored vertex $v \in V$ is at most distance d from a vertex with color in C'.

Another motivation for this definition is as follows. Imagine a city defined by a directed graph G where each vertex is either a house (no color) or a supermarket (colored). One color in C represents one supermarket chain. Imagine that you are an inventor of a product and you want to find k supermarket chains (or providers) to carry your product. Your goal is to minimize the distance a person travels to purchase your product. It is not hard to see that the *k-letter* problem is simply a version of k-provider problem on a line where the distance between adjacent nodes is 1.

It turns out that the critical parameter for our problem is $|C| = |\Sigma|$. This parameter happens to be a small constant for our application. There is a trivial exhaustive search algorithm for the problem with running time $|C|^k.|G|$ where $|C|$ is total number of colors and $|G|$ is the size of the graph. If G is really large our kernalization method guarantee to reduce the size G to a size not larger than a graph with $|C|2^{|C|}$ nodes. Once the size of the graph is reduced, we can apply the naive algorithm to solve the problem on the reduced graph. Our kernalization method runs in linear time. Observe that the size of the graph after kernalization is exponential in $|C|$. So the reduction in input size may appear to be insignificant. This is not the case for our application. The set of colors C is nothing but the alphabet Σ of a natural language under consideration. For English $|C| = |\Sigma| = 26$. So it is manageable for the intended application.

We end this with our original motivation for exploring the structural question mentioned at the beginning. From a 2D-point of view, a protein sequence is nothing but a chain of amino acids. Any protein sequence contains 20 different types of amino acids. Given an arbitrary sequence, one can synthesize the corresponding protein in a lab. However only certain proteins occur naturally. From the point of view, there is a simple analogy between protein sequences and strings from a natural language. In any natural language, one can always come up with a new word. However, only certain combination of letters are used in practice, something very similar to naturally occurring protein sequences. Our combinatorial description seems to capture it well for natural languages. How about protein sequences or any other natural phenomenon? There are some natural languages where our algorithm will fail since some letters in the alphabet are a combination of a vowel and consonant. The algorithm will succeed if we split the letter and expose vowel and consonants individually. This situation can repeat. One must be careful in applying this idea.

References

[1] Anfal Algharabally, Bala Kalyanasundaram and Mahe Velauthapillai (2016). *Support Profile Leads To A Pattern Among Natural Languages*. 6th International Conference on Languages, Literature and Linguistics.

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

Algorithms for Hierarchical and Semi-Partitioned Parallel Scheduling

Vincenzo Bonifaci (Speaker) * Gianlorenzo D'Angelo †
Alberto Marchetti-Spaccamela ‡

1 Introduction

Multicore architectures have become the standard computing platform in many domains, and a *hierarchical* organization of clusters of multiprocessing nodes, with multicore chip-multiprocessors, is common today. In this paper, we propose a theoretical model for scheduling jobs in a multicore architecture that can capture the cost of migrations by assuming that *the processing time of a job depends on the specific set of machines on which the job is assigned*.

This setting subsumes well-known problems, such as the unrelated machine scheduling problem $R||C_{\max}$ and the (preemptive) parallel machine scheduling problem $P|pmtn|C_{\max}$, but also opens up a new class of scheduling models with their own particular challenges, such as *semi-partitioned* or *clustered* scheduling.

While the model presented here does not account exactly for the number of migrations incurred, this number can be bounded, allowing migration costs to be accounted for in the processing times, if desired. This allows for a flexible input representation and can accomodate heterogeneous processors.

2 The model

We are given a set of n jobs $J := \{1, \ldots, n\}$ and a set of m machines $M := \{1, \ldots, m\}$. Each job needs to be assigned to a *set* of machines on which the job is allowed to schedule and it can be preempted and migrated among any such machines. However, its processing time depends on the set of machines on which it is assigned. In detail, we are given a family of admissible sets $\mathcal{A} \subseteq 2^M$, and for each job $j \in J$, a processing time function $P_j : \mathcal{A} \to \mathbb{Z}_+$ with the constraint that the function must be *monotone* on $\mathcal{A}$, i.e., if $\alpha, \beta \in \mathcal{A}$ and $\alpha \subseteq \beta$, then $P_j(\alpha) \leq P_j(\beta)$, modeling the fact that processing overheads (caused, e.g., by migration) increase if the job is executed using a larger set of machines. We define $p_{\alpha j} := P_j(\alpha)$. When α is a singleton, such as $\alpha = \{i\}$, we also write p_{ij} instead of $p_{\{i\}j}$.

The interpretation is that, when the job is run on set $\alpha \subseteq \mathcal{A}$, then it can be only migrated among the machines in α, and the processing time it receives must be $P_j(\alpha)$. In

*vincenzo.bonifaci@iasi.cnr.it. Consiglio Nazionale delle Ricerche, Rome, Italy.
†gianlorenzo.dangelo@gssi.infn.it. Gran Sasso Science Institute, L'Aquila, Italy.
‡alberto@dis.uniroma1.it. Sapienza University of Rome, Italy.

general, if a job is run on machine set M' (which may or may not be in $\mathcal{A}$), its processing time is $p_{\alpha j}$, where α is the inclusion-wise minimal set in $\mathcal{A}$ that contains M' (if there is no such α, then j cannot be run on M').

Given J and $\mathcal{A}$, an *assignment* of jobs in J to sets in $\mathcal{A}$ is a function that assigns each job in J to a set in $\mathcal{A}$. If a job j is assigned to a set α, then its processing time is $P_j(\alpha)$. The set α to which a job j is assigned is also called the *affinity mask* of j. Given an assignment of jobs in J to sets in $\mathcal{A}$, a schedule is *valid* with respect to the assignment if each job is scheduled on time slots of machines in its affinity mask, no job is processed in parallel on more than one machine in the same time interval (though it may be preempted or migrated), each job j receives the required amount of processing time (i.e. $p_{\alpha j}$, if j is assigned to set α), and no machine processes more than one job in the same time interval. We assume that a schedule starts at time 0 and allow preemptions and migrations to occurr only at integer time points. If, in a given schedule, a job j completes at time C_j, then $T := \max_{j \in J} C_j$ is called the *makespan* of the schedule.

In this paper, we consider the problem of finding an assignment of jobs in J to sets in $\mathcal{A}$ and a corresponding valid schedule that minimizes the makespan. We restrict the discussion to *laminar* (or *hierarchical*) instances of the problem, where, for each $\alpha, \alpha' \in \mathcal{A}$, either $\alpha \subseteq \alpha'$ or $\alpha' \subseteq \alpha$ or $\alpha \cap \alpha' = \emptyset$. Without loss of generality, we assume that all sets in the family $\mathcal{A}$ are distinct. In a laminar instance, the *level* of a set β is the number of sets $\alpha \in \mathcal{A}$ such that $\beta \subseteq \alpha$ and the level of the instance is the maximum level among the sets in $\mathcal{A}$. We call the problem with laminar instances the *hierarchical scheduling problem*. The hierarchical scheduling problem generalizes some well-known and new scheduling problems.

- *Identical parallel machines* scheduling with preemption ($P|pmtn|C_{\max}$) [4]: take $\mathcal{A} = \{M\}$. Then each job j can be migrated freely among the machines in M, as long as it receives the processing time p_{Mj}.
- *Unrelated parallel machines* scheduling ($R||C_{\max}$) [3]: take $\mathcal{A}$ to be a family of m singletons, i.e., $\mathcal{A} = \{\{1\}, \{2\}, \ldots, \{m\}\}$. Then each job must be assigned to a single machine (no migration) and its processing time is a function of the machine.
- *Semi-partitioned* scheduling: take $\mathcal{A} = \{M, \{1\}, \{2\}, \ldots, \{m\}\}$. Then each job can either be run *globally* (i.e., freely migrated) on M with processing time p_{Mj}, or assigned *locally* to a specific machine $i \in M$, with processing time $p_{ij} \leq p_{Mj}$.
- *Clustered* scheduling [1]: let $m = kq$. Take $\mathcal{A} = \{M, \{1\}, \ldots, \{m\}, \{1, \ldots, q\}, \{q+1, \ldots, 2q\}, \ldots, \{(k-1)q, \ldots, kq\}\}$. Then each job can be run globally, or locally to a single machine, or locally to a *cluster* of q machines.

Semi-partitioned scheduling generalizes scheduling on unrelated parallel machines; hence, the following proposition is implied by existing results.

Proposition 1. *Hierarchical and semi-partitioned scheduling are* NP*-hard to approximate within any constant factor less than* $3/2$.

Our main result is the following:

Theorem 1. *The hierarchical scheduling problem admits a polynomial-time* 2*-approximation algorithm.*

3 Outline of the approach

Our approach is to divide the problem into two subproblems: given J and $\mathcal{A}$, find an assignment of jobs in J to sets in $\mathcal{A}$ that admits a valid schedule in the interval $[0, T]$ and minimizes T; and given an assignment of jobs in J to sets in $\mathcal{A}$ that admits some valid schedule in the interval $[0, T]$, construct a valid schedule in the same interval.

We show that the first subproblem is captured by the following integer linear program.

$$\min T \tag{IP}$$

$$\sum_{\alpha \in \mathcal{A}} x_{\alpha j} = 1 \qquad \text{for } j \in J \tag{1a}$$

$$\sum_{j=1}^{n} \sum_{\beta \subseteq \alpha} p_{\beta j} x_{\beta j} \leq |\alpha| T \qquad \text{for each } \alpha \in \mathcal{A} \tag{1b}$$

$$p_{\alpha j} x_{\alpha j} \leq T \qquad \text{for each } \alpha \in \mathcal{A},\ j \in J \tag{1c}$$

$$x_{\alpha j} \in \{0, 1\} \qquad \text{for each } \alpha \in \mathcal{A},\ j \in J \tag{1d}$$

We round the ILP by "pushing down" the fractional weights towards the singleton sets of the laminar family and then invoking a standard rounding procedure. The resulting assignment $\mathbf{x}$ satisfies the ILP constraints (1a)-(1d) with $T \leq 2T^*$, where T^* is the optimal makespan.

For the second subproblem, we give an algorithm that takes as input a feasible solution $(\mathbf{x}, T)$ to (IP) and constructs a valid schedule with makespan T. The algorithm works in two phases: a bottom-up load computation phase, followed by a top-down time-slot scheduling phase. This subproblem is solved optimally.

Theorem 2. *There is a polynomial time algorithm that, given a feasible solution* $(\mathbf{x}, T)$ *to* (IP)*, constructs a valid schedule in the interval* $[0, T]$*.*

For a complete version of this work we refer to [2].

References

[1] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers. In *Proceedings of the 31st IEEE Real-Time Systems Symposium*, pages 14–24. IEEE, 2010.

[2] V. Bonifaci and G. D'Angelo and A. Marchetti-Spaccamela. Algorithms for hierarchical and semi-partitioned scheduling. In *Proceedings of the 31st IEEE Int. Parallel and Distributed Processing Symposium*, IEEE, accepted for publication.

[3] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.

[4] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[5] G. Muratore, U. M. Schwarz, and G. J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Operations Research Letters*, 38(1):47 – 50, 2010.

Solving the Airline Manpower Planning Problem

Fredrik Altenstedt (Speaker) * Björn Thalén † Per Sjögren ‡
Staffan Nilsson §

1 Introduction

The objective of the airline manpower planning problem is to have the right number of crew with the right qualifications at the right time. The problem consists of making projections of the demand for crew as well as the future productivity of crew and then suggest decisions that will close the projected gap between supply and demand as cheaply as possible. Here we assume that the projections are given and accurate and we are only interested in the resulting optimization problem.

In order to track how much crew we have at our disposal for each qualification group we divide the crew into crew-groups. A crew group is typically defined by the triplet base-rank-equipment. An example would be Boeing 737 first officers based in Seattle. The tools at our disposal to close the gaps can be divided into major (permanent) and minor (transient) decisions. The major decisions are the award of transition training that will permanently move crew from one crew-group to another. We consider hiring new pilots as a special form of transition. The minor decisions are temporary reallocations of supply between groups and over time, such as when to place mandatory recurrent training, the use of overtime, and the movement of demand between crew-bases.

The airline manpower planning problem has been previously studied in the literature, Yu, Dugan, and Agrüello have written one of the first papers on the subject [2] with other contributions including [1] and [3]

2 Modeling

2.1 Supply and Demand

In order to track the supply and demand for each crew-group over time we discretize the whole planning horizon into time-periods. We refer to the time-period/crew-group combination as a bucket and add constraints to make sure that supply exceeds demand for each bucket.

As mentioned above, crew can be moved between crew groups by assigning them to transition training, which will permanently change which group they belong to. Training

*fredrik.altensted@jeppesen.com. Jeppesen Systems AB, Odinsgatan 9, 411 03 Gothenburg, Sweden.
†bjorn.thalen@jeppesen.com. Jeppesen Systems AB,
‡per.sjogren@jeppesen.com. Jeppesen Systems AB,
§staffan.nilsson@jeppesen.com. Jeppesen Systems AB,

consumes resources such as flight simulators and flight instructors. As the instructors also function as normal crew when not instructing, holding courses will reduce the available crew by more than just removing the students from active duty. Just as for the crew-groups we add buckets for the training resources to make sure that demand does not exceed supply.

In addition to the more permanent transition decisions we have the small transient decisions such as how to when place recurrent training or when to use overtime. We model the transient decisions using helper variables that add/remove time to buckets subject to limits on how much of the transient activity is allowed.

2.2 Seniority

The main complicating factor when solving the airline manpower planning problem is the seniority rules. As receiving a promotion may greatly influence compensation, training has do be done in seniority order while respecting the wishes of the crew. This means that a senior pilot retiring will cause many trainings as a number of pilots moves one notch up the career-ladder. The most junior pilot in the training-chain will then be replaced by a new recruit if needed. The seniority rules are further complicated by lock-in rules. In order to prevent excessive amounts of transitions, a crew is usually forced to stay in their current position for 1-5 years following a transition. Since the length of a transition training depends on the current qualification of the student, lock in might prevent a crew from getting an expensive assignment while at the same time being eligible for a cheap assignment. When modeling seniority we assume that each crew will assign a preference number to each possible training assignment. Staying in the current position is considered an assignment like any other and is given it's own preference. In addition each crew will have a unique seniority number with respect to each assignment. We denote the preference and seniority of a crew c for a training slot t by $p(c,t)$ and $s(c,t)$ respectively. Lower preference number means better and lower seniority number means more senior. Using these two functions we can model a great variation of seniority rules. Abstracting the seniority rules in this way decopules the rules of a specific airline from the solver, the drawback is that we can not accurately express seniority rules involving more than two crew.

Seniority is violated if crew A prefers the assignment of crew B to his own and crew A has better seniority than crew B for that assignment. If we assume we have binary variables $x_{c,t}$ determining if a crew c gets an assignment t, the seniority rules protecting $\bar{c}$ from $\hat{c}$ with respect to the assignment $\hat{t}$ can be expressed as

$$\sum_{\bar{t}:p(\bar{c},\bar{t})\leq p(\bar{c},\hat{t})} x_{\bar{c},\bar{t}} \geq x_{\hat{c},\hat{t}}$$

Note that assignment $\hat{t}$ need not be legal for crew $\bar{c}$ for the constraint to apply. By using the assumption that each crew gets exactly one assignment, these constraints can be strengthened and redundant constraints removed, see [4] for further details.

3 Solution approach

Due to the lock-in rules preventing multiple transitions it is possible to enumerate all possible legal training assignments for all crew. For each of these assignments it is straight forward to compute how the crew will contribute to different buckets.

Given the set of all training assignments we can formulate the problem as a MIP problem, although that problem will be too large to actually solve (when experimenting with a medium sized European carrier CPLEX was unable to find even an initial solution within a reasonable time-frame). Such a formulation is however still very useful as a reference for testing our heuristic solution approach. Instead we solve the problem by using an initial heuristic followed by repeated application of very large-scale neighborhood search for different types of neighborhoods.

The main neighborhood used is the seniority neighborhood. For a given seniority feasible solution and a potential new training assignment t for crew c, we count how many other crew must change assignment if c is to legally get the assignment t. We then use this number to determine if the assignment is included in the neighborhood or not. Other neighborhoods are based on time-periods as well as qualification groups.

The initial solution algorithm works by taking a rather large N-seniority neighborhood and relaxing the integrality. The fractional solution given is used in a rounding heuristic producing a given number of moves and these are then legally assigned to crew by solving another restricted problem.

4 Results

During the presentation we will show the results from two different airlines.

References

[1] Gang Yu et al. (2004). *Optimizing pilot planning and training for Continental Airlines*. Interfaces 34.4 (2004), pp. 253-264.

[2] Gang Yu, Stacy Dugan, and Mike Argello (1998). *Moving toward an integrated decision support system for manpower planning at Continental Airlines: Optimization of pilot training assignments* Industrial Applications of Combinatorial Optimization. Springer, 1998, pp. 124.

[3] Milind G Sohoni, Ellis L Johnson, and T Glenn Bailey (2004). *Long-range reserve crew manpower planning* Management Science 50.6 (2004), pp. 724739.

[4] B. Moren (2012). *Using problem specific structures in branch and bound methods for manpower planning.*. Masters Thesis, University of Linköping.

Decomposition Algorithms for Synchronous Flow Shop Problems With Additional Resources and Setup Times

Sigrid Knust (Speaker) * Stefan Waldherr †

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. [1, 2]). 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 where jobs are 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 completion time C_j of a job j is equal to the time when it has been processed on all machines and leaves the system. The goal is to find a permutation π of the jobs such that the makespan $C_{\max} = \max_j C_j$ is minimized. With each permutation a corresponding (left-shifted) schedule is associated in which each operation starts as early as possible.

In this work, we further consider renewable job resources $\mathcal{R}$ and assume that every job j needs a single assigned resource from a subset $\mathcal{R}(j) \subseteq \mathcal{R}$ during its whole processing, from its start on M_1 until its removal from M_m. After a job is completed on M_m, the corresponding resource may immediately be used by the next job starting on M_1 (i.e., the job occurring m positions later in the job permutation). If the resource is not feasible for the next job, it must be changed, requiring a certain setup time.

Fig. 1 shows an exemplary configuration of a circular production unit with $m = 4$ machines where the required resources are specialized work piece carriers which are equipped to transport jobs on the production unit. The jobs enter the unit at machine M_1. After each cycle, the unit rotates clockwise and all jobs are transported to the next machine with the help of the work piece carriers. At M_4, the current job is removed

*sknust@uni-osnabrueck.de. Institute of Computer Science, University of Osnabrück, Germany.
supported by the Deutsche Forschungsgemeinschaft, KN 512/7-1.
†stefan.waldherr@in.tum.de. Department of Informatics, TU Munich, Germany.

from the production unit and at the end of the cycle the work piece carrier moves back to M_1. At this point, a setup is necessary if the next job to be inserted on M_1 requires a different work piece carrier. The schedule depicts the possible setups that occur between jobs $m = 4$ positions apart in the permutation π. The makespan is equal to the sum of all cycle times plus the sum of all setup times.

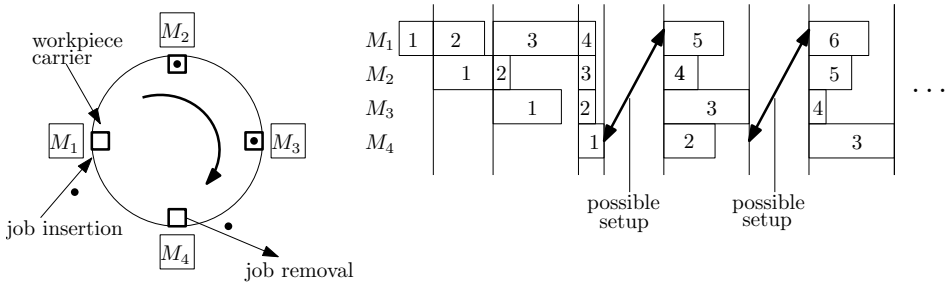


Figure 1: A circular production unit with $m = 4$ machines

If additionally resources have to be considered, a feasible schedule may be represented by a job permutation $\pi = (\pi_1, \ldots, \pi_n)$ and a corresponding resource sequence $\varrho = (\varrho_1, \ldots, \varrho_n)$ with $\varrho_i \in \mathcal{R}(\pi_i)$ for $i = 1, \ldots, n$ where no resource $r \in \mathcal{R}$ appears more than once in any m consecutive positions of ϱ.

For the resource sets $\mathcal{R}(j)$ we distinguish the following situations:

(R1) All jobs can be processed by all resources (i.e., $\mathcal{R}(j) = \mathcal{R}$ for all jobs $j \in \mathcal{J}$).

(R2) The jobs are partitioned into disjoint families $\mathcal{F}$, where each job in a family can be processed by the same set of resources (i.e., if $\mathcal{R}(j) \cap \mathcal{R}(h) \neq \emptyset$, then $\mathcal{R}(j) = \mathcal{R}(h)$). This situation is motivated by practical applications where similar jobs can all be processed by the same resources.

(R3) The sets $\mathcal{R}(j)$ are arbitrary subsets of $\mathcal{R}$.

In situation (R1), no setups are necessary since each job can use the same resource which was used by its predecessor in the subsequence. In contrast, in situations (R2) and (R3), we also have to take into account that a feasible resource sequence ϱ must satisfy $\varrho_i \in \mathcal{R}(\pi_i)$ for $i = 1, \ldots, n$.

2 Decomposition algorithms

Minimizing the makespan in a synchronous flow shop with additional setup times is strongly $\mathcal{NP}$-hard even in the case of two machines, unlimited resources of type (R2) and equal setup times (cf. [3]). In the following we propose two decomposition strategies for the practically relevant situation (R2) (cf. also [3]). Both decomposition approaches can be used to calculate lower bounds as well as heuristic solutions. To get lower bounds, we consider the two objective functions of minimizing the sum of cycle times and the sum of setup times separately.

Decomposition D1:

1. Determine a permutation $\pi = (\pi_1, \ldots, \pi_n)$ of all jobs.

2. Assign a feasible resource $\varrho_i \in \mathcal{R}(\pi_i)$ to each job π_i for $i = 1, \ldots, n$ such that no resource appears more than once in m consecutive positions.

Note that the sum of cycle times is completely determined by the permutation π obtained in Step 1. However, it may happen that for a permutation π no corresponding feasible resource assignment exists. Within this approach, we first determine a job permutation with a small sum of cycle times heuristically and afterwards find a corresponding optimal resource assignment minimizing the sum of all setup times (which can efficiently be done in $\mathcal{O}(n)$). Then a tabu search is applied using a swap neighborhood, where two jobs in π are exchanged and the resources for these two jobs are reassigned after the swap.

Decomposition D2:

1. Determine a sequence $\varrho = (\varrho_1, \ldots, \varrho_n)$ of resources such that no resource appears more than once in m consecutive positions.

2. Assign to each resource in the sequence a corresponding job which may be processed by this resource.

Note that the sum of setup times is completely determined by the sequence ϱ obtained in Step 1. For resources of type (R2) and equal setup times, finding a feasible resource sequence ϱ minimizing the sum of setup times can be calculated in time polynomial in n if the number of machines is fixed. It is strongly $\mathcal{NP}$-hard if the number of machines is part of the input. In Step 2, for a given resource sequence ϱ we try to find a corresponding optimal job permutation minimizing the sum of all cycle times. Since this problem is strongly $\mathcal{NP}$-hard (cf. [3]), we solve it heuristically. Then, again a tabu search algorithm is applied where at first the resource sequence is changed and then a new corresponding permutation π is determined.

The two approaches were tested on two sets of instances (modified benchmark instances of Taillard and real-world data) and computational results are presented.

References

[1] K.-L. Huang (2008). *Flow shop scheduling with synchronous and asynchronous transportation times*. Ph.D. Thesis, The Pennsylvania State University.

[2] 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.

[3] S. Waldherr and S. Knust (2017). Decomposition algorithms for synchronous flow shop problems with additional resources and setup times, *European Journal of Operational Research* 259, 847–863.

An Integer Linear Programming Approach for Handling New Real-World Motivated Constraints of the Curriculum-Based Course Timetabling Problem*

Philipp Hungerländer † Kerstin Maier (Speaker) ‡

1 Introduction

The task of the curriculum-based course timetabling problem (CB-CTT) is to schedule lectures belonging to a set of courses $C = \{c_1, c_2, \ldots, c_v\}$ to k periods $P = \{p_1, p_2, \ldots, p_k\}$ and m rooms $R = \{r_1, r_2, \ldots, r_m\}$, accounting at the same time for certain hard and soft constraints. In the CB-CTT the timetable is generated based upon a set of s university curricula $I = \{i_1, i_2, \ldots, i_s\}$, to which the courses belong. The goal of the International Timetabling Competitions ITC2002 and ITC2007 was to establish models for comparison that cover the most frequently found use cases.

Our model, motivated by a project with University College London (UCL), builds on the standard model from track 3 of ITC2007 [3]. The UCL timetabling problem presents a wide range of challenges, since its features and additional constraints substantially exceed the ones from the ITC-2007 framework. We mention here only the most interesting extensions from an academic viewpoint:

1. Our courses consist of activities with different durations, which relaxes the indistinguishability assumption of lectures from the literature. Therefore we define the set of all activities $A = \{a_1, a_2, \ldots, a_n\}$ with corresponding durations of activities $d_a,\ a \in A$.

2. The UCL framework aims to generate feasible timetables for a set W of 10 consecutive weeks in a manner that guarantees the highest possible timetable regularity. This means that whenever possible, activities should be scheduled in the same period and room over the different weeks. We introduce the corresponding timetable regularity metric, which measures the consistency of time and room assignments for a course throughout the term.

3. The activities have a specific predefined type, which must match the room type of the assigned room.

In the following section we suggest an Integer Linear Programming (ILP) approach for solving this expanded problem, conduct computational experiments and discuss the results obtained with respect to solution quality and practical suitability for UCL.

*An extended abstract [1] on this research topic has been accepted for publication in the Operations Research Proceedings 2016.

†philipp.hungerlaender@aau.at. Laboratory for Information & Decision Systems, MIT, USA.

‡kerstin.maier@aau.at. Department of Mathematics, Alpen-Adria Universität Klagenfurt, Austria.

2 The Integer Linear Programming Formulation

Our ILP solver is based on the ILP approach suggested by Lach and Lübbecke [2]. The problem is split into two stages. In the first stage, each activity is assigned to an appropriate set of consecutive time periods. The assignment of activities to rooms is done in the second stage. Due to space limitation, we state the mathematical formulation only for the second stage and for the first stage we solely mention the most important newly developed constraints.

First Stage: In the first stage each activity has to be scheduled in a consecutive set of time periods. The function $D(p)$ gives the day of period p. Now if activity a is scheduled at period p, then the binary variable x_{ap} is set to 1. Otherwise we have $x_{ap} = 0$. To ensure that an activity is assigned to consecutive time periods, we also need binary variables s_{ap}, which are set to 1, if activity a starts at period p. Otherwise we have $s_{ap} = 0$. Note that variable s_{ap} is only introduced, if there are at least $d_a - 1$ consecutive time periods available after period p on the same day:

$$x_{ap} - \sum_{\substack{t=p-d_a+1 \\ s_{at} \text{ exists}}}^{p} s_{at} = 0, \quad a \in A,\ p \in P, \qquad \sum_{s_{ap} \text{ exists}} s_{ap} = 1, \quad a \in A.$$

The first set of equalities ensures that each activity is assigned to a set of consecutive time periods that all belong to the same day. The second set of equalities guarantees that each activity has exactly one start time period.

One of our main goals for UCL is to minimize the total number of rooms required. While this goal is clearly part of the objective function of the Second Stage, we also need to consider it during the First Stage. We propose the following constraints in order to restrict the total number of activities scheduled per time period: $\sum_{a \in A} x_{ap} \leq M,\ p \in P$, where M is an integer variable that is multiplied by a penalty term p_M in the objective function of the First Stage. Without these inequalities arbitrarily many activities could be assigned to the same time period during the First Stage, which could leave us with no possibility to minimize the number of rooms required in the Second Stage.

Second Stage: After solving the First Stage, in the Second Stage we determine feasible rooms for the activities, where we aim to minimize the following objectives:

1. The number of students, which have no seat during an activity.
2. The number of empty seats in a room during an activity.
3. The total number of rooms.

In order to build an ILP model for the Second Stage, we introduce binary variables u_r, y_{ar} and z_{arp} with the following interpretations:

- $u_r = 1$, if at least one activity is scheduled in room r. Otherwise $u_r = 0$.
- $y_{ar} = 1$, if activity a is scheduled in room r. Otherwise $y_{ar} = 0$.
- $z_{arp} = 1$, if activity a is scheduled in room r at period p. Otherwise $z_{arp} = 0$.

Note that the variables y_{ar} and z_{arp} are only introduced, if it is feasible to schedule activity a in room r at period p, i.e. if the activity type matches with the room type, if the activity is assigned to period p in the First Stage and if the room is available at period p. Accordingly we define $P(a)$, $P(r)$ and $P(a, r)$ as the sets of available time periods for activity a, for room r and for their combination respectively. Analogously we specify $A(r)$ and $A(p, r)$ as the sets of feasible activities for room r at period p and $R(a, p)$ as the set of feasible rooms for activity a at period p.

For each feasible activity-room combination we introduce a penalty parameter p_{ar} that gives the absolute value of the difference between the available seats in room r and the number of students registered for activity a. We also introduce the penalty parameter p_r giving the costs for using room r. Now we can state our ILP model:

$$\min \sum_{a\in A,\ r\in R(a,p)} p_{ar}y_{ar} + \sum_{r\in R} p_r u_r \tag{1a}$$

$$\text{s.t.} \sum_{r\in R(a)} z_{arp} = 1, \quad a \in A,\ p \in P(a), \tag{1b}$$

$$d_a y_{ar} - \sum_{p\in P(a,r)} z_{arp} = 0, \quad r \in R,\ a \in A(r), \tag{1c}$$

$$\sum_{a\in A(p,r)} z_{arp} - u_r \leq 0, \quad r \in R,\ p \in P(r), \tag{1d}$$

$$u_r \in \{0,1\},\ y_{ar} \in \{0,1\},\ z_{arp} \in \{0,1\}, \quad r \in R,\ a \in A(r),\ p \in P(a,r). \tag{1e}$$

Equalities (1b) guarantee that exactly one room is assigned to an activity at each time period. Equalities (1c) ensure that the same room is assigned to all time periods of an activity. Constraints (1d) guarantee that at most one activity is assigned to room r at each time period and also ensure $u_r = 1$, if at least one activity is scheduled in r.

Computational Experiments: Finally we present the results obtained by using our ILP approach on a selection of the original set of UCL curricula, available at `http://tinyurl.com/timetabling-lib`. All experiments were performed on a Linux 64-bit machine equipped with 4 × Intel(R) Xeon(R) CPU e5-2630 v3@2.40GHz and 16 GB RAM. We use Gurobi 6.5.1 as our ILP-solver.

Our benchmark set consists of around 250 activities per week with an average length of ≈ 3.5 time periods. Each week consists of 5 days with 18 time periods (a 30 minutes) per day. In each week we use around 20 of the available 279 rooms. Within 190 seconds computing time we obtained timetables for the entire term with a very high timetable regularity. Furthermore in our timetables used rooms are more than $\frac{3}{4}$ full and rooms, which are used at least once, are occupied almost 50% of the total available time periods.

References

[1] Applebee, Sam and Aschinger, Markus and Bucur, Alexandru P. and Edmonds, Harry and Hungerländer, Philipp and Maier, Kerstin (2016). *New Constraints and Features for the University Course Timetabling Problem. Operations Research Proceedings 2016, accepted.*

[2] Lach, Gerald and Lübbecke, Marco E. (2012). *Curriculum based course timetabling: new solutions to Udine benchmark instances. Annals of Operations Research, 194(1):255-272.*

[3] McCollum, Barry and Schaerf, Andrea and Paechte, Ben and McMullan, Paul and Lewis, Rhyd and Parkes, Andrew J. and Di Gaspero, Luca and Qu, Rong and Burke, Edmund K. (2007). *Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS Journal on Computing, 22(1):120-130.*

Scheduling Meets n-fold Integer Programming

Dušan Knop (Speaker) * Martin Koutecký †

Introduction. Scheduling problems are fundamental in combinatorial optimization. Much work has been done on approximation algorithms for NP-hard cases [9], but relatively little is known about exact solutions when some part of the input is a fixed parameter. In 2014, Mnich and Wiese [6] initiated a systematic study in this direction.

However, our goal is not merely to prove new positive results. In their work, Mnich and Wiese rely on mathematical programming techniques in *fixed* dimension, which have been introduced in 1983 by Lenstra [5] and significantly extended in 2000 by Khachiyan and Porkolab [3]. These techniques are by now well established in the FPT community, even though the power of the latter extension due to Khachiyan and Porkolab has not been fully utilized yet. Independently of this, a new theory of *variable* dimension optimization has been developed in the past 15 years; see Onn's book [7]. A breakthrough result is an FPT algorithm for the so-called n-fold integer programming (n-fold IP) by Hemmecke, Onn and Romanchuk [1]. In contrast to the fixed dimension techniques, n-fold IP is not yet established as an indispensable part of an FPT researchers toolbox. This is what we would like to change.

The new tool – n-fold integer programming. Given nt-dimensional integer vectors $\mathbf{b}, \mathbf{u}, \mathbf{l}, \mathbf{w}$, n-fold integer programming (n-fold IP) is the following problem in variable dimension nt:

$$\min \left\{ \mathbf{wx} : A^{(n)}\mathbf{x} = \mathbf{b}\,,\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\,,\ \mathbf{x} \in \mathbb{Z}^{nt} \right\}, \text{ where} \tag{1}$$

$$A^{(n)} := \begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}$$

is an $(r + ns) \times nt$ matrix with A_1 an $r \times t$ matrix and A_2 an $s \times t$ matrix. Let a be the biggest number in $A^{(n)}$. The vector $\mathbf{x}$ is naturally partitioned into n *bricks* of size t, that is, we index it as $\mathbf{x} = (x^1_1, x^1_2, \ldots, x^1_t, \ldots, x^n_1, \ldots, x^n_t)$. As such, n-fold IP is best suited for *multi-index* problems whose IP formulation has variables indexed by $[n] \times [l_1] \times \cdots \times [l_k]$ for some integers $n, l_1, \ldots, l_k$ such that $l_1, \ldots, l_k$ are fixed parameters and only n is variable.

*knop@kam.mff.cuni.cz. Department of Applied Mathematics, Charles University, Prague, Czech Republic

†koutecky@kam.mff.cuni.cz. Department of Applied Mathematics, Charles University, Prague, Czech Republic

In their seminal paper Hemmecke, Onn and Romanchuk [1] prove that there is an FPT algorithm solving problem (1) with parameters r, s, t and a.

Our contribution. We consider three non-preemptive scheduling models of increasing generality: parallel identical, uniformly related and unrelated machines (in the standard notation [4] denoted by P, Q and R, respectively), and the two most common objective functions: minimizing makespan and sum of weighted completion times (denoted $C_{\max}$ and $\sum w_i C_i$, respectively).

Formally, for *identical* machines, the problem consists of a set of n *jobs* $J = \{J_1, \ldots, J_n\}$ and m *machines* $M = \{M_1, \ldots, M_m\}$, and each job J_i has a *processing time* $p_i \in \mathbb{N}$. For *uniformly related* machines, we additionally have for each machine M_j its *speed* $s_j \in \mathbb{N}$, such that processing job J_i on machine M_j takes time p_i/s_j. For *unrelated* machines, we have for each job J_i an m-dimensional vector $\mathbf{p} = (p_i^1, \ldots, p_i^m)$, $p_i^j \in \mathbb{N} \cup \{\infty\}$ for all j, such that processing job J_i on machine M_j takes time p_i^j (in case $p_i^j = \infty$, J_i cannot be executed on M_j). We also consider a restricted variant of the unrelated machines model where there are K *kinds* of machines and the vector of processing times for a job J_i is given with respect to kinds of machines: $\mathbf{p} = (p_i^1, \ldots, p_i^K)$, such that processing J_i on machine M_j of kind k takes time p_i^k. Additionally, for the sum of weighted completion times objective, we are given for each job J_i its *weight* $w_i \in \mathbb{N}$.

The parameters we consider are the following:

- $p_{\max}$: the maximum processing time of any job,
- $w_{\max}$: the maximum weight of any job,
- m: the number of machines,
- θ: the number of distinct job processing times and weights (in case of the $\sum w_i C_i$ objective)
- K: the number of kinds of machines (defined above).

Our Results.

Theorem 1. *The following scheduling problems are* FPT *with respect to parameter* Θ *as defined below and solvable in time* $\Theta^{\mathcal{O}(\Theta^2)} n^{\mathcal{O}(1)}$*, where*

1. $Q||C_{\max}$*:* $\Theta = p_{\max}$
2. $R||C_{\max}$*:* $\Theta = p_{\max}^K$
3. $R||\sum w_j C_j$*:* $\Theta = (\max\{p_{\max}, w_{\max}\})^K$

Conclusions Although much is known about approximating the scheduling problem, little is known from the parameterized complexity point of view about the most basic problems. The purpose of this paper is twofold. The first is to show new FPT algorithms for some scheduling problems. The second is to demonstrate the use of n-fold integer programming, a recent and powerful variable dimension technique. We hope to encourage research in both directions. To facilitate this research, we point out the following open problems:

- Minimizing weighted flow time $P|r_j|\sum w_jF_j$ parameterized by $p_{\max}+w_{\max}$.
- $P||C_{\max}$ parameterized by the number of different processing times instead of $p_{\max}$.
- $R|pmtn|\sum C_j$ parameterized by m and $p_{\max}$; this is justified by the problem being strongly NP-hard [10], so parameterizing by $p_{\max}$ is not enough.
- $P||C_{\max}$ parameterized by $p_{\max}$ with both m and n given in binary; this might be possible using techniques related to the recently developed result for huge n-fold IPs of Onn and Sarrabezolles [8].
- Multi-agent scheduling was studied by Hermelin et al. [2]; what results can be obtained by applying n-fold IP?
- We are also interested in further applications of n-fold IP and quasiconvex minimization over convex sets in fixed dimension.

References

[1] R. Hemmecke, S. Onn, and L. Romanchuk. n-fold integer programming in cubic time. *Math. Program*, 137(1-2):325–341, 2013.

[2] Hermelin, D., Kubitza, J., Shabtay, D., Talmon, N., Woeginger, G.J.: Scheduling two competing agents when one agent has significantly fewer jobs. In: IPEC 2015, pp. 55–65, 2015.

[3] L. Khachiyan and L. Porkolab. Integer Optimization on Convex Semialgebraic Sets. *Discrete & Computational Geometry*, 23(2):207224, 2000.

[4] E. L. Lawler, J. K. Lenstra, A. H.G. R. Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522, 1993.

[5] H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[6] M. Mnich and A. Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1):533–562, 2014.

[7] S. Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010.

[8] Onn, S., Sarrabezolles, P.: Huge unimodular n-fold programs. SIAM Journal on Discrete Mathematics **29**(4), 2277–2283, 2015.

[9] C. N. Potts and Vitaly A. Strusevich. Fifty years of scheduling: a survey of milestones. *JORS*, 60(S1), 2009.

[10] Sitters, R.: Complexity of preemptive minsum scheduling on unrelated parallel machines. J. Algorithms **57**(1), 37–48, 2005.

Online Packet Scheduling with Bounded Delay and Lookahead

Martin Böhm* Marek Chrobak† Łukasz Jeż‡ Fei Li§ Jiří Sgall*
Pavel Veselý (Speaker)*

Introduction. In the *online bounded-delay packet scheduling problem* (PacketScheduling), packets of unit size arrive at a router over time and need to be transmitted over a network link. Each packet has two attributes: a non-negative weight and a deadline for its transmission. The time is assumed to be discrete (slotted), and only one packet can be sent in each slot. The objective is to maximize the total weight of the transmitted packets.

This problem was introduced by Kesselman *et al.* [10] as a theoretical abstraction that captures the constraints and objectives of packet scheduling in networks that need to provide quality of service (QoS) guarantees. The combination of deadlines and weights is used to model packet priorities. In the literature, the PacketScheduling problem is sometimes referred to as *bounded-delay buffer management in QoS switches*. It can also be formulated as the job-scheduling problem $1|p_j = 1, r_j|\sum w_j U_j$, where packets are represented by unit-length jobs with deadlines, with the objective to maximize the weighted throughput. We focus on the deterministic online setting only.

A simple online greedy algorithm that always schedules the heaviest pending packet is known to be 2-competitive [9, 10]. This ratio was first improved by Chrobak *et al.* [5], and the best currently known ratio is $2\sqrt{2} - 1 \approx 1.828$ [6]. The best lower bound, widely believed to be the optimal ratio, is $\phi = \frac{1}{2}(1 + \sqrt{5}) \approx 1.618$ [9, 2, 4]. Closing the gap between these two bounds is one of the most intriguing open problems in online scheduling.

***s*-Bounded instances.** In an attempt to bridge this gap, restricted models have been studied. In the *s-bounded* variant of PacketScheduling, each packet must be scheduled within k consecutive slots, starting at its release time, for some $k \leq s$ possibly depending on the packet. In other words, we assume that each packet p satisfies $d_p \leq r_p + s - 1$ where r_p is the release time and d_p is the deadline which is the last slot where p can be scheduled.

The lower bound of ϕ from [9, 2, 4] holds even in the 2-bounded case. A matching ϕ-competitive algorithm was given Kesselman *et al.* [10] for 2-bounded instances and by Chin *et al.* [3] for 3-bounded instances. Both results are based on the algorithm EDF_α, with $\alpha = \phi$, which always schedules the earliest-deadline packet whose weight is

*{bohm,sgall,vesely}@iuuk.mff.cuni.cz. Computer Science Institute of Charles University, Prague, Czech Republic. Supported by the project 17-09142S of GA ČR and by the GAUK project 634217.

†marek@cs.ucr.edu. Department of Computer Science and Engineering, University of California, Riverside, USA.

‡lje@cs.uni.wroc.pl. Institute of Computer Science, University of Wrocław, Poland.

§lifei@cs.gmu.edu. Department of Computer Science, George Mason University, USA.

at least the weight of the heaviest pending packet divided by α (ties are broken in favor of heavier packets). EDF_ϕ is not ϕ-competitive for 4-bounded instances; however, for $\alpha = \sqrt{3} \approx 1.732$ the algorithm is $\sqrt{3}$-competitive for the 4-bounded case [3].

We present a ϕ-competitive online algorithm for PacketScheduling restricted to 4-bounded instances, matching the lower bound of ϕ. This improves the results from [3] and answers the question posed by Goldwasser in his SIGACT News survey [7].

Our algorithm, which we call ToggleH, can be seen as a modification of EDF_ϕ, which under certain conditions schedules a packet lighter than w_h/ϕ where h is the heaviest pending packet. It maintains one mark that may be assigned to one of the pending packets. For a given step t, we choose the following packets from among all pending packets:

h = the heaviest packet,
s = the second-heaviest packet,
f = the earliest-deadline packet with $w_f \geq w_h/\phi$, and
e = the earliest-deadline packet with $w_e \geq w_h/\phi^2$.

We then proceed as follows:

if (h is marked in the previous step) $\wedge$ ($w_s < w_h/\phi$) $\wedge$ ($d_e = t$)
    schedule e
    if ($d_h = t+3$) $\wedge$ ($d_f = t+2$) **then** mark h
else
    schedule f

We remark that our algorithm uses memory. It is an interesting question whether there is a memoryless ϕ-competitive algorithm for 4-bounded instances.

Algorithms with 1-lookahead. We investigate a variant of PacketScheduling where an online algorithm is able to learn at time t which packets will arrive by time $t+1$. This property is known as *1-lookahead*. From a practical point of view, 1-lookahead corresponds to the situation in which a router can see the packets that are just arriving to the buffer and that will be available for transmission in the next time slot.

The notion of lookahead is quite natural and it has appeared in the online algorithm literature for paging [1], scheduling [11] and bin packing [8] since the 1990s. Ours is the first paper, to our knowledge, that considers lookahead in the context of packet scheduling.

We provide two results about PacketScheduling with 1-lookahead, restricted to 2-bounded instances. We present an online algorithm with competitive ratio of $\frac{1}{2}(\sqrt{13} - 1) \approx 1.303$ and prove a nearly tight lower bound of $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$ on the competitive ratio of algorithms with 1-lookahead which holds already for the 2-bounded case.

The lower bound is constructed in a similar way as the lower bound of ϕ [9, 2, 4]. We can also generalize the construction for ℓ-lookahead which allows the algorithm at time t to see all packets arriving by time $t + \ell$ for an integer $\ell \geq 0$; the lower bound is then $\frac{1}{2(\ell+1)}(1 + \sqrt{5 + 8\ell + 4\ell^2})$ and uses only 2-bounded instances for any ℓ.

The algorithm for 2-bounded instances with 1-lookahead is based on *plans*, called also *provisional schedules*. We define the *plan* in step t to be the optimal schedule in the time interval $[t, \infty)$ that consists of pending and lookahead (i.e., released at $t + 1$) packets at time t. We consider plans to be an important notion for designing algorithms for PacketScheduling; even the 1.828-competitive algorithm by Englert and Westermann uses plans.

References

[1] Susanne Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, 1997.

[2] Nir Andelman, Yishay Mansour, and An Zhu. Competitive queueing policies for QoS switches. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 761–770, 2003.

[3] Francis Y. L. Chin, Marek Chrobak, Stanley P. Y. Fung, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *J. of Discrete Algorithms*, 4(2):255–276, 2006.

[4] Francis Y. L. Chin and Stanley P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.

[5] Marek Chrobak, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Improved online algorithms for buffer management in QoS switches. In *Proc. 12th Annual European Symposium (ESA'04)*, pages 204–215, 2004.

[6] Matthias Englert and Matthias Westermann. Considering suppressed packets improves buffer management in QoS switches. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 209–218, 2007.

[7] Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010.

[8] Edward F. Grove. Online bin packing with lookahead. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 430–436, 1995.

[9] Bruce Hajek. On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time. In *Proc. Conference on Information Sciences and Systems*, pages 434–438, 2001.

[10] Alexander Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir, Baruch Schieber, and Maxim Sviridenko. Buffer overflow management in QoS switches. *SIAM Journal on Computing*, 33(3):563–583, 2004.

[11] Rajeev Motwani, Vijay Saraswat, and Eric Torng. Online scheduling with lookahead: Multipass assembly lines. *INFORMS J. on Computing*, 10(3):331–340, 1998.

An Adversarial Model for Scheduling With Testing

Christoph Dürr * Thomas Erlebach † Nicole Megow ‡
Julie Meißner (Speaker) §

Introduction

We consider a single machine scheduling problem with n jobs under the objectives of minimizing the makespan and the sum of completion times, respectively. Every job j can be tested prior to its execution. A non-tested job has processing time $\bar{p}_j \in \mathbb{Q}^+$, while a tested job has processing time $0 \leqslant p_j \leqslant \bar{p}_j$. It takes one time unit to test a job. Initially, the algorithm knows for each job j only the upper bound $\bar{p}_j$, and gets to know the time p_j only after a test. Tested jobs can be executed at any moment (after their test). We use competitive analysis to assess the performance of algorithms.

Applications for this setting occur when a job j can be processed either under the safe method which takes time $\bar{p}_j$ or under an alternative method which takes time p_j. However, the alternative method is not always possible, and a preliminary test needs to be done first. For example, when a set of programs need to be processed on a single machine, it is possible to run a code optimizer taking unit time before executing a program. The code optimization might reduce the running time of the job, but the amount of improvement is uncertain. Similarly, for the transmission of a set of files it is possible to run a compression algorithm for each file before the transmission. This might significantly reduce the transmission time, but could also fail if the file is uncompressable or already compressed.

Any online algorithm needs to decide upon testing a job without knowing the processing time after testing it. Testing a job early yields new information early in the procedure, but at the same time it delays the completion time of many jobs. Thus, a careful balancing of how many jobs to test and at which time, is necessary.

Related Work. The line of research on optimization with explorable uncertain data has been initiated by Kahan [6] in 1991. His work concerns selection problems with the goal is to minimize the number of queries that is necessary to find the optimal solution. Later, other problems studied in this uncertainty model include finding the k-th smallest value in a set of uncertainty intervals [6, 5], caching problems in distributed databases [10], computing a function value [7], and classical combinatorial optimization problems, such as shortest path [2], the knapsack problem [4], and the MST problem [1, 9].

*christoph.durr@lip6.fr. Sorbonne Universités, UPMC Univ Paris 06, CNRS, Paris, France.
†te17@leicester.ac.uk. Department of Informatics, University of Leicester, UK.
‡nicole.megow@uni-bremen.de. Department of Mathematics and Computer Science, University of Bremen, Germany. Supported by the German Science Foundation (DFG) under contract ME 3825/1.
§jmeiss@math.tu-berlin.de. TU Berlin, Institute of Mathematics, Germany. Supported by Einstein Foundation Berlin in the framework of MATHEON.

While most works aim for minimal query sets to guarantee exact optimal solutions, Olsten and Widom [10] initiate the study of trade-offs between the number of queries and the precision of the found solution. They are concerned with caching problems. Further work in this vein can be found in [7, 2].

Scheduling and testing jobs with uncertain processing time on one machine to minimize the weighted sum of completion times has been introduced in a talk by Levi [8]. They consider a stochastic model, where processing times are drawn independently from a uniform distribution and propose an efficiently solvable dynamic program for it.

Minimizing the Sum of Completion Times

Deterministic algorithms

We give two deterministic algorithms with competitive ratio 2. Our first algorithm, BOUND2 considers jobs in order of non-decreasing $\bar{p}_j$. All jobs with $\bar{p}_j \leq 2$ will be executed untested. Then all remaining jobs are tested. If the revealed processing time of job j is $p_j \leq 2$, then the job is executed immediately after its test. After all pending jobs have been tested, they are scheduled in order of increasing processing time p_j. An example of such a schedule is displayed in Figure 1.

Figure 1: Schedule, where jobs with $\bar{p}_j \leq 2$ are colored blue, and the other jobs colored green if they get delayed and colored red otherwise. The striped schedule sections denote executions of jobs, while the white sections with a colorful frame represent tests of jobs.

We analyze this algorithm by first considering the simpler case where any job has processing time at least 2 or processing time 0 and then showing this is the worst-case.

Theorem 1. *The algorithm* BOUND2 *is a 2-competitive algorithm.*

We also show that alternating the behavior between immediate execution and delayed execution for jobs with $p_j \in (2-\varepsilon, 2+\varepsilon)$ does not improve the competitive ratio.

For a lower bound we consider two jobs with identical large upper limit $\bar{p}_j = M$. We list all possible algorithms executing two jobs and optimize the value of M to get the best-possible lower bound for this example. Setting $M \approx 1.877$ yields

Theorem 2. *There is no deterministic c-competitive algorithm for* $c < \frac{5+\sqrt{241}}{12} \approx 1.71$.

A randomized algorithm

We give a randomized algorithm RTE using two thresholds T and E. The algorithm executes all jobs with $\bar{p}_j < T$ without testing ordered by increasing $\bar{p}_j$. Then it tests all jobs with $\bar{p}_j \geq T$ in random order and schedules each tested job immediately if $p_j \leq E$. Deferred jobs are executed in increasing order of processing-time at the end of the schedule. Again, the algorithm schedule follows the pattern of Figure 1 with blue jobs having $\bar{p}_j \leq T$, red jobs having $p_j < E$, and green jobs having $p_j \geq E$.

We show that one can assume the processing times p_j are from the set $\{0, T, E, E+\varepsilon\}$ for a small $\varepsilon > 0$. Using case analysis we prove there are three tight cases for the ratio of jobs from each class. This insight simplifies the formulas significantly, allowing to find the optimal values $T = 1.74526, E = 2.86091$ for the best algorithm performance.

Theorem 3. *Algorithm* RTE *is* ≈ 1.74*-competitive.*

To derive a lower bound on the best possible competitive ratio, we construct a random instance and apply Yao's principle. The probability distribution over inputs is as follows: There are n jobs and each job j has upper bound $u > 1$. Its processing time p_j is set to 0 with probability $\frac{1}{u}$ and to u with probability $1 - \frac{1}{u}$. We estimate OPT by assuming that there are exactly n/u jobs with $p_j = 0$ and $n(1 - 1/u)$ jobs with $p_j = u$.

The ratio of the expected algorithm cost and the expected optimal cost is maximized for $u = 1.5 + \frac{1}{2}\sqrt{3} \approx 2.366$ which gives by Yao's principle the following lower bound.

Theorem 4. *No randomized algorithm can achieve a competitive ratio less than* 1.62575*.*

Minimizing the Makespan

We consider the same problem setting under the makespan objective. We show that the following algorithm has best-possible competitive ratio: Test a job j if and only if $\bar{p}_j > \varphi$, where φ is the golden ratio, which is roughly 1.618. We show that then the processing time (including a possible test) of any job executed by the algorithm is at most φ times the processing time in the optimal schedule. A lower bound of φ on the competitive ratio of any deterministic algorithm follows from a single job.

Theorem 5. *The algorithm has an optimal competitive ratio of* $\varphi \approx 1.618$*.*

References

[1] T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihalák, and R. Raman. Computing minimum spanning trees with uncertainty. In *Proc. of STACS*, pages 277–288, 2008.

[2] T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62:1–18, 2007.

[3] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32:538–547, 2003.

[4] M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen. The robust knapsack problem with queries. *Computers & OR*, 55:12–22, 2015.

[5] M. Gupta, Y. Sabharwal, and S. Sen. The update complexity of selection and related problems. In *Proceedings of FSTTCS*, volume 13 of *LIPIcs*, pages 325–338, 2011.

[6] S. Kahan. A model for data in motion. In *Proc. of STOC*, pages 267–277, 1991.

[7] S. Khanna and W. C. Tan. On computing functions with uncertainty. In *Proceedings of PODS*, pages 171–182, 2001.

[8] R. Levi. Practice Driven Scheduling Models. Talk at *Dagstuhl Seminar 16081*: Scheduling, 2016.

[9] N. Megow, J. Meißner, and M. Skutella. Randomization helps computing a minimum spanning tree under uncertainty. In *Proceedings of ESA*, pages 878–890, 2015.

[10] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proceedings of VLDB*, pages 144–155, 2000.

Online Chromatic Number is PSPACE-Complete

Martin Böhm (Speaker)* Pavel Veselý*

Introduction. In the classical graph coloring problem we assign a color to each vertex of a given graph such that the graph is properly colored, i.e., no two adjacent vertices have the same color. The chromatic number χ of a graph G is the smallest k such that G can be colored with k distinct colors. Deciding whether the chromatic number of a graph is at most k is well known to be NP-complete, even in the case with three colors.

The online variant of graph coloring can be defined as follows: The vertices of G arrive one by one, and an online algorithm must color vertices as they arrive so that the revealed graph is properly colored at all times. When a vertex arrives, the algorithm sees edges to previously colored vertices. The online algorithm may use additional knowledge of the whole graph G; more precisely, a copy of G is sent to the algorithm at the start of the input. However, the exact correspondence between the incoming vertices and the vertices of the copy of G is not known to the algorithm. This problem is called Online Graph Coloring.

Graph coloring is of considerable interest to the scheduling community due to its application for scheduling objects with general incompatibility constraints. As Halldórsson [4] writes, online graph coloring corresponds to various dynamic scheduling situations, including channel allocation, storage allocation and communication in massively parallel networks.

In this work we focus on a graph parameter called the *online chromatic number* $\chi^O(G)$ of a graph G. This parameter is analogous to the standard chromatic number of a graph: It denotes the smallest number k such that there exists a deterministic online algorithm which is able to color the specified graph G using k colors for any incoming order of vertices.

The online problem Online Graph Coloring has been known since 1976 [1], and the notion of the online chromatic number has appeared first in 1990 [3]. One of the open problems in the area was the computational complexity of deciding whether $\chi^O(G) \leq k$ for a specified simple graph G, given G and k on input; see e.g. Kudahl [6]. We denote this decision problem as Online Chromatic Number. We fully resolve this problem:

Theorem 1 *The decision problem* Online Chromatic Number *is PSPACE-complete.*

As is usual in the online computation model, we can view Online Graph Coloring as a game between two players, which we call Painter (representing the online algorithm) and Drawer (often called Adversary in the online algorithm literature).

*{bohm,vesely}@iuuk.mff.cuni.cz. Computer Science Institute of Charles University, Prague, Czech Republic. Supported by the project 17-09142S of GA ČR. The result was first presented at IWOCA 2016 [2].

In each round Drawer chooses an uncolored vertex v from G and sends it to Painter without telling him to which vertex of G it corresponds, only revealing the edges to the previously sent vertices. Then Painter must properly color ("paint") v, i.e., Painter cannot use a color of a neighbor of v. We stress that in this paper Painter is restricted to be deterministic. The game continues with the next round until all vertices of G are colored.

Deciding the outcome of many two-player games is PSPACE-complete; among those are Amazons, Checkers and Hex, to name a few. However, in most of these games both players have roughly the same power. In Online Graph Coloring, the player Drawer has perfect information (knows which vertices are sent and how they are colored) but Painter does not. This is the main difficulty in proving PSPACE-hardness.

Proof outline. The fact that Online Chromatic Number belongs to PSPACE is not hard to see: The online coloring is represented by a game tree which is evaluated using the Minimax algorithm. This can be done in polynomial space, since the number of rounds in the game is bounded by n, i.e., the number of vertices, and possible moves of each player can be enumerated in polynomial space.

Inspired by [6], we prove the PSPACE-hardness of Online Chromatic Number by a reduction from Q3DNF-SAT, i.e., the satisfiability of a fully quantified formula in the 3-disjunctive normal form (3-DNF). The similar problem of satisfiability of a fully quantified formula in the 3-conjunctive normal form is well known to be PSPACE-complete. Since PSPACE is closed under complement, Q3DNF-SAT is PSPACE-complete as well.

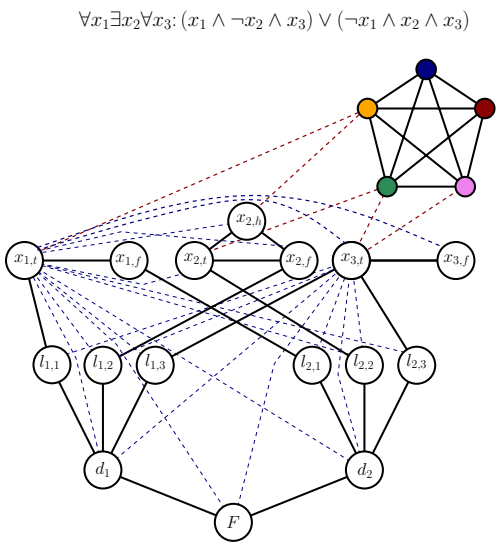


Figure 1: The construction for a sample formula. In the top row of the formula gadgets there are two vertices for a universal quantified variable and three for an existentially quantified one. The layers below are there for each literal, clause and finally one for the final satisfiability test.

Construction with a large precolored part. Our first construction will reduce the PSPACE-complete problem Q3DNF-SAT to Online Coloring with Precoloring with a large precolored part. Given a fully quantified formula Q in the 3-disjunctive normal form, we will create a graph G_1 that will simulate this formula.

Our main resource will be a large precolored clique K_{col} on k vertices and naturally using k colors; the number k will be specified later. Using such a precolored clique, we can restrict the allowed colors on a given uncolored vertex v by connecting it with the appropriate vertices in K_{col}, i.e., we connect v to all vertices in K_{col} which do not have a color allowed for v.

Recall that we see Online Graph Coloring as a game between two players, Painter and Drawer. Assuming that the given formula is unsatisfiable, Drawer first reveals gadgets corresponding to variables. By assigning colors, Painter actually chooses an assignment for the variables. No matter which assignment is selected, the resulting graph will need at least $k + 1$ colors.

Assuming that the given formula is satisfiable, if DRAWER keeps the same order of incoming vertices as before, PAINTER can use the knowledge of the satisfiable assignment to save one color. If DRAWER ignores this ordering and sends one vertex from a later stage, the player PAINTER gains more information about the graph to be able to color it later as if the vertices arrived in the right order.

See Figure 1 for an illustration of the construction.

Construction with a precolored part of logarithmic size. As a second step towards the general case without precoloring, we show how to reduce the required number of precolored vertices to logarithmic size.

We start with the construction of the previous section, but uncolor the large clique. Instead, each vertex of the original construction for the formula now has a *node* associated with it, where node is a simple 3-vertex graph shown in Figure 2. The intuition is as follows: if nodes arrive before the rest of the graph, the game proceeds as in the previous step. If some nodes arrive after other vertices, the player PAINTER can save colors on the nodes themselves.

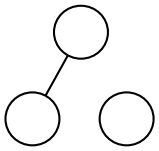


Figure 2: Node

We use a binary encoding on the precolored vertices so that PAINTER can identify all the nodes with only logarithmically many precolored vertices.

Removing the precoloring. We remove the remaining precolored vertices one by one with an inductive lemma. At every step, we multiply the size of the graph by a large constant, making sure that the sheer size of the newly added gadgets allows the player PAINTER to color the entire graph efficiently even when large portions of the construction from previous steps are revealed beforehand. This is the most technical part of our construction and requires careful analysis.

References

[1] D. R. BEAN. *Effective coloration.* The Journal of Symbolic Logic, Vol. 41, No. 2, pp. 469-480 (1976).

[2] M. BÖHM, P. VESELÝ. *Online Chromatic Number is PSPACE-Complete.* In proceedings of *27th International Workshop on Combinatorial Algorithms* (IWOCA 2016). LNCS 9843, 16–28 (2016).

[3] A. GYÁRFÁS, J. LEHEL. *First fit and on-line chromatic number of families of graphs.* Ars Combinatoria 29C, 168–176 (1990).

[4] M. M. HALLDÓRSSON. *Online Coloring Known Graphs.* In SODA: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (1999).

[5] M. M. HALLDÓRSSON. *Online Coloring Known Graphs.* The Electronic Journal of Combinatorics, 7(1), R7 (2000).

[6] C. KUDAHL. *Deciding the On-line Chromatic Number of a Graph with Pre-Coloring is PSPACE-Complete.* In proceedings of *9th International Conference on Algorithms and Complexity* (CIAC 2015). LNCS 9079, 313–324 (2015). Also ArXiv 1406.1623.

Computational Complexity, Resource Augmentation Bounds, and Models for Self-Suspending Real-Time Tasks*

Jian-Jia Chen (Speaker) † Wen-Hung Huang ‡ Georg von der Brüggen §

1 Introduction

In computing systems, an execution entity (job/process/task) may suspend itself when it has to wait for some activities to continue/finish its execution. For real-time embedded systems, such self-suspending behavior has been shown to cause substantial performance/schedulability degradation in the literature.

We consider a system of n sporadic self-suspending tasks. A sporadic task τ_i releases an infinite number of jobs that arrive with the minimum inter-arrival time constraint. A sporadic real-time task τ_i is characterized by its *worst-case execution time* C_i, its *minimum inter-arrival time* (also known as period) T_i and its *relative deadline* D_i. In addition, each job of task τ_i has also a specified worst-case self-suspension time S_i. If the relative deadline D_i of task τ_i in the task set is always equal to (no more than, respectively) the period T_i, such a task set is called an *implicit-deadline* (a *constrained-deadline*, respectively) task set (system).

There are two models that are widely used in the literature: *dynamic* and *segmented* self-suspension (sporadic) task models. The dynamic self-suspension model allows a job of task τ_i to suspend itself at any moment before it finishes as long as the worst-case self-suspension time S_i is not violated. The *segmented* self-suspension model further characterizes the computation segments and suspension intervals as an array $(C_i^1, S_i^1, C_i^2, S_i^2, ..., S_i^{m_i-1}, C_i^{m_i})$, composed of m_i computation segments separated by $m_i - 1$ suspension intervals. A detailed review can be found in [2].

For self-suspending task systems, there are two separated problems: 1) how to *design scheduling policies* to schedule the self-suspending tasks and 2) how to *validate* the schedulability of a scheduling algorithm. The former is referred to as the *scheduler design* problem, whilst the latter is referred to as the *schedulability test* problem.

2 Computational Complexity

It was shown by Ridouard et al. [7] that the scheduler design problem for the segmented self-suspension task model is $\mathcal{NP}$-hard in the strong sense. Lakshmanan and Rajkumar

*This paper is supported by the DFG, as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/).

†jian-jia.chen@cs.uni-dortmund.de Department of Informatics, TU Dortmund, Germany

‡wen-hung.huang@cs.uni-dortmund.de Department of Informatics, TU Dortmund, Germany

§georg.von-der-brueggen@tu-dortmund.de Department of Informatics, TU Dortmund, Germany

Task Model	Scheduler design problem	Schedulability test problem		
Segmented self-suspension	strongly $\mathcal{NP}$-hard [7]	Fixed-Priority Scheduling	Dynamic-Priority Scheduling	
			Constrained Deadlines	Implicit Deadlines
		strongly co$\mathcal{NP}$-hard [1]	strongly co$\mathcal{NP}$-hard (special case is [3])	strongly co$\mathcal{NP}$-hard [1]
Dynamic self-suspension	unbounded (by a constant or by the number of tasks) speedup factors in any FP scheduling, EDF, EDZL, etc. [1] unknown computational complexity	unknown	strongly co$\mathcal{NP}$-hard (special case is [3])	unknown

Table 1: The computational complexity classes (and some speedup factors) of the scheduler design problem and the schedulability test problem for self-suspending tasks.

[5] proposed a pseudo-polynomial-time worst-case response time analysis, which has been recently disproved by Nelissen et al. [6].

We have recently proved that the schedulability analysis for fixed-priority (FP) preemptive scheduling even with only one segmented self-suspending task as the lowest-priority task is co$\mathcal{NP}$-hard in the strong sense when there are more than one self-suspension interval (or equivalently more than two computation segments). The computational complexity analysis is valid for both implicit-deadline and constrained-deadline cases, when the priority assignment is given. Our proof also shows that validating whether there exists a feasible priority assignment is co$\mathcal{NP}$-hard in the strong sense for constrained-deadline segmented self-suspending task systems.

Table 1 summarizes the computational complexity mentioned above.

3 Lower Bounds for Dynamic Self-Suspension Algorithms

For the dynamic self-suspension task model, Huang et al. [4] developed an algorithm with a speedup factor of 2 with respect to the *optimal fixed-priority schedule*. For dynamic self-suspending task systems, the speedup factor for any FP preemptive scheduling, compared to the optimal schedules, is not bounded by a constant if the suspension time cannot be reduced by speeding up. This can be proved by using the following specific task set with two implicit-deadline tasks:

- For task τ_1, we set $T_1 = D_1 = 1$, $S_1 = 0$, and $C_1 = B$.
- For task τ_2, we set $T_2 = D_2 = \frac{1}{B^2}$, $S_2 = \frac{1}{B^2}(1 - B)$, and $C_2 = 1$.

We implicitly assume that $0 < B \leq 0.25$ and $1/B$ is a positive integer. Let $\mathbf{T}^{negative}$ be the above task set. To prove the lower bounds of speedup factors, two steps are involved:

1. We can prove that this task set $\mathbf{T}^{negative}$ can be in fact feasibly scheduled by a simple heuristic algorithm when the system runs at any speed faster than or equal to $2B(\frac{1/B-0.5}{1/B-1})$.
2. We can also prove that this task set is not schedulable by several typical (well-motivated) scheduling algorithms at the original speed 1.

By these, we can conclude that the speedup factor of these scheduling algorithms is at least $\frac{1}{2B}(\frac{1/B-1}{1/B-0.5})$. Since B can be arbitrarily small, the typical scheduling algorithms in real-time systems, including fixed-priority (FP), earliest-deadline-first (EDF), least-laxity-first (LLF), and earliest-deadline-zero-laxity (EDZL) scheduling algorithms.

Details can be found in [1]. How to design good schedulers with a constant speedup factor remains as an open problem.

4 Hybrid Self-Suspension Task Models

The majority of the literature assumes either dynamic or segmented self-suspension patterns. We can also consider *hybrid self-suspension task models* that are more flexible than the segmented self-suspension task model and less pessimistic than the dynamic self-suspension task model. The hybrid self-suspension task models specify the maximum suspension time S_i of task τ_i like the dynamic self-suspension model, with a predefined number of self-suspension intervals. Instead of using $(C_{i,1}, C_{i,2}, \ldots, C_{i,m_i+1})$ as the execution time pattern of the computation segments, the hybrid self-suspension task models provide several options depending on whether the execution pattern of a job can be known when the job arrives to the system or not:

- *Pattern-oblivious*: The execution pattern is unknown before the job finishes its execution. It only assumes that the number of self-suspension intervals of a job of task τ_i is at most $m_i - 1$. However, it is possible that all different execution paths are known offline.
- *Pattern-clairvoyant*: The execution pattern is known when a job arrives, i.e., each job of task τ_i has its individual execution/suspension pattern and the pattern is known at the beginning of a job.

We carefully examine different hybrid self-suspension task models. Empirically, our revised scheduling approaches based on the literature are shown effective in terms of the number of task sets that are schedulable. Compared to the dynamic self-suspension task model and the segmented self-suspension task model (that enforces the execution upper bounds on the computation segments), the hybrid self-suspension task models can achieve different degrees of improvement, depending on the properties of the execution/suspension patterns. The hybrid self-suspension models open a new dimension for suspension-aware real-time embedded systems. For example, in the past, the dynamic self-suspension task model has been widely used for analyzing the multiprocessor synchronization protocols. If the number of suspension intervals is small, our conclusion shows that quantifying the execution/suspension patterns can potentially help improve the schedulability significantly.

References

[1] J.-J. Chen. Computational complexity and speedup factors analyses for self-suspending tasks. In *Real-Time Systems Symposium (RTSS)*, Porto, Portugal, 2016.

[2] J.-J. Chen et al. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. Technical Report 854, Faculty of Informatik, TU Dortmund, 2016. http://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2016-chen-techreport-854.pdf.

[3] P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-Complete. In *27th Euromicro Conference on Real-Time Systems, ECRTS*, pages 281–286, 2015.

[4] W. Huang, J. Chen, H. Zhou, and C. Liu. PASS: priority assignment of real-time tasks with dynamic suspending behavior under fixed-priority scheduling. In *Design Automation Conference*, pages 154:1–154:6, 2015.

[5] K. Lakshmanan and R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12, 2010.

[6] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis. Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 80–89, 2015.

[7] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *RTSS*, pages 47–56, 2004.

Improved Efficient Approximation Schemes for Scheduling Jobs on Identical and Uniform Machines *

Klaus Jansen (Speaker) † Kim-Manuel Klein ‡ José Verschae §

1 Problem Description

Minimum makespan scheduling is one of the fundamental problems in the literature on approximation algorithms [5, 6]. In the *identical machine* setting the problem asks for an assignment of a set of n jobs $\mathcal{J}$ to a set of m identical machines $\mathcal{M}$. Each job $j \in \mathcal{J}$ is characterized by a non-negative processing time $p_j \in \mathbb{Z}_{>0}$. The load of a machine is the total processing time of jobs assigned to it, and our objective is to minimize the *makespan*, that is, the maximum machine load. This problem is usually denoted $P||C_{\max}$.

2 Known Results

It is well known that $P||C_{\max}$ admits a *polynomial time approximation scheme* (PTAS) [7], and there has been many subsequent works improving the running time or deriving PTAS's for more general settings. The first PTAS was given by Hochbaum and Shmoys [7] and had a running time of $(n/\varepsilon)^{O((1/\varepsilon)^2)} = n^{O((1/\varepsilon)^2 \log(1/\varepsilon))}$. This was improved to $n^{O((1/\varepsilon)\log^2(1/\varepsilon))}$ by Leung [12]. Subsequent articles improve further the running time. In particular Hochbaum and Shmoys (see [8]) and Alon et al. [1, 2] obtain an *efficient PTAS* (EPTAS) with running time $2^{(1/\varepsilon)^{\mathrm{poly}(1/\varepsilon)}} + O(n \log n)$; doubly exponentional in $1/\epsilon$. An EPTAS is a PTAS whose running time is $f(1/\varepsilon)poly(|I|)$ where $|I|$ is the encoding size of the input and f is some function. Alon et al. [1, 2] consider general techniques that work for several objective functions, including all L_p-norm of the loads and maximizing the minimum machine load.

The fastest known PTAS for $P||C_{\max}$ achieves a running time of $2^{O(1/\varepsilon^2)\log^3(1/\varepsilon))} + O(n \log n)$ for $(1+\varepsilon)$-approximate solutions [9]. Very recently, Chen et al. [3] showed that, assuming the *exponential time hypothesis* (ETH), there is no PTAS that yields $(1+\varepsilon)$-approximate solutions for $\varepsilon > 0$ with running time $2^{(1/\varepsilon)^{1-\delta}} + poly(n)$ for any $\delta > 0$ [3].

*This work was partially supported by DFG Project, Entwicklung und Analyse von effizienten polynomiellen Approximationsschemata für Scheduling- und verwandte Optimierungsprobleme, Ja 612/14-2, by FONDECYT project 3130407, and by Nucleo Milenio Información y Coordinación en Redes ICM/FIC P10-024F.

†kj@informatik.uni-kiel.de Department of Computer Science, University of Kiel, Christian-Albrechts-Platz 4, 24098 Kiel, Germany.

‡kmk@informatik.uni-kiel.de Department of Computer Science, University of Kiel, Christian-Albrechts-Platz 4, 24098 Kiel, Germany.

§jverschae@uc.cl Facultad de Matemticas and Escuela de Ingeniera, Pontificia Universidad Catlica de Chile, Santiago, Chile.

3 Configuration ILP

Given a guess $T \in \mathbb{N}$ on the optimal makespan, which can be found with binary search, the problem reduces to deciding the existence of a packing of the jobs to m machines (or bins) of capacity T. If we aim for a $(1+\varepsilon)$-approximate solution, for some $\varepsilon > 0$, we can assume that all processing times are integral and T is a constant number, namely $T \in O(1/\varepsilon^2)$. This can be achieved with well known rounding and scaling techniques [1, 2, 8] which is specified also in the full version [10]. Let $\pi_1 < \pi_2 < \ldots < \pi_d$ be the job sizes appearing in the instance after rounding, and let b_k denote the number of jobs of size π_k. The mentioned rounding procedure implies that the number of different job sizes is $d = O((1/\varepsilon)\log(1/\varepsilon))$. Hence, for large n we obtain a highly symmetric problem where several jobs will have the same processing time. Consider the *knapsack polytope* $\mathcal{P} = \{c \in \mathbb{R}^d_{\geq 0} : \pi \cdot c \leq T\}$. A packing on one machine can be expressed as a vector $c \in Q = \mathbb{Z}^d \cap \mathcal{P}$, where c_k denotes the number of jobs of size π_k assigned to the machine. Elements in $Q = \mathbb{Z}^d \cap \mathcal{P}$ are called *configurations*. Considering a variable $x_c \in \mathbb{Z}_{\geq 0}$ that decides the multiplicity of configuration c in the solution, our problem reduces to solving the following linear integer program (ILP):

$$[\text{conf} - \text{ILP}] \quad \sum_{c \in Q} c \cdot x_c = b, \tag{1}$$

$$\sum_{c \in Q} x_c = m, \tag{2}$$

$$x_c \in \mathbb{Z}_{\geq 0} \qquad \text{for all } c \in Q. \tag{3}$$

4 New Results

In this talk we derive new insights on this ILP that help us to design faster algorithms for $P||C_{\max}$. We prove that $P||C_{\max}$ has an EPTAS with running time $2^{O((1/\varepsilon)\log^4(1/\varepsilon))}+O(n)$. Furthermore we study some more general problems. These including makespan scheduling on *uniform machines* $Q||C_{\max}$, and a more general class of objective functions on parallel machines. We show that all these problems admit a PTAS with running time $2^{O((1/\varepsilon)\log^4(1/\varepsilon))} + O(n)$; see also our full version [10]. Hence, our algorithms are best possible up to polylogarithmic factors in the exponent assuming the ETH [3].

Our main technical contribution is a new structural result on the *configuration-ILP*. More precisely, we show the existence of a highly symmetric and sparse optimal solution, in which all but a constant number of machines are assigned a configuration with small support. This structure can then be exploited by integer programming techniques [11, 13] and dynamic programming. We believe that our structural result is of independent interest and should find applications to other settings.

References

[1] N. Alon, Y. Azar, G. Woeginger, and T. Yadid. *Approximation schemes for scheduling*, 8th ACM-SIAM Symposium on Discrete Algorithms (SODA), 1997, 493–500.

[2] N. Alon, Y. Azar, G. Woeginger, and T. Yadid. *Approximation schemes for scheduling on parallel machines*, Journal of Scheduling, 1: 1998, 55–66.

[3] L. Chen, K. Jansen, and G. Zhang. *On the optimality of approximation schemes for the classical scheduling problem*, 25th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2014, 657–668.

[4] F. Eisenbrand and G. Shmonin. *Carathodory bounds for integer cones*, Operations Research Letters, 34:2006, 564–568.

[5] R.L. Graham. *Bounds for certain multiprocessing anomalies*, Bell System Technical Journal, 45:1966, 1563–1581

[6] R.L. Graham. *Bounds on multiprocessing timing anomalies*, SIAM Journal on Applied Mathematics, 17:1969, 416–429.

[7] D.S. Hochbaum and D.B. Shmoys. *Using dual approximation algorithms for scheduling problems: theoretical and practical results*, Journal of the ACM, 34: 1987, 144–162.

[8] D.S. Hochbaum. *Approximation algorithms for NP-hard problems*, PWS Publishing Company, 1997.

[9] 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, 457–485

[10] K. Jansen, K. M. Klein, and J. Verschae. *Closing the gap for makespan scheduling via sparsification techniques*, 43rd International Colloquium on Automata, Languages, and Programming, (ICALP) 2016, 72:1–72:13 and arXiv:1604.07153.

[11] H.W. Lenstra. *Integer programming with a fixed number of variables*, Mathematics of Operations Research, 8:1983, 538–548.

[12] J.Y-T. Leung. *Bin packing with restricted piece sizes*, Information Processing Letters, 31:1989, 145–149.

[13] R. Kannan. *Minkowski's convex body theorem and integer programming*, Mathematics of Operations Research, 12:1987, 415–440.

A 2.542-Approximation for Precedence Constrained Single Machine Scheduling with Release Dates and Total Weighted Completion Time Objective*

Martin Skutella†

We consider the following classical machine scheduling problem denoted by $1|r_j, prec|\sum w_jC_j$ in the standard classification scheme. We are given a set of jobs $N = \{1, 2, \ldots, n\}$ and for every job $j \in N$ a processing time $p_j \geq 0$, a release date $r_j \geq 0$, and a weight $w_j \geq 0$. The jobs $j \in N$ need to be processed during non-overlapping time intervals of length p_j, and j's processing must not start before its release date r_j. Moreover, there are precedence constraints given by a partial order "$\prec$" on N where $j \prec k$ means that job j must be completed before job k may be started, that is, j's processing interval must precede k's. We may therefore without loss of generality assume throughout the paper that $j \prec k$ implies $r_j \leq r_k$. The objective is to minimize the total weighted completion time $\sum_{j\in N} w_jC_j$ where C_j denotes the first point in time at which j's processing is completed.

Known complexity results. Even for unit job weights, the special cases of the problem without non-trivial release dates $1|prec|\sum C_j$ (i.e., $r_j = 0$ for all $j \in N$) or without precedence constraints $1|r_j|\sum C_j$ are strongly NP-hard. In preemptive scheduling, the processing of a job may be repeatedly interrupted and resumed at a later point in time. In the absence of precedence constraints, the problem with unit job weights $1|r_j, pmtn|\sum C_j$ can be solved in polynomial time, but for arbitrary weights $1|r_j, pmtn|\sum w_jC_j$ is strongly NP-hard. Without non-trivial release dates preemptions are superfluous such that $1|prec, pmtn|\sum C_j$ is equivalent to $1|prec|\sum C_j$ and thus strongly NP-hard.

List scheduling. Before dipping into the rich history of approximation algorithms for these scheduling problems, we first discuss the most important algorithmic ingredient for both heuristic and exact solutions: *list scheduling*. Consider a list representing a total order on the set of jobs N, extending the given partial order "$\prec$". A straightforward way to construct a feasible schedule is to process the jobs in the given order as early as possible with respect to release dates. The resulting schedule is a *list schedule*.

Depending on the given list and the release dates of jobs, the machine might remain idle when one job is completed but the next job in the list is not yet released. On the other hand, if job preemptions are allowed, it is certainly not advisable to leave the machine idle while another job at a later position in the list is already available (released) and waiting. Instead, we better start this job and preempt it from the machine as soon as the next job in the list is released. In *preemptive list scheduling* we process at any

*This abstract is essentially taken from the introduction and conclusion of [15].

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

point in time the first available job in the list. The resulting preemptive schedule is feasible (as $j \prec k$ implies $r_j \leq r_k$) and is called *preemptive list schedule*.

Known techniques and results. There is a vast literature on approximation algorithms for the various scheduling problems mentioned above. Here we only mention those results that are particularly relevant in the context of this abstract and refer to Chekuri and Khanna [4] for a more comprehensive overview. Various kinds of linear programming (LP) relaxations have proved to be useful in designing approximation algorithms. One of the simplest and most intuitive classes of LP relaxations is based on completion time variables only. These LP relaxations were introduced by Queyranne [12] and first used in the context of approximation algorithms by Schulz [13], who presents a 2-approximation algorithm for the problem $1|\,prec\,|\sum w_jC_j$ and a 3-approximation algorithm for $1|\,r_j,\,prec\,|\sum w_jC_j$; see also Hall, Schulz, Shmoys, and Wein [9]. These algorithms compute an optimal LP solution and then do list scheduling in order of increasing LP completion times. Moreover, Hall et al. [9] show that preemptive list scheduling in order of increasing LP completion times is a 2-approximation algorithm for $1|\,r_j,\,prec,\,pmtn\,|\sum w_jC_j$.

Phillips, Stein, and Wein [11] and Hall, Shmoys, and Wein [10] introduce the idea of list scheduling in order of so-called α-points to convert preemptive schedules to nonpreemptive ones. For $\alpha \in (0,1]$, the α-point of a job with respect to a preemptive schedule is the first point in time when an α-fraction of the job has been completed. Goemans [7] and Chekuri, Motwani, Natarajan, and Stein [5] show that choosing α randomly leads to better results. In particular, Chekuri et al. [5] present an $e/(e-1)$-approximation algorithm for $1|\,r_j\,|\sum C_j$ by starting from an optimal preemptive schedule. Goemans [7] and Goemans, Queyranne, Schulz, Skutella, and Wang [8] give approximation results for the more general weighted problem $1|\,r_j\,|\sum w_jC_j$ based on a preemptive schedule that is an optimal solution to an LP relaxation in time-indexed variables. Similarly, Schulz and Skutella [14] give an $(e+\varepsilon)$-approximation algorithm for $1|\,r_j,\,prec\,|\sum w_jC_j$ for any $\varepsilon > 0$; see also [16, 17].

Bansal and Khot prove in a recent landmark paper [3] that there is no $(2-\varepsilon)$-approximation algorithm for $1|\,prec\,|\sum w_jC_j$, assuming a stronger version of the Unique Games Conjecture. Ambühl, Mastrolilli, Mutsanas, and Svensson [2], based on earlier work of Correa and Schulz [6] and Ambühl and Mastrolilli [1], prove an interesting relation between the approximability of $1|\,prec\,|\sum w_jC_j$ and the vertex cover problem

Our contribution. We present a $\sqrt{e}/(\sqrt{e}-1)$-approximation algorithm for the problem $1|\,r_j,\,prec\,|\sum w_jC_j$ based on the following two ingredients: (i) For the problem $1|\,r_j,\,prec,\,pmtn\,|\sum w_jC_j$ we slightly strengthen the 2-approximation result of Hall et al. [9], and show that preemptive list scheduling in order of increasing LP completion times on a machine running at double speed yields a schedule whose cost is at most the cost of an optimal schedule on a regular machine. (ii) Modifying the analysis of Chekuri et al. [5] we show how to turn the preemptive schedule on the double speed machine into a nonpreemptive schedule on a regular machine while increasing the objective function by at most a factor of $\sqrt{e}/(\sqrt{e}-1)$.

Conclusion. Despite our enthusiastic yet ultimately fruitless efforts to improve this approximation result, we feel that the new performance ratio $\sqrt{e}/(\sqrt{e}-1)$ is hardly the last word on the considered scheduling problem. On the other hand, the history of approximation algorithms for the special case $1|\,prec\,|\sum w_jC_j$ and, in particular, recent non-approximability results make it seem somewhat unlikely to achieve a performance

ratio strictly better than 2. Therefore, and due lack of imagination of other meaningful approximation ratios, we conclude with the following conjecture, granting an extra $+\varepsilon$ in the performance ratio to the release dates.

Conjecture. For any $\varepsilon > 0$, there is a $(2+\varepsilon)$-approx. alg. for $1|\, r_j,\, prec\,|\sum w_j C_j$.

References

[1] C. Ambühl, M. Mastrolilli, Single machine precedence constrained scheduling is a vertex cover problem, Algorithmica 53 (2009) 488–503.

[2] C. Ambühl, M. Mastrolilli, N. Mutsanas, O. Svensson, On the approximability of single-machine scheduling with precedence constraints, Mathematics of Operations Research 36 (2011) 653–669.

[3] N. Bansal, S. Khot, Optimal long code test with one free bit, in: Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, 2009, pp. 453–462.

[4] C. Chekuri, S. Khanna, Approximation algorithms for minimizing average weighted completion time, in: J. Leung (Ed.), Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, 2004.

[5] C. S. Chekuri, R. Motwani, B. Natarajan, C. Stein, Approximation techniques for average completion time scheduling, SIAM Journal on Computing 31 (2001) 146–166.

[6] J. R. Correa, A. S. Schulz, Single machine scheduling with precedence constraint, Mathematics of Operations Research 30 (2005) 1005–1021.

[7] M. X. Goemans, Improved approximation algorithms for scheduling with release dates, in: Proceedings of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms, New Orleans, LA, 1997, pp. 591–598.

[8] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, Y. Wang, Single machine scheduling with release dates, SIAM Journal on Discrete Mathematics 15 (2002) 165–192.

[9] L. A. Hall, A. S. Schulz, D. B. Shmoys, J. Wein, Scheduling to minimize average completion time: Off-line and on-line approximation algorithms, Mathematics of Operations Research 22 (1997) 513–544.

[10] L. A. Hall, D. B. Shmoys, J. Wein, Scheduling to minimize average completion time: Off-line and on-line algorithms, in: Proceedings of the 7th Annual ACM–SIAM Symposium on Discrete Algorithms, Atlanta, GA, 1996, pp. 142–151.

[11] C. Phillips, C. Stein, J. Wein, Minimizing average completion time in the presence of release dates, Mathematical Programming 82 (1998) 199–223.

[12] M. Queyranne, Structure of a simple scheduling polyhedron, Mathematical Programming 58 (1993) 263–285.

[13] A. S. Schulz, Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds, in: W. H. Cunningham, S. T. McCormick, M. Queyranne (Eds.), Integer Programming and Combinatorial Optimization, Vol. 1084 of Lecture Notes in Computer Science, Springer, 1996, pp. 301–315.

[14] A. S. Schulz, M. Skutella, Random-based scheduling: New approximations and LP lower bounds, in: J. Rolim (Ed.), Randomization and Approximation Techniques in Computer Science, Vol. 1269 of Lecture Notes in Computer Science, Springer, 1997, pp. 119–133.

[15] M. Skutella, A 2.542-approximation for precedence constrained single machine scheduling with release dates and total weighted completion time objective, Operations Research Letters 44 (2016) 676–679.

[16] M. Skutella, Approximation and randomization in scheduling, Ph.D. thesis, Technische Universität Berlin, Germany (1998).

[17] M. Skutella, List scheduling in order of α-points on a single machine, in: E. Bampis, K. Jansen, C. Kenyon (Eds.), Efficient Approximation and Online Algorithms: Recent Progress on Classical Combinatorial Optimization Problems and New Applications, Vol. 3484 of Lecture Notes in Computer Science, Springer, 2006, pp. 250–291.

The General Scheduling Problem With Uniform Release Dates Is Not APX-Hard

Antonios Antoniadis* Ruben Hoeksma (Speaker)† Julie Meißner ‡
José Verschae§ Andreas Wiese¶

1 Introduction

The *general scheduling problem (GSP)* generalizes single machine scheduling problems with total cost objectives. That is, any single machine scheduling problem where jobs have a cost function dependent on their completion time. In its most general setting we define GSP as follows.

Definition 1 (General Scheduling Problem (GSP)) *Given a set of jobs J, each job j with a processing requirement p_j, a release date r_j, and a job-specific cost function $f_j : \mathbb{N} \to \mathbb{N}_0 \cup \{\infty\}$. Schedule the jobs on a single machine while minimizing the sum of the jobs' costs, $\sum f_j$.*

The best known result for this setting is an $O(\log \log P)$-approximation in polynomial time [1]. Noteworthy is that classical objective functions like weighted flow time and weighted tardiness can be modeled in this setting, while even for these objective functions no better results are known.

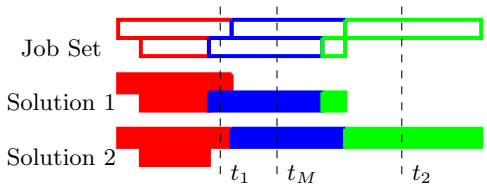
The best known lower bound for GSP is strong NP-hardness, which holds even for uniform release dates ($r_j = 0$ for all j), thus leaving a large gap in our understanding of the complexity of the problem. In the case of uniform release dates this gap is smaller, yet our understanding is still incomplete. When release dates are uniform, there exist a $(4 + \epsilon)$-approximation [3] and an $(e + \epsilon)$-approximation [2] in polynomial and quasi-polynomial time, respectively. In this work we nearly close the gap for the identical release dates setting by showing the following theorem.

Theorem 2 *GSP with identical release dates is not APX-hard, unless* $\mathsf{SAT} \subseteq \mathsf{DTIME}(2^{\mathrm{poly}(\log n)})$.

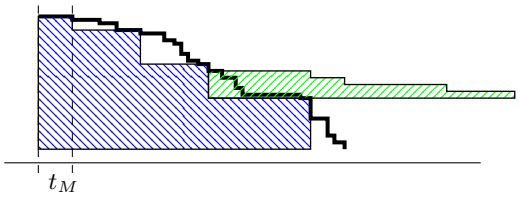
The proof for Theorem 2 consists of a *quasi-polynomial time approximation scheme (QPTAS)*[1] for GSP with identical release dates. The existence of this QPTAS directly implies the theorem.

*antoniad@cs.uni-bonn.de. Department of Computer Science, University of Bonn, Germany.
†rhoeksma@dim.uchile.cl. Center for Mathematical Modeling, Universidad de Chile.
‡jmeiss@math.tu-berlin.de. Institut für Mathematik, Technische Universität Berlin, Germany.
§jverschae@uc.cl. Facultad de Matemáticas & Escuela de Ingeniería, Pontificia Universidad Católica de Chile.
¶awiese@dii.uchile.cl. Departamento de Ingeniería Industrial, Universidad de Chile.

[1] A QPTAS is an approximation scheme running in $O(2^{\mathrm{poly}(\log n)})$ time, where n is the input size.



(a) Picking the second part of the top job rules out Solution 1, while not picking it rules out Solution 2. Note that both solutions have the same total cost, yet are incomparable since on t_1 and t_2 they cover different heights.



(b) The bold curve denotes the height profile of job parts that cover t_M and are selected by the optimal solution. The step function underneath shows an underestimating profile that approximates the former curve. The subprofile estimates the capacity covered by job parts such that the corresponding job has a part that covers t_M and whose end point lies at the third step of the former step function.

Figure 1

2 UFP cover and generalized UFP cover

For identical release dates, no solution of GSP can benefit from preemption of jobs or leaving idle time in the schedule. Therefore, it is a pure sequencing problem. Bansal and Pruhs [1] where first to reduce GSP to a covering problem. Then, Höhn et al. [2] used a reduction to the *Unsplittable Flow on a Path (UFP)*-cover problem together with a QPTAS for that problem to show an $(e + \epsilon)$-approximation for GSP with identical release dates. We use a reduction to a more general version of the UFP cover problem that maintains the approximation factor between the two problems.

Definition 3 (Generalized UFP-cover) *Given are a path P, with for each edge $e \in P$ a demand D_e, and a set of jobs J, with for every job $j \in J$ a height, p_j. Each job j consists of k_j parts, with associated paths $(e_j^0, \ldots, e_j^1), (e_j^1, \ldots, e_j^2), \ldots, (e_j^{k_j-1}, \ldots, e_j^{k_j})$ and costs $c_j^1 \leq \ldots \leq c_j^{k_j}$, where c_j^i denotes the cost for selecting parts $1, \ldots, i$ of job j. The objective is to choose a set of jobs, with for each of those jobs a prefix of parts, such that in each edge $e \in P$ the total height of the jobs of which one of their chosen parts covers e is at least D_e and such that the total cost of the parts is minimized.*

When $k_j = 1$ for all jobs j, this definition gives us exactly the UFP-cover problem. While for the UFP-cover problem the reduction from GSP does not preserve the approximation ratio, for Generalized UFP-cover it does.

Lemma 4 *For any instance of GSP with identical release dates in which each cost function attains only polynomially many different values, we can construct in polynomial time an instance of generalized UFP-cover such that approximations are preserved, i.e., for any $\alpha \geq 1$ an α-approximate solution for the generalized UFP-cover instance can be transformed in polynomial time to an α-approximate solution for the GSP instance.*

3 QPTAS for GSP with identical release dates

Our QPTAS for Generalalized UFP-cover with identical release dates is based on the QPTAS for UFP-cover by Höhn et al. [2]. The basic idea is to group the jobs by cost and

size and then guess an underestimating profile of the demand covered by the selected set of jobs that cover the middle edge, t_M, in the optimal solution. The guessed profile is an increasing step function until t_M and a decreasing step function after t_M. The profile is such that the underestimation of the demand covered (deficit) in each edge is bounded by a constant fraction of the demand covered in t_M. A set of jobs is then chosen such that the underestimating profile is covered. By appending that set of jobs with another set of jobs that covers the deficit, this ensures that the chosen jobs cover at least as much as the set selected in the optimal solution.

In the UFP-cover case, where each job only has a single part, this divides the problem into two independent subproblems on the left and right of t_M. Which can be treated in exactly the same way. In the Generalized UFP-cover case, this is not so straightforward. Figure 1a illustrates that. When we guess the underestimating profile, we cannot just use profiles per part of the jobs, since the choice would greatly influence the options in the left and right subproblems.

To overcome this issue, we go beyond the single level profile and guess subprofiles for all the parts to the right of t_M. This results in different classes of jobs, where jobs in one class have similar height, cost and covering paths for each of their parts (to the right of t_M). See Figure 1b for an impression of these profiles for two parts. The subprofiles essentially consist of guessing the number of jobs that we select in each of the groups that was defined. We show that any such selection of jobs suffices and therefore, that we can delay the actual decision.

Delaying the decision is useful since we can only guess the jobs that actually cover t_M in the final solution. However, the jobs of which the part that covers t_M is not selected may still be needed in the left subproblem. Since we cannot profile these jobs, we delay the decision and first solve the left subproblem with the added information that the solution should fit the profiling of the jobs covering t_M. This corresponds to passing information about this set of jobs, thus subdividing the set that covers t_M into smaller subgroups in every recursive step of the algorithm. In general, this could grow exponentially. However, we show that here this is not the case.

The result is a depth-first recursion that starts on the left of the path P and moves to the right.

References

[1] N. Bansal and K. Pruhs (2010). *The geometry of scheduling.* Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, pp. 407–414.

[2] W. Höhn, J. Mestre, and A. Wiese (2014). *How unsplittable-flow-covering helps scheduling with job-dependent cost functions.* Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, pp. 625–636. Springer.

[3] M. Cheung, J. Mestre, D. Shmoys, and J. Verschae (2017). *A Primal-Dual Approximation Algorithm for Min-Sum Single-Machine Scheduling Problems.* SIAM Journal on Discrete Mathematics. To appear. SIAM

Models and Algorithms for a Partition Problem Arising in Warehousing

Dominik Kress * Nils Boysen † Erwin Pesch (Speaker) ‡

1 Introduction

The storage assignment problem is among the most essential decision problems to be solved in any warehouse. Each stock keeping unit (SKU) is to be assigned a storage position from where it is to be retrieved during order picking. In many warehouses, the detailed slotting, i.e., the decision on the specific shelf each SKU is stored in, is less of an issue, but the problem rather reduces to a partition problem; SKUs are to be jointly stored in one area or group, so that picking-orders can efficiently and conveniently be retrieved without having to access too many groups. In this context, we treat the following basic partition problem, which we denote as the *SKU partition problem* (see our paper [4], which is the foundation of this extended abstract): Consider a given set of SKUs, which are to be partitioned into groups of equal size, and a deterministic set of (weighted) picking-orders each defining a subset of SKUs demanded by an order's customer. Depending on the partitioning of items, orders require different numbers of groups to be accessed during order picking. We refer to these numbers as the orders' *group numbers*. Our objective is to find a partitioning of SKUs that minimizes the weighted sum of group numbers over all picking-orders.

2 An application

In general, partition problems have plenty potential applications in a wide range of areas. We, however, focus on warehousing and consider the assignment of storage space to SKUs in *carousel racks* as an application of the SKU partition problem (see Figure 1).

A carousel is a special kind of automated storage and retrieval system, in which linked shelves or drawers are turned in an oval closed loop. Required SKUs are turned towards the front and are accessed (typically by a human order picker) via a window-like pick face. These parts-to-picker systems either rotate horizontally or vertically and they are typically applied to store small or medium sized items in a space-efficient, yet easy

*dominik.kress@uni-siegen.de. University of Siegen, Management Information Science, Kohlbettstrße 15, 57068 Siegen, Germany

†nils.boysen@uni-jena.de. Friedrich-Schiller-University Jena, Operations Management, Carl-Zeiß-Straße 3, 07743 Jena, Germany

‡erwin.pesch@uni-siegen.de. University of Siegen, Management Information Science, Kohlbettstraße 15, 57068 Siegen, Germany. Center for Advanced Studies in Management, HHL Leipzig, Jahnallee 59, 04109 Leipzig, Germany

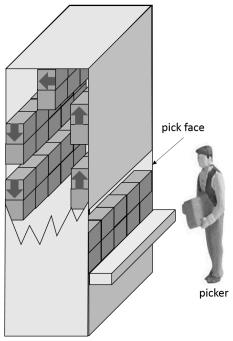


Figure 1: A carousel rack

to access manner. A recent literature survey on carousels is provided in [6]. Depending on the current position of the loop, the picker can conveniently access all active shelves that are currently displayed in the pick face. A major source for picker idle time is the time it takes to rotate the carousel in order to change the set of shelves presented in the pick face. If we think of a subset of SKUs that can simultaneously be accessed as a group, then our basic partition problem can directly be applied to determine groups that minimize the number of carousel moves when having to retrieve a given order set. Order weights can be applied to indicate the frequency with which each order (presumably) occurs.

3 Complexity, models and algorithms

In [4], we prove that the SKU partition problem is NP-hard in the strong sense by reduction from the bisection problem. Furthermore, we present two MIP models. While one of them is rather intuitive, the other formulation is based on a well-known model formulation for the clique partitioning problem. Concerning this latter problem, it is well known that ejection chain heuristics, which are based on ideas presented in [3] and [7], perform well [2, 1]. This motivated us to implement an ejection chain based heuristic for the SKU partition problem. Furthermore, we develop a branch and bound procedure that applies constraint propagation techniques.

In a computational study we show that the ejection chain heuristic tends to provide high-quality solutions within a very short time. Furthermore, with respect to applying the branch and bound method and solving the SKU partition problem with CPLEX based on the different model formulations, we show that CPLEX is well suited for small to medium sized models. When the number of groups is relatively large, the clique partitioning based model formulation performs best. For large instances, the branch and bound procedure results in the best upper bounds on the objective function value and the smallest optimality gaps.

With respect to the practical impact of our research, we compare the results of our group-based perspective with a classical organ pipe arrangement and a random assignment of SKUs to storage positions in carousel racks. The organ pipe arrangement (see, for example, [5]) first sorts the SKUs in nonincreasing order of their overall demand. Then, based on this ordering, the SKUs are assigned to the shelves of the carousel in groups of fixed size. The first group (high level of demand) is assigned to shelf

$h = \lceil (k+1)/2 \rceil$, where k denotes the total number of shelves. The next groups are assigned to shelves $h-1$, $h+1$, $h-2$ and so forth. Our results show that our group-based perspective results in substantially smaller average picker idle times when compared to this classical approach as well as a random assignment of SKUs to the shelves.

References

[1] U. Dorndorf, F. Jaehn and E. Pesch (2008). *Modelling robust flight-gate scheduling as a clique partitioning problem.* Transportation Science, 42, 292–301.

[2] U. Dorndorf and E. Pesch (1994). *Fast clustering algorithms.* ORSA Journal on Computing, 6, 141–153.

[3] F. Glover (1996). *Ejection chains, reference structures and alternating path methods for traveling salesman problems.* Discrete Applied Mathematics, 65, 223–253.

[4] D. Kress, N. Boysen and E. Pesch (2017). *Which items should be stored together? A basic partition problem to assign storage space in group-based storage systems.* IISE Transactions, 49, 13–30.

[5] N. Litvak (2006). *Optimal picking of large orders in carousel systems.* Operations Research Letters, 34, 219–227.

[6] N. Litvak and M. Vlasiou (2010). *A survey on performance analysis of warehouse carousel systems.* Statistica Neerlandica, 64, 401–447.

[7] E. Pesch and F. Glover (1997). *TSP ejection chains.* Discrete Applied Mathematics, 76, 165–181.

A New Approach to Predicting More Reliable Project Runtimes via Probabilistic Model Checking

Ulrich Vogl (Speaker) * Markus Siegle †

1 Introduction

For more than five decades, efforts of calculating exact probabilistic quantiles for arbitrary project runtimes have not been successful due to the tremendous computation requirements, paired with hard restrictions on the available computation power. The methods established today are PERT (Programming Evaluation and Review Technique, [1, p.303-365]) and CCPM (Critical Chain Project Management, [2]). They make simplifying assumptions by focusing on the critical path (PERT) or estimating appropriate buffers (CCPM). In view of this, and since today's machines offer an increased computation power, we have developed a new approach: For the calculation of more exact quantiles or – reversely – of the resulting buffer sizes, we combine the capabilities of classical reduction techniques for series-parallel structures with the capabilities of probabilistic model checking (pMC) [4]. In order to avoid the state space explosion problem, we propose a heuristics algorithm.

2 Problem description and algorithm

The goal is to approximate the runtime distribution of a project task graph which is given as a directed acyclic graph (DAG) with one source and one sink node. Each node (task) is equipped with a stochastic execution time, given as a continuous probability distribution. Our algorithm can be outlined as follows:

- The idea is to reduce the original DAG in a stepwise fashion until the remaining graph consists of only one node whose runtime distribution approximates that of the original overall graph.
- We seek to find subgraphs which can be reduced to a single node by (exact) serial or parallel reduction, as described for example in [3]. Serial reduction means that two serially connected nodes are reduced to a single node whose distribution is the convolution of the two operand distributions. Parallel reduction means that two or more "parallel" nodes are reduced to a single node which is distributed according to the maximum of the operand runtimes.
- When no further series-parallel reduction is possible, we identify the starting and end points of a so-called complex cluster (a generally structured subgraph). We

*ulrich.vogl@unibw.de. Universität der Bundeswehr München, Inst. für Technische Informatik
†markus.siegle@unibw.de. Universität der Bundeswehr München, Inst. für Technische Informatik

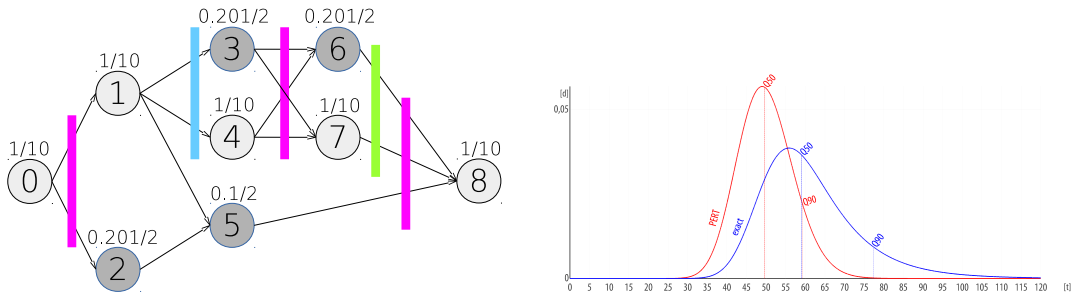
use the concept of syncpoint (see below) to define such clusters. The cluster is then reduced by a *complex reduction* to a single node. This reduction uses pMC.

- In order to avoid state space explosion, it is necessary to limit the size of the graph to be fed into pMC. Therefore the clusters analysed by pMC should be as small as possible. A related challenge consists of finding an appropriate (heuristic) fitting for each source distribution, because pMC tools (such as PRISM [5]) usually only accept exponential distributions.
- It is an interesting side effect that already one complex reduction step can often eliminate a local complexity hotspot and thereby enable further series-parallel reduction steps.

The search for appropriate starting points for cutting out clusters to be reduced can be directed by focusing on particular edge subsets: We call a set $\mathcal{E}$ of edges a *syncpoint*, if and only if for the node set P consisting of the starting points of $\mathcal{E}$ and for the node set S consisting of the end points of $\mathcal{E}$ the following holds:

1. Each edge from P to S is in $\mathcal{E}$.
2. All nodes in P have the same set of 'common' successor nodes
3. All nodes in S have the same set of 'common' predecessor nodes.

If only condition (2) with $|P| > 1$ or condition (3) with $|S| > 1$ holds, we use the concept of backward or forward half syncpoint (BHSP or FHSP).



(a) full, FH(blue), BH(green) SPs (b) PERT calculated vs. precise quantiles

Figure 1: (a) example project graph and (b) computed runtime distributions

3 Practical implementation of the reduction steps

We use a numerical implementation of the series-parallel reduction as described in [3, p.167-226]. Furthermore, for each complex reduction we have to find an appropriate pMC model by fitting the given source distributions. For the following example we use the probabilistic Model Checker PRISM [5]. To represent each given source distribution of a complex cluster, we use the convolution of two Erlang distributions $Erl(\lambda_1, k_1) * Erl(\lambda_2, k_2)$. As long as the coefficient of variation is not greater than 1, the fitting delivers matching first two moments as well as a minimized error at the 3rd moment.

Consider the example graph in Fig. 1(a), where all nodes possess an Erlang distribution (see individual λ/k values at the nodes). Fig. 1 (b) shows the density of the PERT

approach (critical path 0-1-4-7-8 only) as well as the exact target distribution of the entire graph (calculated by our method). It shows how substantial the deviation caused by the PERT-based simplification is: a confidence level of 50% is reported as 90%!

But what about CCPM? Table 1 describes all six possible paths through the graph, each equipped with an associated (feeding) buffer. The Cut & Paste Method (C&PM) [2] delivers the better quantile (84.23% at 72.53 time units), although even there remains a remarkable distance to the desired 90% quantile (appearing at 77.47 time units). Applying the Root Square Error Method (RSEM) [6, p.25ff] we get a still worse result (80.13% quantile at 70.00 time units).

Table 1: CCPM runtimes: path-wise (+buffers), maximum, corr. precise quantile

	0-2-5-8	0-1-5-8	0-1-4-7-8	0-1-4-6-8	0-1-3-7-8	0-1-3-6-8	maximum	quantile
C&PM	44.48	45.80	**48.35**	47.03	47.03	45.71	**72.53**	**84.23**%
feed.buff.	+22.24	+22.90	**+24.18**	+23.52	+23.52	+22.86		
RSEM	**44.48**	45.80	48.35	47.03	47.03	45.71	**70.00**	**80.13**%
feed.buff.	**+25.52**	+23.47	+10.15	+14.27	+14.27	+17.44		

4 Summary and future work

The presented idea offers a remarkable chance to improve the accurancy of established project planning methods. Estimates of the probable runtime – even for complex structures – can be calculated more precisely and also – compared to the customary simulation-based approaches – with manageable computation effort. One of the currently leading management methods, CCPM, can be improved by the combined use of exact calculations and heuristic approximations: For a given project schedule, one obtains a handy calculation of the time-to-finish distribution. In the opposite direction – given a desired project-finalization quantile – one gets a new, better founded calculation method for the CCPM-specific buffer dimensioning. This holds even if the scheduling complexity of the CCPM is increased by an additional calculus regarding the resources- or skill-dependencies (already envisaged in our work plan). Eventually we will use the presented method to solve resource conflicts of the kind "bad multitasking" [2] by taking or hedging a founded decision for a particular prioritization.

References

[1] Shtub, Avraham et al. (2014). *Project Management: Processes, Methodologies and Economics (2nd edition)*. Pearson Education Limited, Essex.

[2] Goldratt, Eliyahu M. (1997). *Critical Chain*. The North River Press Publishing Corporation, Great Barrington.

[3] Klar, Rainer et al. (1995). *Messung und Modellierung paralleler und verteilter Rechensysteme*. B.G. Teubner, Stuttgart

[4] Kwiatkowska, Marta et al. (2007). *Stochastic Model-Checking. Vol. 4486 of Lecture Notes in Computer Science: Formal Methods for Performance Evaluation*. Springer, Heidelberg

[5] Parker, Dave et al. (2016). *The PRISM Model-Checker website*. http://www.prismmodelchecker.org

[6] Dilmaghani, Farhad (2008). *Critical chain project management (CCPM) at Bosch Security Systems (CCTV) Eindhoven: a survey to explore improvement opportunities in the scheduling and monitoring of product development projects, S. 25f.*. Master Thesis, University of Twente

Box Placement as Time Dependent Scheduling To Reduce Automotive Assembly Line Worker Walk Times

Helmut Sedding *

1 Introduction and related work

Henry Ford established moving assembly lines in 1913 for a cost efficient car production in high volumes [5]. The product is moved at a fixed pace on the assembly line, which is divided into several stations. The workload at each station must adhere to a global cycle time. Assigning tasks to stations is well-known as assembly line balancing. For a recent survey on this NP hard optimization problem, see Battaïa and Dolgui [1]. As the line side space at assembly stations is very scarce [3], Bautista and Pereira [2] introduce an extended model, ensuring space requirements are met at each station.

In this work, we focus on increasing the local productivity of a station with a single worker. This may allow to reduce the cycle time or, at least, to better balance the assembly line. Specifically, we focus on shortening the worker's walk times. This non-productive time can contribute to the workload by a significant amount [3]. With the same intention, Scholl et al. [8] optimize walk times between working points and consecutive products. This setting is similar to a scheduling problem with sequence dependent setup times. We, on the other hand, focus on the walks to fetch material and return to the product. Here, walk time additionally depends on the product position, which is time-dependent. This time-dependency adds a further layer of complexity: by changing the duration of tasks at the beginning of a worker's schedule, all following tasks change their position, and thus their walk time. As a consequence, they need to be reoptimized. This contrasts with classical scheduling models, where, e.g., swapping two adjacent jobs influence the objective function, but no other jobs. On the subject of time-dependent scheduling, literature is comprehensively reviewed by Gawiejnowicz [4]. The specific problem of optimizing the sequence of a worker's tasks to minimize walk times is treated in Sedding and Jaehn [9], and Jaehn and Sedding [6]. They introduce a model, analyze the complexity, prove several structural properties, and deduct both exact and heuristic algorithms for solving the problem.

An earlier planning step for the assembly line decides on the line side placement of boxes. In practice, this planning step is largely done by placing full-size card boxes. However, this gives only a limited view on the consequences. In recent years, dedicated software planning tools introduce digital planning assistance for placing the boxes by showing resulting worker paths. However, the placement of boxes is largely manual. An automated placement of boxes is still in its infancy. In the literature, Klampfl et al.

*{firstname.lastname}@uni-ulm.de. Institute of Theoretical Computer Science, Ulm University, 89069 Ulm, Germany.

[7] provide the first operations research view on this problem, with tests on five tasks and boxes. In this work, we introduce a novel model, test a MIP formulation, exact algorithms, and heuristics for this problem.

2 Modeling

First, we provide a definition of the problem setting. We assume that the worker only works at one position at the car, e.g., at the left door to install several parts. Right before each assembly task, the worker needs to pick the respective material from its box, alongside the assembly line. In practice, smaller boxes are placed in shelves with a predetermined number of levels (usually, three or four). We simplify this strategy in our model by placing all boxes on one level. For this, we divide each shelf-placed box width by the number of shelf levels. A solution can afterwards be transformed by using the obtained positions as a guidance point for stacking the boxes in two dimensions. This simplification allows us to tackle the box placement as a one dimensional problem, very similar to a scheduling problem. We also assume that the space at the station is scarce. Hence, it is not necessary to insert gaps between boxes.

Each worker is given a fixed list of tasks $1, \ldots, n$, in set J. Starting at time zero, this list is to be done as early as possible for each product. Consider task $j \in J$. Its material is provided in a box of given width W_j. It is placed at a position π_j, resulting from the box sequence, which is gapless and without overlaps. To optimize this sequence, we regard the box placement as a single machine scheduling problem (of boxes). The objective is to minimize the makespan of the assembly tasks. The processing time of task j is comprised of the variable walk time, and a fixed assembly time of length l_j. The walk time to the box depends on the product position. The product moves with a constant speed along the straight assembly line. Thus, the product position proportionally translates to the start time t_j of the job. In each direction, the walk time grows proportional by distance, by factor a if the box is downstream, and b if the box is upstream. Both factors are in interval $(0, 1)$, as we measure space in time units of the assembly line movement. Then, the objective is to find a box sequence that minimizes the last job's completion time.

3 Solution procedures

To give way for obtaining results with a generic solver, and to clarify the described problem setting, we introduce a suiting MIP formulation. Parameters are the walk time factors $a, b \in (0, 1)$, and, for each job $j = 1, \ldots, n$, length l_j and a box of width W_j:

$$\min C_n \quad \text{subject to:} \tag{1}$$

$$0 = C_0, \text{ and } C_{j-1} + \omega_j + l_j = C_j \quad \text{for all } j = 1, \ldots, n \tag{2}$$

$$-a\delta_j \leq \omega_j, \text{ and } b\delta_j \leq \omega_j \quad \text{for all } j = 1, \ldots, n \tag{3}$$

$$C_{j-1} - \sum_{k=1,\ldots,n} x_{j,k}\pi_{[k]} = \delta_j \quad \text{for all } j = 1, \ldots, n \tag{4}$$

$$0 = \pi_{[1]}, \text{ and } \pi_{[k-1]} + \sum_{j=1,\ldots,n} x_{j,k-1}W_j = \pi_{[k]} \quad \text{for all } j = 2, \ldots, n \tag{5}$$

$$\sum_{j=1,\ldots,n} x_{j,k} = 1 \quad \text{for all } k = 1, \ldots, n \tag{6}$$

$$\sum_{k=1,\dots,n} x_{j,k} = 1 \quad \text{for all } j = 1,\dots,n \tag{7}$$

$$x_{j,k} \in \{0,1\} \quad \text{for all } j,k = 1,\dots,n \tag{8}$$

This formulation allows to set the box sequence by deciding on the binary positional assignment variables $x_{j,k}$. The resulting box position for a job j is $\sum_{k=1,\dots,n} x_{j,k}\pi_{[k]}$.

We conducted tests on solving real world instances ($n \geq 30$) with this model in Gurobi 7.0. However, only smaller instances are solved in adequate time. This necessitates the development of faster algorithms. We analyze the problem structurally and show several combinatorial properties of special cases. These allow us to form several lower bounds, for use in a branch and bound algorithm. Moreover, we propose a dominance rule that eliminates nodes which are guaranteed to be worse than other branches. The preliminary numerical results already show that the newly developed algorithm, implemented in C++, delivers promising results by computing exact solutions much faster than Gurobi's MIP solver.

References

[1] O. Battaïa and A. Dolgui. A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142(2):259–277, 2013.

[2] J. Bautista and J. Pereira. Ant algorithms for a time and space constrained assembly line balancing problem. *European Journal of Operational Research*, 177(3):2016–2032, 2007.

[3] N. Boysen, S. Emde, M. Hoeck, and M. Kauderer. Part logistics in the automotive industry: Decision problems, literature review and research agenda. *European Journal of Operational Research*, 242(1):107–120, 2015.

[4] S. Gawiejnowicz. *Time-dependent scheduling*. Monographs in Theoretical Computer Science. Springer, 2008.

[5] D. A. Hounshell. *From the American system to mass production, 1800 - 1932*. Number 4 in Studies in Industry and Society. Johns Hopkins University Press, 1984.

[6] F. Jaehn and H. A. Sedding. Scheduling with time-dependent discrepancy times. *Journal of Scheduling*, 19(6):737–757, 2016.

[7] E. Klampfl, O. Gusikhin, and G. Rossi. Optimization of workcell layouts in a mixed-model assembly line environment. *International Journal of Flexible Manufacturing Systems*, 17(4):277–299, 2006.

[8] A. Scholl, N. Boysen, and M. Fliedner. The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics. *OR Spectrum*, 35(1):291–320, 2013.

[9] H. A. Sedding and F. Jaehn. Single machine scheduling with nonmonotonic piecewise linear time dependent processing times. In *Proceedings of the 14th International Conference on Project Management and Scheduling*, 222–225. TUM School of Management, 2014.

Online Interval Scheduling with Bounded Number of Failures

Marco Bender * Clemens Thielen † Stephan Westphal (Speaker) ‡

1 Introduction

We consider the problem of scheduling intervals on m identical machines. An interval i is given by a *release date* $r_i \geq 0$ and a *processing requirement* (or *length*) $p_i > 0$. If an interval i is accepted, it must be assigned to start immediately at time r_i on a machine that is currently not occupied. It is then processed for p_i time units, i.e., it finishes at time $r_i + p_i$. The goal is to maximize the number of accepted intervals subject to the constraint that no two intervals assigned to the same machine overlap.

In our online setting, we are initially given a set $\mathcal{I} = \{1, \ldots, n\}$ of intervals with release dates and processing requirements and the information that up to k of these intervals might fail, i.e., they cannot be accepted. We assume throughout the paper that the intervals $\mathcal{I} = \{1, \ldots, n\}$ are sorted in non-decreasing order of release dates and, for identical release dates, in non-increasing order of lengths, where we break ties arbitrarily. We denote this ordering of the intervals by π. Hence, an instance $\sigma = (\mathcal{I}, F)$ of the problem is defined by the set $\mathcal{I}$ of intervals and a subset $F \subseteq \mathcal{I}$ of failing intervals with $|F| \leq k$. An online algorithm initially knows the set $\mathcal{I}$ of intervals and the upper bound k on the number of failures, but only learns whether an interval i fails at the time r_i when it is supposed to start. For each non-failing interval i, the online algorithm has to decide immediately at time r_i whether to accept it and, if it accepts i, to which machine the interval should be assigned. Once a non-failing interval is accepted on some machine, it may not be aborted, preempted, or moved to another machine. We refer to this problem as the *online interval scheduling problem with at most k failures* and write for short k-OIS.

An offline algorithm knows the complete set F of failing intervals in advance and can compute an optimal solution for the remaining intervals in $\mathcal{I} \setminus F$.

We measure the quality of online algorithms for k-OIS by means of *competitive analysis* (cf. [1]). For an instance σ, the profit (number of accepted intervals) obtained by an online algorithm ALG is denoted by $\text{ALG}(\sigma)$, and $\text{OPT}(\sigma)$ denotes the profit of an optimal (offline) solution for instance σ. For $c \geq 1$, a deterministic online algorithm ALG is called *c-competitive* for k-OIS if $\text{OPT}(\sigma) \leq c \cdot \text{ALG}(\sigma)$ for every instance σ. Analogously, if ALG is a randomized online algorithm, it is called *c-competitive (against an*

*marco.bender@inform-software.com. INFORM GmbH, Pascalstr. 35, 52076 Aachen, Germany.

†thielen@mathematik.uni-kl.de. University of Kaiserslautern, Department of Mathematics, Germany.

‡stephan.westphal@tu-clausthal.de. Clausthal University of Technology, Institute for Applied Stochastics and Operations Research, Germany.

oblivious adversary) if $\text{OPT}(\sigma) \leq c \cdot \mathbb{E}[\text{ALG}(\sigma)]$ for every instance σ, where $\mathbb{E}[\text{ALG}(\sigma)]$ denotes the expected value of the profit obtained by ALG on instance σ with respect to the probability distribution defining ALG.

2 Deterministic Online Algorithms

2.1 A Single Machine

We start by providing a lower bound on the competitive ratio of deterministic online algorithms for the case of a single machine.

Theorem 1 *No deterministic online algorithm for k-OIS on a single machine ($m = 1$) can achieve a competitive ratio smaller than* $\max\{2, k\}$.

Definition 2 *We say that an interval i' is* π-contained *in interval i if $i < i'$ (so, in particular, $r_i \leq r_{i'}$) and $r_i + p_i \geq r_{i'} + p_{i'}$, i.e., if i is before i' in the ordering π and $[r_{i'}, r_{i'} + p_{i'}] \subseteq [r_i, r_i + p_i]$. In this case, we also use the notation $i' \subseteq_\pi i$.*

We say that an interval i' sticks out *of an interval i if $r_i < r_{i'} < r_i + p_i$ and $r_{i'} + p_{i'} > r_i + p_i$.*

In the following, we consider the algorithm COUNT for a single machine that accepts a non-failing interval i if the machine is idle at its release date r_i, and if at least one of the intervals $i + 1, \ldots, n$ sticks out of i and i π-contains at most $\max\{k - l(i) - 1, 0\}$ intervals or none of the intervals $i+1, \ldots, n$ stick out of i and i π-contains at most $k - l(i)$ intervals. Here, $l(i)$ denotes the number of failing intervals in $1, \ldots, i - 1$.

In the analysis for the theorems below, we make use of the following argument: If an interval i that is accepted by OPT π-contains another non-failing interval $i' \subseteq_\pi i$, then OPT can simply accept interval i' instead yielding the same number of accepted intervals. This argument helps to analyze the competitive ratio of COUNT and show that it is best possible:

Theorem 3 COUNT *is* $\max\{2, k\}$*-competitive for k-OIS on a single machine ($m = 1$).*

2.2 Multiple Machines

In this section, we present the algorithm MULTI for multiple machines. Intuitively speaking, MULTI runs m copies of (a slightly modified version of) COUNT, one for each machine. If an interval i is released at time r_i, it is given to the first machine. If i is accepted and assigned to machine 1, the processing of interval i is terminated. Otherwise, interval i is rejected on the first machine and we pass it on to machine 2. The procedure is continued until the interval is either accepted on some machine or rejected on all of them.

Theorem 4 MULTI *is* $(4 + \frac{k-3}{m})$*-competitive for k-OIS on $m \geq 2$ machines.*

For multiple machines, we are able to establish the following lower bound for deterministic algorithms:

Theorem 5 *No deterministic online algorithm for k-OIS on m machines can be better than* $\Omega(\sqrt[m]{k})$*-competitive.*

3 Randomized Online Algorithms

We now turn to randomized algorithms and establish the following lower bound by applying Yao's Principle:

Theorem 6 *No randomized online algorithm for k-OIS on m machines can be better than $\Omega\left(\log(k/m)\right)$-competitive.*

3.1 A Randomized Algorithm for Laminar Intervals on a Single Machine

The proof of Theorem 6 uses a special structure for the intervals: The considered set of intervals is *laminar*, i.e., for every pair of intervals, it holds that the intervals are either disjoint or one of them is contained in the other.

We construct a randomized $(\log(k+2))$-competitive algorithm for laminar instances of k-OIS on a single machine. This competitiveness almost matches the lower bound for randomized algorithms shown in Theorem 6 (which holds even for laminar instances).

Theorem 7 *There exists a $(\log(k+2))$-competitive algorithm for k-OIS on a single machine ($m = 1$) when the set of intervals is laminar.*

3.2 Extension of the Algorithm to Multiple Machines

We extend the idea of the randomized algorithm for one machine to $m \geq 2$ machines and prove this way that:

Theorem 8 *There exists a randomized $(\log(k+2))$-competitive algorithm for k-OIS when the set of intervals is laminar.*

References

[1] A. Borodin and R. El-Yaniv (1998). *Online Computation and Competitive Analysis* Cambridge University Press

[2] U. Faigle and W.M. Nawjin (1995). *Note on scheduling intervals online.* Discrete Applied Mathematics 58(1), 13-17

[3] A.W.J. Kolen and J.K. Lenstra and C. Papadimitriou and F.C.R. Spieksma (2007). *Interval scheduling: A survey.* Naval Research Logistics 54, 530-543

[4] M.Y. Kovalyov and C. Ng and T.C.E. Cheng (2007). *Fixed interval scheduling: Models, applications, computational complexity and algorithms.* European Journal of Operational Research 178(2), 331-342

[5] S.O. Krumke and C. Thielen and S. Westphal (2011). *Interval scheduling on related machines.* Computers and Operations Research 38(12), 1836-1844

Scheduling in Advanced OFDMA Wireless Networks: An Algorithmic Perspective

Reuven Cohen (Speaker) * Guy Grebla †

1 Introduction

One of the most important entities in every cellular network is the scheduler logic. This logic needs to determine in real-time which of the data packets awaiting wireless transmission will use the limited spectrum. In modern cellular networks, which use bandwidth of 10Mhz, 20Mhz or even more, the scheduler must be very efficient, because it is invoked hundreds of times every second, and it needs to choose from thousands of waiting data packets.

To meet the exponentially growing demand for mobile data, modern (OFDMA) cellular technologies such as LTE-Advanced require not only additional spectrum but also optimized transmission using the most sophisticated wireless technologies. These technologies include adaptive modulation and coding, carrier aggregation, Fractional Frequency Reuse (FFR), enhanced Inter-Cell Interference Coordination (eICIC), Coordinated Multipoint (CoMP) operation, cell sectorization, and wireless relays.

This paper studies the problem of OFDMA scheduling from the algorithmic perspective in a cellular network with some of the most advanced wireless technologies. The goal is to see how the inclusion of each technology affects each scheduling problem and requires a more sophisticated scheduling algorithm.

2 The Basic Model

In the basic model, each BS has a single omni-directional antenna that transmits packets to the users in its cell. The frame is divided into 10 1ms subframes, and the scheduler needs to make a scheduling decision for every 1ms subframe. A subframe is divided into physical resource blocks. A subcarrier contains 14 OFDMA in each 0.5ms time slot. Each resource block consists of 12 subcarriers in a 0.5ms time slot. Therefore, each subframe contains $12 \cdot 14 = 168$ OFDMA symbols. The bit capacity of a symbol depends on the MCS (modulation and coding scheme) of the packet. For example, with a modulation of 16-QAM and a coding rate of 3/4, each symbol accommodates $4 \cdot 3/4 = 3$ bits. The minimum allocation unit is referred to as a *scheduled block,* which consists of 2 physical resource blocks transmitted one after the other during a 1ms subframe. The number of scheduled blocks in a subframe depends on the system capacity: it is 100 in a 20MHz system, for example.

*rcohen@cs.technion.ac.il. Department of Computer Science, Technion, Haifa 32000, Israel
†grebla@cs.technion.ac.il.

The BS scheduler needs to make real-time scheduling decisions once every 1ms subframe. These decisions must take into account many factors, such as the signal-to-interference-plus-noise ratio (SINR) of every user in the cell, the priority of every user, QoS considerations for each packet, and so forth. In keeping with the concepts proposed in [2], the scheduler is assumed to use a utility function that assigns a *utility* to the transmission of every waiting data packet in the next 1ms subframe.

Before a decision is made whether or not to transmit a given packet in the next subframe, the packet is associated with a *weight.* The weight indicates the number of scheduled blocks (or, equivalently, the number of OFDMA symbols) required for transmitting the packet by the BS. The weight is determined according to the length of the packet and according to the MCS chosen by the scheduler for transmission. A transmission using a more robust MCS, for instance, requires more scheduled blocks than a transmission using a less robust MCS.

Both the utility and the weight of each transmission depend on the MCS. Thus, each MCS can be represented by a {*utility, weight*} pair. The basic model assumes that each packet has a target success probability and is transmitted using the most efficient MCS that guarantees this success probability. For example, if the target success probability is 0.98, and the most efficient MCS that guarantees it requires 3.4 scheduled blocks, then the weight of transmitting this packet is $\lceil 3.4 \rceil = 4$.

It follows that in the basic model every packet can be transmitted using only one MCS, referred to as the "default MCS," and it is associated with only one {*utility, weight*} pair. In such a case, determining which packets will be transmitted in the next subframe is equivalent to finding the set of data packets whose aggregate weight is not larger than the number of scheduled blocks in a subframe and whose aggregate utility is maximum. This, in turn, is equivalent to the well-known NP-complete Knapsack problem.

When the scheduler determines not only which packets will be transmitted but also which MCS will be used for each, the scheduling problem is equivalent to the Multiple Choice Knapsack Problem (MCKP) [1]. MCKP is a generalization of Knapsack, and it is therefore also NP-complete. Like Knapsack, it also has very efficient algorithms: a 2-approximation that works in linear time, and an optimal algorithm that works in pseudo-polynomial time.

3 The Impact of Cell Sectorization on Scheduling

Inter-cell interference is a dominating factor in the performance of cellular networks. If neighboring BSs transmit using the same set of frequencies at the same time, their users will experience bad SINR and the network will perform poorly even if the best scheduler is used. The easiest way to cope with such interference is to divide the available set of frequencies into three subsets ("colors") and to assign a color to each cell such that two neighboring cells will not have the same color. In other words, a big knapsack is divided into multiple independent small knapsacks. Then, the same scheduling algorithms described above can be used, because each BS has to fill its own (small) knapsack, independently of the other BSs. Since this solution decreases the number of scheduling blocks that can be assigned by each scheduler by a factor of 3, more efficient solutions are desired, two of which are cell sectorization and FFR (Fractional Frequency Reuse). These solutions can be used independently, but they are very often combined.

The scheduling problem in each cell can be addressed by running an independent scheduler in each sector. In this case, each sector is viewed as an independent BS with its own antenna and scheduling logic. The scheduling logic has its own knapsack (red, green or blue), and it runs the same algorithms discussed earlier. However, a more efficient approach is that one scheduling logic ("a joint scheduler") performs the scheduling decisions for the three sectors together.

The joint scheduler has to determine not only which packets to transmit in every subframe using which MCS, but also which of the three antennas should be used to transmit each scheduled packet. The optimization problem that such a scheduler needs to solve is no longer Knapsack or MCKP. For simplicity, consider first the case where only one MCS can be used for the transmission to each user from each antenna. This is the most efficient MCS that guarantees the target success probability, referred to earlier as the default MCS. Since the SINR experienced by the user from each antenna is different, the default MCS is different for each antenna. In other words, for each waiting packet, the joint scheduler needs to decide whether to transmit the packet to the target user using the red antenna and MCS(red), the blue antenna and MCS(blue), or the green antenna and MCS(green). This multi-knapsack problem is known as GAP (Generalized Assignment Problem).

Next, consider a joint scheduler that can also make a dynamic selection of the MCS for each packet. For every 1ms subframe, such a scheduler needs to decide which packets to transmit by each antenna and using which MCS. This scheduler has more flexibility than a scheduler that uses the default MCS for each packet, and it is therefore likely to obtain better performance. The combination of GAP and MCKP is a relatively new optimization problem, referred to as MC-GAP (Multiple Choice GAP) [1], and defined as follows:

Instance: Same as in GAP, but each item s_i has multiple configurations (MCSs) to choose from. For each item s_i, a configuration c and a knapsack j, a utility $p_j^c(s_i)$ and a weight $w_j^c(s_i)$ are given.

Objective Determine which items will be selected for each knapsack and using which configuration such that each item is chosen at most once, the aggregate weight on each knapsack does not exceed its capacity, and the aggregate utility is maximized.

The relationship between GAP and MC-GAP is similar to the relationship between Knapsack and MCKP. In particular, [1] shows that any α-approximation algorithm for MCKP can be transformed into a $(1+\alpha)$-approximation algorithm for MC-GAP. This implies that at least from the algorithmic perspective, there is no performance penalty for giving the scheduler more freedom by using MC-GAP instead of GAP.

References

[1] R. Cohen and G. Grebla. Joint scheduling and fast cell selection in OFDMA wireless networks. *IEEE/ACM Transactions on Networking*, 23(1), Feb. 2015.

[2] R. Cohen and L. Katzir. A generic quantitative approach to the scheduling of synchronous packets in a shared uplink wireless channel. *IEEE/ACM Transactions on Networking*, 15(4):932–943, Aug. 2007.

Multiprocessor Real-Time Scheduling with Hierarchical Processor Affinities

Vincenzo Bonifaci [*] Björn Brandenburg[†] Gianlorenzo D'Angelo [‡]
Alberto Marchetti-Spaccamela (Speaker) [§]

1 Introduction

Most modern multiprocessor real-time operating systems offer the possibility to restrict task migration with *affinity masks*, which specify on a per-task basis on which processors a task may be scheduled. The usefulness of processor affinities in several contexts such as application performance, fault tolerance or security is well-documented [5, 6].

We are given a set of n sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$; each task gives rise to a potentially infinite sequence of invocations (or jobs); each job of a task may arrive at any time once a minimum inter-arrival time has elapsed since the arrival of the previous job of the same task, requires a processing time and has to be completed before its deadline.

We assume *priotities* among tasks/jobs. Namely, priority assignment policies used in real-time scheduling can be classified as:
- *fixed priority* (FP): each task has a single fixed priority applied to all of its jobs;
- *job-level fixed priority* (JLFP): jobs of a task may have different priorities, but each job has a single static priority (an example of this is Earliest Deadline First (EDF));
- *job-level dynamic priority* (JLDP): a single job may have different priorities at different times, as for example in Least Laxity First (LLF) scheduling.

The problem of scheduling real-time workloads with *arbitrary processor affinities* (APAs) has been considered [2, 4]. To avoid schedulability losses due to overly simple implementations of processor affinities, two notions of scheduling with arbitrary processor affinities, *weak* and *strong* APA scheduling, have been identified in prior work [4]. Commonly used schedulers, such as Linux's push-and-pull scheduler, implement only weak APA scheduling. However, it has been demonstrated that strong APA scheduling provides improved schedulability, and that it can be realized by leveraging the concept of task shifting, i.e., by allowing higher-priority tasks to be moved among processors in order to make room for lower-priority tasks that are limited by affinity constraints.

Previous work has shown that strong APA scheduling can be implemented with a runtime cost of $O(m^2)$ per task arrival and $O(mn)$ per task departure, where m is the number of processors and n is the number of tasks [4]. The second bound could be large when there are many tasks. We remark that it might be difficult to improve these bounds in general, due to the combinatorial structure of the underlying matching problem.

[*] vincenzo.bonifaci@iasi.cnr.it. Consiglio Nazionale delle Ricerche, Rome, Italy.
[†] bbb@mpi-sws.org. MPI-SWS, Germany.
[‡] gianlorenzo.dangelo@gssi.infn.it. Gran Sasso Science Institute, L'Aquila, Italy.
[§] alberto@dis.uniroma1.it. Sapienza University of Rome, Italy.

We observe that in many practical scenarios affinity masks are not at all arbitrary; rather, they often follow a *hierarchical structure*, since affinity masks commonly mirror the underlying hardware topology. For example, some particular cache-sensitive, high-frequency tasks may be partitioned to a specific processor to ensure L1-cache affinity, whereas other tasks may be restricted to a subset of cores that share an L3 cache, while others yet may be assigned a global affinity mask to optimize their average-case response times. We formalize this notion of hierarchical affinities by requiring that affinity masks follow a *laminar* set structure, that is, given any two affinity masks α and β, either α is a subset of β, or vice versa, or the two sets of processors are disjoint. This definition reflects the tree-like structure of the memory hierarchy.

2 The model

We are given a set of n sporadic tasks $\tau = \{T_1, T_2, \ldots, T_n\}$ to be scheduled on a set of m processors $\pi = \{\Pi_1, \Pi_2, \ldots, \Pi_m\}$. Each task $T_i = (c_i, d_i, p_i)$ is characterized by a *worst-case execution time* c_i, a *relative deadline* d_i, and a *minimum inter-arrival time* or *period* p_i. We assume that c_i, d_i, and p_i are integers and that the tasks have *constrained* deadlines, that is, $d_i \leq p_i$. The extension of the sporadic task model considered here (proposed in [2, 4]) associates with each task $T_i \in \tau$ a *processor affinity* mask $\alpha(T_i) \subseteq \pi$ that is the set of processors on which the jobs of T_i are allowed to be scheduled. We abbreviate $\alpha(T_i)$ as α_i. We assume that the family of affinity masks is *hierarchical* (or *laminar*), that is, for each $i, k = 1, \ldots, n$, either $\alpha_i \subseteq \alpha_k$ or $\alpha_k \subseteq \alpha_i$ or $\alpha_i \cap \alpha_k = \emptyset$. The *level* of an affinity mask α is the number of distinct affinity masks β such that $\beta \subseteq \alpha$. The *height* h of a task system is the maximum level among all the affinity masks of the task system. Note that $h \leq m$ since the affinity masks of a task system form a laminar family; moreover, by standard combinatorial arguments, there are at most $2m$ distinct affinity masks.

We consider both FP and JLFP scheduling. In the case of FP policies, we denote by ϕ_i the priority of T_i. In the case of JLFP policies, we denote by ϕ_i (at any time) the priority of the unique pending job of T_i (at that time).

Let $\tau(t)$ be the set of ready tasks at time t. We represent the scheduler state at time t by a bipartite graph $G(t) = (\tau(t) \cup \pi, E(t))$, where arc (T_i, Π_j) belongs to $E(t)$ iff $\Pi_j \in \alpha_i$. Hence, finding a valid allocation of tasks in $\tau(t)$ to processors π is equivalent to finding a matching $\chi(t)$ in $G(t)$.

However, not all matchings are equally desirable; in particular, one would like to maximize the number of non-idle processors while mantaining the specified priority ordering, without causing affinity violations. Note that in some cases, a processor may have to idle even though tasks are waiting. Two notions have been proposed to formalize how a correct scheduler should behave in this context by Cerqueira et al.[4].

Definition 1. *Weak Invariant. At any time t, for each ready task T_b not matched by $\chi(t)$ and for each $\Pi_j \in \alpha_b$, there exists a task T_i such that $(T_i, \Pi_j) \in \chi(t)$ and $\phi_i \geq \phi_b$.*

As discussed in [4], the above requirement does not consider possible task *shiftings* that could improve schedulability without violating the affinity constraints. To take shiftings into account, a stronger definition is required based on alternating paths in the graph $G(t)$. Given a task T_b not matched to any processor by $\chi(t)$, an *alternating path* $(T_b = T_{\ell_0}, \Pi_{j_1}, T_{\ell_1}, \ldots, \Pi_{j_k}, T_{\ell_k})$, $k \geq 0$, from T_b to task T_{ℓ_k} is a path in $G(t)$

where $(T_{\ell_q}, \Pi_{j_q}) \in \chi(t)$ and $(T_{\ell_{q-1}}, \Pi_{j_q}) \in E(t)$, for each $q = 1, \ldots, k$. A processor Π_j is *reachable* from T_b according to $\chi(t)$ if there exists an alternating path from T_b to a task T_ℓ such that $\Pi_j \in \alpha_\ell$. Let $R_b(t)$ denote the set of processors reachable from task T_b in $G(t)$ with respect to the matching $\chi(t)$.

Definition 2. *Strong Invariant. At any time t, for each ready task T_b not matched according to $\chi(t)$ and for each $\Pi_\ell \in R_b(t)$, there exists a task T_i such that $(T_i, \Pi_\ell) \in \chi(t)$ and $\phi_i \geq \phi_b$.*

3 Results

Theorem 1. *For any set of tasks with hierarchical processor affinities (HPA), there exists a scheduler with a runtime complexity of $O(m)$ per task arrival and $O(\log n + m^2)$ per task departure that maintains the strong invariant.*

The algorithm is conceptually divided into two phases. In the first phase, we select a set τ' of tasks in $\tau(t)$ that will be scheduled at time t. Tasks in τ' are selected in such a way that there exists an assignment of tasks in τ' to processors in π that respects the affinity masks and satisfies the strong APA invariant. In the second phase we find a feasible assignment of tasks in τ' to processors in π according to their affinity masks.

Additionally, in the case of a bilevel affinity hierarchy and when job priorities are based on deadlines, we argue that the performance of our strong HPA scheduler, HPA-EDF, can be related to system optimality in the following way: any collection of jobs that is schedulable (under *any* policy) on m unit-speed processors with the given affinity constraints, is correctly scheduled by HPA-EDF on m processors of speed 2.415.

Finally, we have experimentally validated our approach by implementing a version of our strong HPA scheduler in LITMUS$^{\mathrm{RT}}$ [1], a real-time extension of the Linux kernel.

For a complete version of this work we refer to [3].

References

[1] LITMUS$^{\mathrm{RT}}$: The Linux testbed for multiprocessor scheduling in real-time systems.

[2] S. Baruah and B. B. Brandenburg. Multiprocessor feasibility analysis of recurrent task systems with specified processor affinities. In *RTSS*, pages 160–169, 2013.

[3] V. Bonifaci, B. Brandenburg, G. D'Angelo, and A. Marchetti-Spaccamela. Multiprocessor real-time scheduling with hierarchical processor affinities. In *ECRTS*, pages 215–225, 2016.

[4] F. Cerqueira, A. Gujarati, and B. B. Brandenburg. Linux's processor affinity API, refined: Shifting real-time tasks towards higher schedulability. In *RTSS*, pages 249–259, 2014.

[5] E. Markatos and T. LeBlanc. Using processor affinity in loop scheduling on shared-memory multiprocessors. In *IEEE Trans. on Parallel and Distributed Systems*, 5(4): 379 – 400, 1994.

[6] J. D. Salehi, J. F. Kurose, and D. F. Towsley. The effectiveness of affinity-based scheduling in multiprocessor network protocol processing (extended version). *IEEE/ACM Trans. Netw.*, 4(4):516–530, 1996.

Tight Competitive Analysis for Online TSP on the Line

Antje Bjelde [*] Yann Disser [†] Jan Hackfeld [‡] Christoph Hansknecht [§]
Maarten Lipmann [¶] Julie Meißner [‡] Kevin Schewior (Speaker) [‖]
Miriam Schlöter [‡] Leen Stougie [**]

1 Introduction

We consider the *online traveling-salesperson problem* (TSP) *on the line*. An instance of this problem consists of *requests* $\sigma_1, \sigma_2, \ldots, \sigma_n$ where $\sigma_i = (t_i, a_i)$ for all i. The task is to schedule the tour of a server initially located at the origin and moving at most at unit speed so that, for all i in arbitrary order, it *serves* request σ_i, i.e., visits location a_i on the real line after time t_i. In the *open* version, the *makespan* is the earliest time when this task is fulfilled; in the *closed* version, the server additionally needs to have returned to the origin. We are interested in *online algorithms*, i.e., algorithms that, for all i, only learn about request σ_i at time t_i. We call an algorithm *c-competitive* if, for all instances, its makespan is at most a factor c times larger than the (offline) optimum makespan.

Whereas this problem bears some resemblance to classical online problems such as the k-server problem, it has first been considered about two decades ago by Ausiello et al. [2]. Although online TSP on the line and variants of it have been extensively studied since then, e.g., [1, 3, 5], there are no satisfactory (tight) results in terms of competitive ratio known for this very natural online problem. For the open version, the optimal competitive ratio is known to be at least $(9 + \sqrt{17})/8 \approx 1.64$ and at most 2; for the closed version the gap remains between a trivial 2 and $7/3 \approx 2.33$ [3].

Our main contribution is a tight analysis of both the open and the closed version of the problem. We present algorithms with competitive ratios $(9 + \sqrt{17})/8 \approx 1.64$ and approximately 2.04 for the closed and open version, respectively. While the tightness of the former result follows from the literature, we give a matching lower-bound construction for the latter one. For more details than provided by the observations and intuition given below, we refer to the extended version of this abstract [4].

[*] antje.bjelde@hu-berlin.de. HU Berlin, School of Business and Economics. Supported by Einstein Foundation Berlin in the framework of MATHEON

[†] disser@mathematik.tu-darmstadt.de. TU Darmstadt, Institute of Mathematics, Germany.

[‡] {hackfeld,jmeiss,schloeter}@math.tu-berlin.de. TU Berlin, Institute of Mathematics, Germany. The sixth author was supported by Einstein Foundation Berlin in the framework of MATHEON and by the German Science Foundation (DFG) under contract ME 3825/1. The third and eighth authors were supported by DFG under contract SPP 1736.

[§] c.hansknecht@tu-braunschweig.de. TU Braunschweig, Institute for Mathematical Optimization, Germany.

[¶] Maarten.lipmann@gmail.com. Maarten Lipmann, Amsterdam, Netherlands.

[‖] kschewior@gmail.com. Universidad de Chile, Santiago, Chile. Supported by the Millennium Nucleus Information and Coordination in Networks ICM/FIC RC130003.

[**] l.stougie@vu.nl. Vrije Universiteit, Department of Econometrics and Operations Research & CWI, Amsterdam, Netherlands.

2 Closed online TSP on the line

We show that the best known lower bound for closed online TSP on the line [3] is tight.

Theorem 1 *For $\rho = (9 + \sqrt{17})/8 \approx 1.64$, there is a ρ-competitive algorithm for closed online TSP on the line.*

We provide some intuition. Consider some time t and the current position p_t of the algorithm's server. Among those requests that are released but unserved, let $\sigma^{\mathrm{L}}(t) = (t^{\mathrm{L}}(t), a^{\mathrm{L}}(t))$ and $\sigma^{\mathrm{R}}(t) = (t^{\mathrm{R}}(t), a^{\mathrm{R}}(t))$ be those two with minimal and maximal location, respectively, and call them the *extreme requests*. Here, we assume $\sigma^{\mathrm{L}}, \sigma^{\mathrm{R}}$ exist and $a^{\mathrm{L}}(t) \leq 0 \leq a^{\mathrm{R}}(t)$. In the following, $T_{\mathrm{LR}}(t)$ (and $T_{\mathrm{RL}}(t)$ accordingly) is the tour that starts at the origin at time 0, serves $\sigma^{\mathrm{L}}(t)$, then $\sigma^{\mathrm{R}}(t)$, and then goes back to the origin, all as early as possible.

Our algorithm decides solely depending on t, p_t, and the extreme requests either to serve the requests immediately in some order or to wait. To see why waiting may be necessary, suppose that, at time t, the algorithm immediately visits $\sigma^{\mathrm{L}}(t)$ and $\sigma^{\mathrm{R}}(t)$ in this order. Starting at time 0, the optimum may however immediately serve the requests in the other order. Let $t^\star$ be the time when it serves $\sigma^{\mathrm{L}}(t)$, and let t_0 be the time when the algorithm reaches the origin after having served $\sigma^{\mathrm{L}}(t)$, and assume $t^\star \leq t_0$. Throughout the interval $[t^\star, t_0)$, the optimum might continue moving to the left, resulting in position $p^\star$ at time t_0. By presenting the request $\sigma^\star = (t_0, p^\star)$ and using simple bounds on the algorithm's and optimum's costs, we get that our algorithm needs to fulfill

$$t_0 \geq \frac{\rho \cdot |a^{\mathrm{L}}(t)| - (2-\rho) \cdot t^{\mathrm{L}}(t)}{2\rho - 3}. \tag{1}$$

Likewise we get a symmetric variant of this inequality. In contrast to this lower bound on the waiting time, we derive the following upper bound: If no more requests are released, our algorithm's makespan may be at most $\rho|T_{\mathrm{greedy}}|$ where T_{greedy} is that tour out of $T_{\mathrm{LR}}, T_{\mathrm{RL}}$ with shorter makespan. So, assuming $p_t = 0$, our algorithm may wait at most until $t_1 = \rho|T_{\mathrm{greedy}}| - 2|a^{\mathrm{L}}(t)| - 2|a^{\mathrm{R}}(t)|$. We can show that, by our choice of ρ, there exists a *safe tour* starting in the origin at time 0 and fulfilling both the upper and lower bound on the waiting time.

Indeed, our algorithm tries to be on the safe tour before it serves the first request. If this is impossible, the algorithm tries to at least fulfill Inequality (1) (or its symmetric variant). If even that is not possible, the algorithm simply minimizes the makespan of its tour in case no more requests are released. Handling this case is the main challenge of the analysis.

3 Open online TSP on the line

There is a very simple lower bound of 2 for open online TSP on the line: There is one request σ_1, either $(1, -1)$ or $(1, 1)$, whichever is further away from the online server. Clearly, the optimum makespan is 1, and the online algorithm's is at least 2. Remarkably, the tight lower bound is very close to 2 but *not* exactly 2.

Theorem 2 *Let $\rho \approx 2.04$ be the second-largest root (out of the four real roots) of $9\rho^4 - 18\rho^3 - 78\rho^2 + 210\rho - 107$. There is no $(\rho - \varepsilon)$-competitive algorithm for open TSP on the line for any $\varepsilon > 0$.*

Assume that σ_1 as described above has a positive location. The main idea is to subsequently present requests $\sigma_1^{\mathrm{L}}, \sigma_1^{\mathrm{R}}, \sigma_2^{\mathrm{L}}, \sigma_2^{\mathrm{L}}, \ldots$ alternatingly at positive and negative locations until two of them are *critical*, which we show to happen after a finite number of steps for any ρ'-competitive algorithm with $\rho' \leq \rho$. So assume that two critical (and thus unserved) requests $\sigma^{\mathrm{L}}, \sigma^{\mathrm{R}}$ exist, and the algorithm serves σ^{L} first. We show, however, that the algorithm, either serves σ^{R} too late to be ρ-competitive, or it needs to visit the location of σ^{L} again afterwards due to another request. The resulting makespan can be shown to be too large, and we can use a similar construction when the algorithm serves σ^{R} first.

Theorem 3 *Let $\rho \approx 2.04$ as in Theorem 2. There is a ρ-competitive algorithm for open online TSP on the line.*

This algorithm resembles our algorithm for the closed version of the problem in the sense that it delays its decision on which extreme request to serve first as long as possible subject to staying ρ-competitive in different situations.

4 Further results and future work

A generalization of online TSP on the line is online Dial-A-Ride on the line. Here, every request σ_i needs to be transported from its start location a_i to its target location b_i. There are different versions depending on the capacity c of the server and whether requests may be *preempted*, i.e., dropped off at a location different from their target location. For instance, we give an algorithm with competitive ratio $\sqrt{2}+1 \approx 2.41$ for the preemptive open version with $c \geq 1$, improving the previously best known upper bound of $\sqrt{2}+2 \approx 3.41$. There are however no tight competitive ratios known for *any* version of online Dial-A-Ride on the line.

For the offline version of online TSP on the line, we adapt a known polynomial-time dynamic program for the open version to the closed version. We also settle the complexity of the non-preemptive version, which was only known for $c \in \{1, 2\}$, and show that is NP-hard for any $c \geq 2$. The complexity of few versions remains open, e.g., offline Dial-A-Ride with unbounded capacity.

References

[1] N. Ascheuer, S. O. Krumke, and J. Rambau. Online Dial-a-Ride problems: Minimizing the completion time. In *Proc. of STACS*, pages 639–650, 2000.

[2] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Serving requests with on-line routing. In *Proc. of SWAT*, pages 37–48, 1994.

[3] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001.

[4] A. Bjelde, Y. Disser, J. Hackfeld, C. Hansknecht, M. Lipmann, J. Meißner, K. Schewior, M. Schlöter, and L. Stougie. Tight bounds for online TSP on the line. In *Proc. of SODA*, pages 994–1005, 2017.

[5] S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295(1-3):279–294, 2003.

Complexity and Online Algorithms For a Coloring Problem on a Line

Thomas Erlebach [*] Fu-Hong Liu [†] Hsiang-Hsuan Liu [† ‡] (Speaker)
Mordechai Shalom [§] Prudence W.H. Wong [‡] Shmuel Zaks [¶]

1 Introduction

We consider a coloring problem that arises in the context of wavelength assignment on an optical line network [1, 6]. Related coloring problems on a line network with different objective functions include [2, 3, 7, 8].

We are given a set of n intervals $\mathcal{I} = \{I_1, I_2, \cdots, I_n\}$. Each I_j is an interval $[s_j, e_j)$, where s_j and e_j denote the start and end time of the interval I_j, respectively. Two intervals I_i and I_j are said to be *overlapping* if $I_i \cap I_j \neq \emptyset$. We say that interval I_j contains time t if $t \in I_j$. The load at time t, denoted by $load(t)$ is the number of intervals containing t. The *length* of I_j, denoted by $\ell(I_j)$, is defined as $e_j - s_j$. The maximum and minimum length over all intervals in $\mathcal{I}$ is denoted by $\ell_{\max}$ and $\ell_{\min}$, respectively. The length of any set of intervals $\mathcal{S}$ is defined as the sum of the length of all intervals in $\mathcal{S}$, i.e., $\ell(\mathcal{S}) = \sum_{I \in \mathcal{S}} \ell(I)$. Without loss of generality, we can assume that the intervals in $\mathcal{I}$ form a contiguous interval, otherwise, the scheduling of each contiguous interval is independent of the others, so we can focus on just one such contiguous interval.

We are also given an infinite set of colors $\Lambda = \{1, 2, 3, \cdots\}$, each color i is associated with a cost $\lambda(i) \geq 1$ which is an increasing function such that $\lambda(i) < \lambda(i')$ if $i < i'$. Unless specified otherwise we assume $\lambda(i) = i$. A coloring $\omega : \mathcal{I} \to \Lambda$ is *valid* if for any pair of overlapping intervals $I_i \neq I_j$, $\omega(I_i) \neq \omega(I_j)$. The cost of ω at any time t, denoted by $cost(\omega, t)$, is the maximum cost of the color over all intervals containing t, i.e., $cost(\omega, t) = \max_{I:t \in I} \lambda(\omega(I))$. The total cost of ω, denoted as $cost(\omega)$, is defined as the sum of the cost over all times t, i.e., $cost(\omega) = \int_t cost(\omega, t)$. The objective of the MinSumMax problem is to find a valid coloring ω such that $cost(\omega)$ is minimized.

For any algorithm $\mathcal{A}$ we also denote its coloring by $\mathcal{A}$, and its cost is $cost(\mathcal{A})$. We denote by $\mathcal{O}$ the optimal offline algorithm.

[*] t.erlebach@leicester.ac.uk Department of Informatics, University of Leicester, UK
[†] {fhliu,hhliu}@cs.nthu.edu.tw Department of Computer Science, National Tsing Hua University, Taiwan
[‡] {hhliu,pwong}@liverpool.ac.uk Department of Computer Science, University of Liverpool, UK
[§] cmshalom@telhai.ac.il TelHai College, Upper Galilee, 12210, Israel
[¶] zaks@cs.technion.ac.il Department of Computer Science, Technion, Haifa, Israel

2 Our contribution

2.1 The MinSumMax problem

A more general problem has been studied in [1] which was proved to be NP-complete. On the other hand, their 2-approximation algorithm applies to our problem.

We first show by a reduction from circular arc coloring [5] that, our problem MinSumMax which is simpler remains NP-complete.

Theorem 1. MinSumMax *is NP-complete.*

We then propose online algorithms which give $O(1)$-competitiveness when the ratio of maximum and minimum length of intervals is bounded by a constant and $O(\log \frac{\ell_{\max}}{\ell_{\min}})$-competitiveness for arbitrary length intervals where $\ell_{\max}$ and $\ell_{\min}$ denote the maximum and minimum length of intervals.

Bounded length intervals. Consider bounded length intervals for which we assume there is a constant k such that for any interval I in the input, we have $\ell(I) \in [\ell_{\min}, k \cdot \ell_{\min})$. When an interval $I_j \in \mathcal{I}$ arrives, $\mathcal{G}$ assigns the minimum color that is valid for it, i.e., $\mathcal{G}$ assigns the minimum color i such that for all $j' < j$ and $I_{j'} \cap I_j \neq \emptyset$, we have $\mathcal{G}(I_{j'}) \neq i$.

Our analysis identifies intervals that "contribute" to the cost of a coloring and uses their length to bound the total cost. The *skyline* of a coloring ω is the function of the maximum color used at any time t. We denote the set of intervals on the skyline by $\mathcal{I}_{\mathrm{S}}$ which contains intervals whose color is on the skyline for some time t, i.e., an interval I is in $\mathcal{I}_{\mathrm{S}}$ if there exists $t \in I$ such that $cost(\omega, t) = \lambda(\omega(I))$. We further select a subset $\mathcal{I}_S^*$ of intervals on the skyline of $\mathcal{G}$ from $\mathcal{I}_{\mathrm{S}}$, partition the time horizon into segments based on $\mathcal{I}_S^*$, and show that we can "charge" the cost of $\mathcal{G}$ and $\mathcal{O}$ to this subset, thus allowing us to bound the two costs. The partition of time is based on a notion of extended interval. For any interval I_j, we define its *hat* interval as $I_j^{\mathrm{h}} = [s_j - k\ell_{\min}, e_j + k\ell_{\min})$ and *extended hat* interval as $I_j^{\mathrm{e}} = [s_j - 3k\ell_{\min}, e_j + 3k\ell_{\min})$.

We show that we can select $\mathcal{I}_S^*$ satisfying the following properties. We denote by $q(j)$ the indexes of intervals chosen, i.e., $\mathcal{I}_S^* = \{I_{q(1)}, I_{q(2)}, \cdots\}$ where $q(j) < q(j+1)$.

Lemma 2. *We can select $\mathcal{I}_S^*$ such that for any $j \geq 1$, we have* (i) $I^{\mathrm{h}}_{q(j)} \cap I^{\mathrm{h}}_{q(j+1)} = \emptyset$*;* (ii) $I^{\mathrm{e}}_{q(j)} \cup I^{\mathrm{e}}_{q(j+1)}$ *forms a contiguous interval;* (iii) $s_{q(j+1)} - s_{q(j)} \leq 7k\ell_{\min}$ *and* $e_{q(j+1)} - e_{q(j)} \leq 7k\ell_{\min}$*.* (iv) *For any interval $I \in \mathcal{I}_S$, there exists $j \geq 1$ such that $I \subseteq I^{\mathrm{e}}_{q(j)}$ and $\mathcal{G}(I) \leq \mathcal{G}(I_{q(j)})$.*

The above properties then lead to the following theorem.

Theorem 3. *The greedy algorithm $\mathcal{G}$ is $7k$-competitive when $k = \ell_{\max}/\ell_{\min}$ is a constant.*

Arbitrary length intervals. One can show that the above competitive ratio of the greedy algorithm is tight up to a constant factor, meaning that for arbitrary length intervals, the competitive ratio can be very large. To have a better competitive ratio, we use a standard technique of classification to partition the input intervals $\mathcal{I}$ into $O(\log \frac{\ell_{\max}}{\ell_{\min}})$ classes where in each class the length of the intervals is bounded to be within a ratio of 2. This means that if we apply the greedy algorithm for individual class, then we obtain a competitive ratio of $14 = 2 \times 7$. Let $L = 1 + \lceil \log \frac{\ell_{\max}}{\ell_{\min}} \rceil$. We denote the classes by $C_1, C_2, \cdots, C_L$. We assume that the classified-greedy algorithm, denoted by $\mathcal{C}$, knows about the ratio of the maximum to minimum length of intervals.

$\mathcal{C}$ divides the colors into bands of size L, the i-th color in each band is reserved for intervals in class C_i. Precisely, the color set $\Lambda_i = \{i, i+L, i+2L, i+3L, \cdots\}$ is reserved for intervals in C_i. Intervals in each class C_i are colored independently using the greedy algorithm $\mathcal{G}$ over the color set Λ_i.

At first glance, the competitive ratio of $\mathcal{C}$ is $O(L^2)$. A more detailed analysis reveals that the competitive ratio is indeed $O(L)$.

Theorem 4. *For arbitrary length intervals, $\mathcal{C}$ is $O(\log \frac{\ell_{\max}}{\ell_{\min}})$-competitive.*

Lower bound for arbitrary length intervals. There is a lower bound for any deterministic online algorithm:

Theorem 5. *There is an adversary such that for any deterministic online algorithm $\mathcal{A}$, $cost(\mathcal{A})/cost(\mathcal{O}) \geq \frac{1}{2} \log \frac{\ell_{\max}}{\ell_{\min}}$.*

2.2 Variant — permutation problem

In the MinSumMax problem the algorithm has to assign a color to each interval. We consider a variant of the problem where the input is sets of intervals each of which contains intervals that are pairwise non-overlapping and all such intervals in a set are given the same color. The problem is to find a permutation of the colors such that the cost of the coloring (as defined above) is minimized. This problem may sound easier since we do not need to assign color but only need to find a permutation of colors. It turns out that this variant is still NP-complete via a reduction from the directed optimal linear arrangement problem [4].

References

[1] M. Alicherry and R. Bhatia. Line system design and a generalized coloring problem. In *ESA*, pages 19–30, 2003.

[2] J. Chang, S. Khuller, and K. Mukherjee. LP rounding and combinatorial algorithms for minimizing active and busy time. In *SPAA*, pages 118–127, 2014.

[3] L. Epstein, T. Erlebach, and A. Levin. Variable sized online interval coloring with bandwidth. *Algorithmica*, 53(3):385–401, 2009.

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

[5] M.R. Garey, D.S. Johnson, G.L. Miller, and C.H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Algebraic Discrete Methods*, 1:216–227, 1980.

[6] V. Kumar and A. Rudra. Approximation algorithms for wavelength assignment. In *FSTTCS*, pages 152–163, 2005.

[7] G.B. Mertzios, M. Shalom, A. Voloshin, P.W.H. Wong, and S. Zaks. Optimizing busy time on parallel machines. *TCS*, 562:524–541, 2015.

[8] M. Shalom, A. Voloshin, P.W.H. Wong, F.C.C. Yung, and S. Zaks. Online optimization of busy time on parallel machines. *TCS*, 560:190–206, 2014.

Generalized Lower Bounds for Online Matching on the Line

Antonios Antoniadis * Carsten Fischer (Speaker) † Andreas Tönnis ‡

1 Introduction

The online matching on the line problem is a notorious special case of the online metric matching problem. The task is to match a set of requests R online to a given set of servers S. The distance metric between all points in R and S is a line metric and the objective for the online algorithm is to minimize the sum of distances between matched server-request pairs.

For the more general problem, where the underlying metric is not restricted to the line, the best possible competitive ratio is $2n-1$ and there are several different algorithms that achieve this ratio [5, 4, 7]. These algorithms are closely related to the offline algorithm for weighted, bipartite matching and use a scoring function on alternating paths to guide their decision.

In the more special case of line metrics, the linear lower bound does not hold up. For a long time, the best known lower bound for the problem was 9 and it was believed to be tight. But in 2003, it was shown, that no deterministic algorithm for OML could be better than 9.001-competitive [2] and this remains the best known lower bound.

For upper bounds, the best known algorithms are the work function algorithm and the k-lost cows algorithm. The work function algorithm is inspired by the corresponding algorithm for the k-server problem. It strikes a balances between greedy decisions and the optimal set of free servers in hindsight. The algorithm is conjectured to be $O(\log n)$-competitive, but the best known upper bound for the algorithm is $O(n)$ [6, 8]. The only known deterministic algorithm with a sub-linear competitive ratio is the k-lost cows algorithm by Antoniadis et al. [1]. Their algorithm uses a connection of the matching problem to the lost cow search problem. They give a tight analysis and prove that their algorithm is $\Theta(n^{0.58})$-competitive.

The best known randomized algorithms are due to Gupta and Lewi. In [3], they introduce three algorithms for the online metric matching problem, which are all $O(\log(n))$-competitive.

2 Our contribution: New lower bounds

We describe two important properties of algorithms for online matching on the line. It is well known, that if we want to match a request r to a server, we can restrict ourselves

*antoniad@cs.uni-bonn.de. Universität Bonn, Germany,

†carsten.fischer@uni-bonn.de. Universität Bonn, Germany. Supported by ERC Starting Grant 306465 (BeyondWorstCase).

‡atoennis@uni-bonn.de. Department of Computer Science, University of Bonn, Germany. Supported by ERC Starting Grant 306465 (BeyondWorstCase).

to the nearest free servers s_L and s_R, placed on the left and right of r. We call these two servers the *surrounding servers* for request r. Rob van Stee introduced the concept of *local* algorithms:

Definition 1 ([8]) *Let s_L and s_R be the surrounding servers for request r. An online algorithm is called local if it serves r with one of s_L and s_R and the choice is based only upon the history of servers and requests in the* local-interval *$I = [s_L, s_R]$ of r.*

To the best of our knowledge, all known (deterministic) algorithms, belong to the class of local algorithms. The second natural property we want to introduce is *symmetry*:

Definition 2 *Let A be a local algorithm, and consider the arrival of a request r with a local interval $I = [s_L, s_R]$. Algorithm A chooses one of the servers s_L and s_R to match r to. Let $m = (s_L + s_R)/2$ be the point in the middle of interval I. We say that interval I' is a* mirror *of I if it is the same as I but mirrored by m, i.e., every server and arrived request (including the non-yet matched request r) in the interval is as in $[s_L, s_R]$ but mirrored by m, and the same servers-requests pairs are matched. We say that A is* symmetric*, if for any interval I it chooses to match r to the opposite server than it would do in I's mirrored interval I'.*

Let $\mathcal{A}$ denote the class of algorithms, which behave local and symmetric. It seems to be very natural to behave symmetric for a local algorithm, so $\mathcal{A}$ covers a broad class of algorithms. To be precise: All deterministic algorithms we are aware of, e.g. the work function algorithm or the k-lost-cows algorithm, belong to $\mathcal{A}$. We show that there is no hope to break the $\Omega(\log(n))$-barrier on the competitive ratio of such algorithms.

Theorem 3 *Let A be an algorithm, which belongs to $\mathcal{A}$. Then, there exists an instance with n servers, where n is arbitrary large, such that the competitive ratio of A is in $\Omega(\log(n))$.*

We give a bottom-up construction of the instance based on recursively combining gadgets. The resulting instance resembles a binary tree. The algorithm generates a matching with a lots of crossings, while the optimal solution matches all requests to the next server to the left (or right, respectively). So the cost of the optimum solution on these instances is in $O(n)$, while the algorithm incurs cost of $\Omega(n)$ at each of the $\log(n)$ levels of the instance. We note that the lower-bound instance of Koutsoupias and Nanavati [6] can be considered as a special case of our instance.

We also show that this result can be extended to all algorithms, which allow for such a gadget-construction, without growing the cost of the optimum solution too much. Furthermore, we show that this lower bound also applies to a broad class of randomized algorithms, e.g. the Harmonic-algorithm introduced in [3].

References

[1] A. Antoniadis, N. Barcelo, M. Nugent, K. Pruhs and M. Scquizzato. *A $o(n)$-Competitive Deterministic Algorithm for Online Matching on a Line.* In Proc. 12th Intl. Workshop Approx. and Online Algorithms (WAOA), pages 11–22, 2014.

[2] B. Fuchs, W. Hochstättler and W. Kern. *Online Matching on a Line.* In Theor. Comput. Sci. 332(1-3): 251–264, 2005.

[3] A. Gupta and K. Lewi. *The Online Metric Matching Problem for Doubling Metrics.* In Proc. 39th Intl. Coll. Autom. Lang. Program. (ICALP), pages 424–435, 2012.

[4] B. Kalyanasundaram and K. Pruhs. *Online Weighted Matching.* In J. Algorithms 14(3): 478–488, 1993.

[5] S. Khuller, S.G. Mitchell, and V.V. Vazirani. *On-line Algorithms for Weighted Bipartite Matching and Stable Marriages.* In Theor. Comput. Sci. 127(2): 255–267, 1994.

[6] E. Koutsoupias, A. Nanavati. *The Online Matching Problem on a Line.* In Proc. Intl. Workshop Approx. and Online Algorithms (WAOA), pages 179–191, 2003.

[7] S. Raghvendra. *A Robust and Optimal Online Algorithm for Minimum Metric Bipartite Matching.* In Intl. Workshop Approximation Algorithms for Combinatorial Optimization Problems (APPROX): 18:1–18:16, 2016.

[8] R. van Stee. *Online Matching on the Line.* In SIGACT News 47(1): 99–110, 2016

Branch-and-Cut for Machine Scheduling With Non-Renewable Resources and the $L_{\max}$ Objective*

Péter Györgyi (Speaker) † Tamás Kis ‡

1 Introduction

In a machine scheduling problem with non-renewable resources, besides the machine(s), there are non-renewable resources, like raw material, energy, or money, consumed by the jobs. The non-renewable resources have some initial stock, and they are replenished over time in given quantities. The objective function can be any of the widely-used optimization criteria in machine scheduling problems, see e.g., [2, 3].

In this paper we propose an exact method for the following variant. There is a single machine, a set of n jobs $\mathcal{J}$, and a set of ρ non-renewable resources $\mathcal{R}$. Each job j has a processing time $p_j > 0$, a due-date d_j, and resource requirements $a_{ij} \geq 0$ for $i \in \mathcal{R}$. The non-renewable resources have an initial stock $\tilde{b}_{i,1} \geq 0$ at time $u_1 = 0$, and they are replenished at $q-1$ distinct supply dates $0 < u_2 < \cdots < u_q$ in quantities $\tilde{b}_{i\ell} \geq 0$ for $\ell = 2, \ldots, q$, and $i \in \mathcal{R}$. Note that if $\tilde{b}_{i\ell} = 0$ for some $\ell \geq 2$, then resource i is not supplied at date u_ℓ, or it has an initial stock 0 if $b_{i,1} = 0$. However, for each $i \in \mathcal{R}$, the total demand does not exceed the total supply, i.e., $\sum_{j\in\mathcal{J}} a_{ij} \leq \sum_{\ell=1}^{q} \tilde{b}_{i\ell}$. The cumulative supply of resource i up to supply date u_ℓ is $b_{i\ell} = \sum_{k=1}^{\ell} \tilde{b}_{ik}$. A schedule specifies the starting time S_j of each job $j \in \mathcal{J}$; it is feasible if (i) no pair jobs overlap in time, i.e., $S_{j_1} + p_{j_1} \leq S_{j_2}$ or $S_{j_2} + p_{j_2} \leq S_{j_1}$ for each pair of distinct jobs j_1 and j_2, and (ii) for each resource $i \in \mathcal{R}$, and for each time point t, the total supply until time t is not less than the total consumption of those jobs starting not later than t, i.e., if $u_\ell \leq t$ is the last supply date before t, then $\sum_{j\in J:S_j\leq t} a_{ij} \leq b_{i\ell}$ for each resource $i \in \mathcal{R}$. We aim at finding a feasible schedule S minimizing the maximum lateness$L_{\max}(S) := \max_{j\in\mathcal{J}} S_j + p_j - d_j$.

This problem is NP-hard in the strong sense [2], but it admits a polynomial time approximation scheme if a_j is proportional to p_j even if we have a fixed number of parallel machines [3]. There are sporadic results on exact methods for machine scheduling problems with non-renewable resources. Briskorn et al [1] propose a branch-and-bound method for the problem $1|inv|\sum w_j C_j$, where some of the jobs consume, while other jobs produce some non-renewable resources, and there is an initial stock, but no further replenishments. In order to solve the problem efficiently, the authors propose dominance rules, lower and upper bounds to prune the search-tree. Computational results are reported up to 20-job instances.

*Supported by the OTKA grant 112881

†gyorgyi.peter@sztaki.mta.hu. Institute for Computer Science and Control, Kende str. 13-17, H-1111 Budapest, Hungary

‡kis.tamas@sztaki.mta.hu. Institute for Computer Science and Control, Kende str. 13-17, H-1111 Budapest, Hungary

Our contributions We propose an exact method for solving the single machine scheduling problem with non-renewable resources and resource-consuming jobs and with the maximum lateness objective. Our method is based on branch-and-cut, i.e., linear-programming based branch-and-bound method augmented with cutting planes. We propose a number of inequalities to strengthen the LP relaxation, and present computation results on problem instances with up to 100 jobs.

2 Problem formulation

We use three main types of variables to model the problem. C_j denotes the completion time of job $j \in \mathcal{J}$ and for each ordered pair of jobs $j_1, j_2 \in \mathcal{J}$ with $j_1 < j_2$, the binary variable ord_{j_1,j_2} has value 1 if and only if j_1 precedes j_2 in the schedule. Finally, there are $q \cdot |\mathcal{J}|$ binary decision variables $z_{j\ell}$, $j \in \mathcal{J}, \ell = 1, \ldots, q$ to assign jobs to supplies, i.e., $z_{j\ell} = 1$ if and only if job j only consumes resources supplied at u_ℓ or before. Then $z_{j\ell} \geq z_{j,\ell-1}$ must hold and if $z_{j\ell} - z_{j,\ell-1} = 1$ then job j must be scheduled after u_ℓ. We provide only the most important part of our MIP model:

$$L_{\max} \geq C_j - d_j, \quad j \in \mathcal{J} \tag{1}$$

$$C_j - p_j \geq \sum_{\ell=2}^{q} u_\ell \cdot (z_{j,\ell} - z_{j,\ell-1}), \quad j \in \mathcal{J} \tag{2}$$

$$\sum_{j \in \mathcal{J}} a_{ij} z_{j\ell} \leq b_{i\ell}, \quad \ell = 1, \ldots, q, \ i \in \mathcal{R} \tag{3}$$

$$z_{j,\ell} \geq z_{j,\ell-1}, \quad j \in \mathcal{J}, \ell = 2, \ldots, q \tag{4}$$

The objective is to minimize $L_{\max}$, subject to the above constraints and those ensuring a total ordering of the jobs, which is standard.

3 Cutting planes

We defined four different classes of cuts to strengthen the LP relaxation. First, we have adapted the cutting planes of [4] to our problem:

$$\sum_{j \in S} p_j (d_{\max}(S) + L_{\max} - C_j + p_j) \geq \frac{1}{2} (\sum_{j \in S} p_j^2 + (\sum_{j \in S} p_j)^2), \quad S \subset \mathcal{J},$$

where $d_{\max}(S)$ is the maximum d_j among those jobs in S. The second class consists of cuts derived using a pair of jobs and their impact on $L_{\max}$:

$$L_{\max} \geq C_{j_1} - (p_{j_2} - d_{j_2} + d_{j_1}) \cdot (1 - ord_{j_1,j_2}) + p_{j_2} - d_{j_2}, \quad j_1, j_2 \in \mathcal{J}.$$

The cuts in the third and fourth class may cut off feasible solutions, but they leave at least one optimal solution. That is, the cuts in the third class cut off solutions in which two jobs both starting in some interval $[u_\ell, u_{\ell+1}]$ are not in earliest due-date (EDD) order:

$$ord_{j_1,j_2} \geq z_{j_1,\ell+1} - z_{j_1,\ell} + z_{j_2,\ell+1} - z_{j_2,\ell} - 1, \quad j_1, j_2 \in \mathcal{J}, \ d_{j_1} < d_{j_2},$$

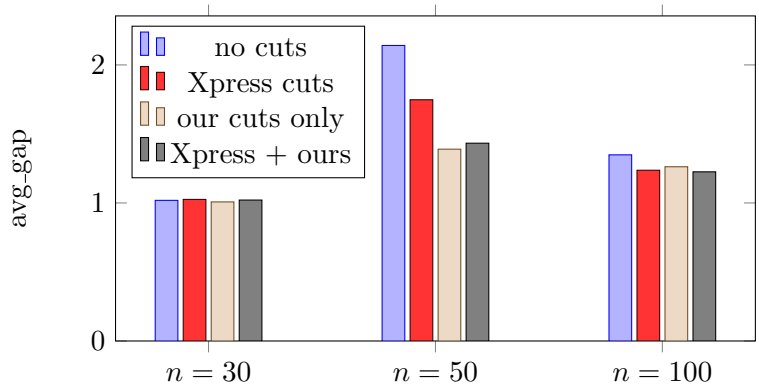


Figure 1: The case $\rho = 1, q = 10$. Each bar represents the average of 10 instances.

whereas those in the fourth class cut off solutions in which the total processing time of those jobs starting in some interval $[u_\ell, u_{\ell+1}]$ is more than $u_{\ell+1} - u_\ell + p_{\max}$, since there always exists an optimal solution respecting this bound:

$$\sum_{j \in \mathcal{J}} (z_{j,\ell} - z_{j,\ell-1}) \cdot p_j \leq u_{\ell+1} - u_\ell + p_{max}, \quad \ell = 1, \ldots, q.$$

4 Computational results

We have generated a data set consisting of 30-, 50-, and 100-job instances, 1 through 10 resource types, and varying number of supply dates. We implemented our method using FICO Xpress and the Mosel language. All computations were performed on a PC with i5 processor, the time limit of each run was 600 seconds.

Figure 1 depicts the average gaps (maximum lateness of the best schedule divided by the best lower bound) for the different settings. As can be seen, the 50-job instances are the hardest in our benchmark set, and on this set the best results are obtained by using only our cuts and no built-in cuts of Xpress at all. Our cuts significantly improve the performance of branch-and-cut especially on the most difficult instances. In most cases the results without the built-in cuts of Xpress are even better. Interestingly, if we have more than one resource, the problem becomes easier in general.

References

[1] Briskorn, D., Jaehn, F., Pesch, E.: Exact algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 16, 105-115 (2013).

[2] Carlier, J.: Problèmes d'ordonnancements à contraintes de ressources: algorithmes et complexité, Thèse d'état, 1984.

[3] Györgyi P., Kis, T., Approximation schemes for parallel machine scheduling with non-renewable resources, *European J. of Operational Research*, 258, 113–123 (2017).

[4] Queyranne, M.: Structure of a simple scheduling polyhedron, *Mathematical Programming*, 58, 1-3, 263–285 (1993).

Optimal Algorithms for Train Shunting and Relaxed List Update Problems

Tim Nonner * Alexander Souza (Speaker) †

1 Introduction

This paper considers a Train Shunting problem where we are given a set J of n cars and a set of stations. Each car has a *source* station and a later *target* station. Moreover, we have a locomotive which visits the stations in a predefined order, and once the locomotive passes the source of a car, it needs to be added to the current train configuration. On the other hand, once its target is passed, it needs to be removed. Any such action is called a *shunting operation*. Adding or removing a car at the end of the train is called an *outer* shunting operation and incurs *low* cost, but adding or removing truly in the interior requires a more complex *inner* shunting operation and thus yields *high* cost. The objective is to schedule the adding and removal of cars as to minimize the total shunting cost. This problem actually originated from a discussion at the Deutsche Bahn AG. Thus, even though it is simple and stated cleanly mathematically, it has a concrete practical application. We will explain later that we can also think of Train Shunting as a relaxed version of the well-known List Update problem.

Related work. Shunting problems are usually considered in the context of a single *hump-yard* or *shunting-yard* which serves as a central facility to rearrange trains. For a detailed description, we refer to the survey of Gatto et al. [5] and the papers of Di Stefano and Koci [4], Beygang, Dahms, and Krumke [2], Jacob et al. [7], and Gatto et al. [9].

By contrast, our problem considers shunting from a more global perspective since the evolution of a train is treated from its origin to its destination. Already in 2006, a similar perspective was taken in a seminal operations research paper by Kroon et al. [8], in which the construction of the dutch timetable is explained.

2 Train Shunting

We derive polynomial time algorithms for Train Shunting by reducing this problem to finding independent sets in bipartite graphs. Our approach works as follows: We first observe that any two cars exclude each other from using a cheap outer shunting operation if they *overlap*. The key observation is then that these overlap dependencies can be captured in a bipartite constraint graph, and our main theorems state that any maximal (with respect to inclusion) independent set in this graph corresponds to a set

*Tamedia AG, Switzerland
†alexander@optineon.com Optineon GmbH, Switzerland

of cars that can be served with cheap outer shunting operations, and vice versa. The proofs show how to convert any maximal independent set into a solution for Train Shunting algorithmically in $\mathcal{O}(n^2)$ running time. It is well-known that the weighted and unweighted Independent Set problem is polynomially solvable in the case of bipartite graphs.

Let s_j denote the event that car j enters the train (at its source) and t_j the event that j leaves (at its target). If $s_j < s_{j'} < t_j < t_{j'}$, then we say that the cars j, j' are *overlapping*, and otherwise, we say that they are *independent*. Especially, if $s_j < s_{j'} < t_{j'} < t_j$, then we say that they are *nested*.

This induces an ordering of the shunting events, it turns out that only the overlapping pairs of cars j, j' are problematic: Either the addition of car j' or the removal of car j can not be a (cheap) outer shunting operation. Therefore, we have to decide which car encounters an outer and which an inner shunting operation. Motivated by this observation, we define the following bipartite graph $G = (S \cup T, E)$ with vertices $S \cup T$ and edges E, where S and T denote the set of source- and target-events, respectively, and for any two overlapping cars j, j', we add the edge $\{s_{j'}, t_j\}$ to the set E.

Now we define solutions of Train Shunting as follows: Any sequence C over elements of J is called a *configuration*, and the left-most element in C is called the *end*. Using this, a solution $\mathcal{C}$ defines a configuration $C(e)$ for each event e such that $j \in C(e)$ for $s_j \leq e < t_j$ and $j \notin C(e)$ for $e < s_j$ and $e \geq t_j$.

The following two main results establish an equivalence between solutions for Train Shunting and independent sets in bipartite graphs.

Theorem 1 *Let $\mathcal{C}$ be any solution of the* Train Shunting *problem and let U be the events having outer shunting operations. Then U is an independent set in G.*

Theorem 2 *Let U be a maximal independent set in G. Then there is a solution $\mathcal{C}$ for* Train Shunting *such that exactly the events U have outer shunting operations.*

These results allow us to treat several variants of the problem in a generic way. Specifically, we obtain an algorithm with running time $\mathcal{O}\left(n^{5/2}\right)$ – using an algorithm of Hopcroft and Karp [6] –for the uniform case, where all low costs and all high costs are identical, respectively. Furthermore, for the non-uniform case we have running time of $\mathcal{O}(n^3)$. Both versions translate to a symmetric variant, where it is also allowed to add and remove cars at the front of the train at low cost. In addition, we formulate a dynamic program with running time $\mathcal{O}(n^4)$, which exploits the special structure of the graph. Although the running time is worse, it allows us to solve many extensions, for instance, economies of scale, dependencies between consecutive stations, and price-collection. Specifically, in the economies of scale variant, we provide a discount if many inner shunting operations are performed at the same station. Dependencies between stations, for example, occur if we want to avoid that two consecutive stations need to perform inner shunting operations. Finally, in the price-collection variant, we get paid for transporting cars, and the goal is find the best tradeoff between profit and shunting operation cost.

3 Relaxed List Update

In the List Update problem, we are given a linked list, which supports the operations *put* and *get*. A call of Put(i, x) stores a data item x at position i in the list, yielding access cost of i. A call of Get(x) returns and removes x (if present) from the list, at access cost equal to the position of x. This classical List Update problem was introduced in a seminal paper by Sleator Tarjan [10]. It was shown by Ambühl [1] that the offline List Update problem is NP-hard.

The Relaxed List Update problem, which we introduce here, features the following cost model: An access at the head of the list encounters *low* cost $c \geq 0$. Otherwise, we are charged a *high* cost of $c' > c$. That is, the access is either cheap, if it is at the head of the list, or expensive, if it is not. Observe that each sequence of put/get operations translates directly into an instance of Train Shunting with uniform low cost c and high cost $c' > c$.

References

[1] Christoph Ambühl. Offline list update is np-hard. In *ESA*, pages 42–51, 2000.

[2] Katharina Beygang, Florian Dahms, and Sven O. Krumke. Train marshalling problem: Algorithms and bounds. *Technical report*, pages 1 – 25, 2010.

[3] Alberto Ceselli, Michael Gatto, Marco E. Lübbecke, Marc Nunkesser, and Heiko Schilling. Optimizing the cargo express service of swiss federal railways. *Transportation Science*, 42(4):450–465, November 2008.

[4] Gabriele Di Stefano and Magnus Love Koci. A graph theoretical approach to the shunting problem. *Electr. Notes Theor. Comput. Sci.*, 92:16–33, 2004.

[5] Michael Gatto, Jens Maue, Matús Mihalák, and Peter Widmayer. Shunting for dummies: An introductory algorithmic survey. In *Robust and Online Large-Scale Optimization*, pages 310–337. 2009.

[6] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225 – 231, 1973.

[7] R. Jacob, P. Marton, J. Maue, and M. Nunkesser. Multistage methods for freight train classification. *Networks*, 57(1):87–105, 2011.

[8] Leo Kroon, Dennis Huisman, Erwin Abbink, Pieter-Jan Fioole, Matteo Fischetti, Gábor Maróti, Alexander Schrijver, Adri Steenbeek, and Roelof Ybema. The new dutch timetable: The or revolution. *Interfaces*, 39(1):6 – 17, 2009.

[9] Marc Nunkesser Michael Gatto, Riko Jacob. Optimization of a railway hub-and-spoke system: Routing and shunting. In *WEA*, 2005.

[10] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202 – 208, 1985.

On the Parameterized Complexity of Scheduling With Side Constraints: Recent Results and New Challenges

René van Bevern *

In a keynote speech at MAPSP 2015, Rolf Niedermeier motivated the parameterized complexity analysis of scheduling problems and, in particular, presented then recent results on the Job Interval Selection Problem [2]. This talk reports on new results on the parameterized complexity of scheduling problems with release times and deadlines [4], precedence constraints [3], and sequence-dependent batch setup times or routing [1].

1 Parallel identical machines, release times and deadlines

The problem $\mathrm{P}|p_j, d_j|$ of checking whether it is feasible to process each of n jobs $j \in J$ non-preemptively for a given amount $p_j \in \mathbb{N}$ of time within a given interval $[r_j, d_j)$, $r_j, p_j \in \mathbb{N}$, on m parallel identical machines is NP-hard even for $m = 1$. Since it generalizes Bin Packing, which becomes trivial if each item fills more than half of a bin, it is natural to study the influence of *looseness* and *slack* on the complexity of $\mathrm{P}|p_j, d_j|$:

Definition 1 *A job $j \in J$ has* looseness $|d_j - r_j|/p_j$ *and* slack $|d_j - r_j| - p_j$.

If all jobs have looseness one, then $\mathrm{P}|p_j, d_j|$ is trivial. Cieliebak et al. [6] showed that $\mathrm{P}|p_j, d_j|$ is polynomial-time solvable for jobs of maximum slack one but NP-hard for any constant maximum slack $\sigma \geq 2$ and for any constant maximum looseness $\lambda > 1$. Using a reduction from Bin Packing, one can strengthen the result on maximum looseness:

Theorem 2 ([4]) *$P|p_j, d_j|$ with any constant maximum looseness $\lambda > 1$ is*
(i) weakly NP-hard even for $m = 2$ machines and
(ii) strongly W[1]-hard parameterized by the number m of machines.

Theorem 2(i) excludes polynomial-time algorithms even when both λ and m are fixed. One can, however, show a pseudo polynomial-time algorithm for each fixed m and λ:

Theorem 3 ([4]) *$P|p_j, d_j|$ with m machines and maximum looseness λ is solvable in $\ell^{O(\lambda m)} \cdot n$ time if the jobs are sorted by release times, where $\ell = \max_{j \in J} |d_j - r_j|$.*

Theorem 2(ii) excludes $f(m) \cdot n^{O(1)}$-time algorithms for any fixed maximum looseness $\lambda > 1$ even if all numbers in the input are bounded polynomially in n. In contrast, one can show a linear-time algorithm if both m and and the maximum slack σ are fixed.

Theorem 4 ([4]) *$P|p_j, d_j|$ with m machines and maximum slack σ is solvable in time $O((\sigma + 1)^{(2\sigma+1)m} \cdot \sigma m \cdot \log \sigma m \cdot n)$ if the jobs are sorted by release times.*

Question 5 ([4]) *Is $P|p_j, d_j|$ NP-hard or polynomial-time solvable when both maximum looseness λ and slack σ are fixed? In case of the latter, is it solvable in $f(\lambda, \sigma) \cdot n^{O(1)}$ time?*

* rvb@nsu.ru. Novosibirsk State University and Sobolev Institute of Mathematics, Novosibirsk, Russia

2 Resource and precedence constraints

Mnich and Wiese [7] have recently shown an $f(p_{\max}) \cdot n^{O(1)}$-time algorithm for $P||C_{\max}$, the problem of non-preemptively processing each of n jobs $j \in J$ for a given amount $p_j \in \mathbb{N}$ of time on m parallel identical machines with minimum makespan, where $p_{\max} = \max_{j \in J} p_j$. They asked whether there is an $f(p_{\max}, w) \cdot n^{O(1)}$-time algorithm for $P|\text{prec}|C_{\max}$, where an additionally given partial order imposes precedence constraints on the jobs and w is the width of the partial order—the cardinality of its largest antichain. For unit processing times, Bodlaender and Fellows [5] showed twenty years earlier:

Theorem 6 ([5]) $P|\text{prec},p_j=1|C_{\max}$ *is W[2]-hard w. r. t. the number m of machines.*

Careful inspection of their proof shows that the hard instances have width $w = m + 1$. This is remarkable since the problem is trivial for $w \leq m$ and since it already answers Mnich and Wiese's question negatively. One can give a stronger negative answer; the sought algorithm does not even exist when the number of machines is fixed to be two:

Theorem 7 ([3]) $P2|\text{prec},p_j \in \{1,2\}|C_{\max}$ *is W[2]-hard w. r. t. the partial order width w.*

Using an additional *lag* parameter, one can derive positive results:

Definition 8 (lag) *Let $(\sigma_j)_{j \in J}$ be the (not necessarily feasible) schedule that starts each job $j \in J$ at the earliest possible time σ_j, i. e.,*
– *$\forall j \in J_0 : \sigma_j = 0$, where J_0 are the jobs without predecessors,*
– *$\forall j \in J \setminus J_0 : \sigma_j = \max_{k \prec j}(\sigma_k + p_k)$, where $\prec$ is the partial order.*
The lag *of a schedule $(s_j)_{j \in J}$ that starts each job $j \in J$ at time s_j is $\max_{j \in J}(s_j - \sigma_j)$.*

Theorem 9 ([3]) *A minimum-makespan schedule with lag at most λ for $P|\text{prec}|C_{\max}$, if one exists, can be found in $(2\lambda + 1)^w \cdot n^{O(1)}$ time, where w is the partial order width.*

The algorithm easily generalizes to the Resource-Constrained Project Scheduling Problem (RCPSP) [3] and is a refinement of a geometric pseudo-polynomial time algorithm for the RCPSP with fixed partial order width w due to Servakh [9]: Theorem 9 can be proved by modeling the problem as the search for a shortest piecewise linear path from one corner of a w-dimensional orthotope to the opposite corner that avoids infeasible suborthotopes and whose linear segments start and end in $2n$ many w-dimensional subhypercubes with edge length 2λ.

The width parameter alone might still lead to positive results for the three-machine problem with unit processing times, since it is a long-standing open question whether it is polynomial-time solvable or NP-hard:

Question 10 ([3]) *Is $P3|p_j=1|C_{\max}$ solvable in $f(k) \cdot n^{O(1)}$ time for any parameter k that does not bound the input size, e. g., for the partial order width?*

3 Open Shop with routing or batch setup times

Open Shop is the problem of processing each job $j \in J$ on each machine $q \in M$ non-preemptively for $p_{iq} \in \mathbb{N}$ time with minimum makespan. Each machine processes only one job at a time; each job is processed by only one machine at a time. In Routing Open Shop, the jobs are located in the vertices of a graph $G = (V, E)$ with edge weights that

determine the time needed for moving machines between vertices. The edge weights also have a natural interpretation as sequence-dependent family or batch setup times.

Routing Open Shop is NP-hard for $|V| = |M| = 2$, yet polynomial-time solvable in this case when allowing preemption [8]. For unit processing times, one can show

Theorem 11 ([1]) *Routing Open Shop with unit processing times is*
(i) solvable in $2^{O(|V||M|^2 \log |V||M|)} \cdot |J|^{O(1)}$ *time and*
(ii) solvable in $O(2^{|V|} \cdot |V|^2) + |J|^{O(1)}$ *time if each vertex contains at least* $|M|$ *jobs.*

Routing Open Shop with unit processing times can be interpreted at the task of processing large batches of roughly equal-length jobs in several storage depots, where processing each individual batch item takes significantly less time than moving a machine from one depot to another. In this context, it is plausible that the requirement on the number of jobs in each vertex in Theorem 11(ii) is fulfilled, yet it seems challenging to get rid of it:

Question 12 ([1]) *Is Routing Open Shop with unit processing times fixed-parameter tractable parameterized by* $|V|$*, i. e., solvable in* $f(|V|) \cdot (|J| + |M|)^{O(1)}$ *time?*

Acknowledgments. This work is supported by RFBR grant 16-31-60007 mol_a_dk.

References

[1] R. van Bevern and A. V. Pyatkin. Completing partial schedules for Open Shop with unit processing times and routing. In *Proc. 11th CSR*, volume 9691 of *LNCS*, pages 73–87, 2016.

[2] R. van Bevern, M. Mnich, R. Niedermeier, and M. Weller. Interval scheduling and colorful independent sets. *J. Sched.*, 18:449–469, 2015.

[3] R. van Bevern, R. Bredereck, L. Bulteau, C. Komusiewicz, N. Talmon, and G. J. Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. In *Proc. 9th DOOR*, volume 9869 of *LNCS*, pages 105–120, 2016.

[4] R. van Bevern, R. Niedermeier, and O. Suchý. A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. *J. Sched.*, 2016. In press.

[5] H. L. Bodlaender and M. R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Oper. Res. Lett.*, 18(2):93–97, 1995.

[6] M. Cieliebak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *Proc. 3rd IFIP TCS*, volume 155 of *IFIP*, pages 209–222, 2004.

[7] M. Mnich and A. Wiese. Scheduling and fixed-parameter tractability. *Math. Program.*, 154(1-2):533–562, 2015.

[8] A. V. Pyatkin and I. D. Chernykh. Zadacha open shop s marshrutizatsiyej na dvukhvershinnoj seti i razresheniyem preryvanij. *Diskretn. Anal. Issled. Oper.*, 19 (3):65–78, 2012. English translation in *J. Appl. Ind. Math.*, 6(3):346-354.

[9] V. V. Servakh. Effektivno rasreshimy sluchaj zadachi kalendarnogo planirovaniya s vozobnovimymi resursami. *Diskretn. Anal. Issled. Oper.*, 7(1):75–82, 2000.

Computing Efficient Pressing Operations for Glued Laminated Timber Production

Heiner Ackermann (Speaker) * Andreas Dinges †

1 Introduction

Glued laminated timber (glulam) consists of several layers of timber glued together with moisture-resistant adhesives. It is used in areas, such as roof construction, where beams with large profiles and high strength are required, and which cannot be sawn from a single trunk. Glulam is available in various profiles.

Glulam production is a long lasting process and requires multiple production stages. First, boards in uniform thickness but different widths are sawn and dried. Secondly, boards of the same widths are jointly processed in batches. Within a batch, glulam with fixed width but different heights, depending on the number of boards stacked, is produced.

In the first step of a batch, boards are finger-jointed in order to produce boards of the required lengths. Subsequently, adhesive is applied and boards are stacked. Thereafter, stacked boards pass a high-frequency press in order to harden the adhesives. The resulting beams are finally planed and packed. Within a pressing operation, several beams can be processed, as long as they are of the same length and do not exceed the maximum pressing height.

Formerly, customers have ordered large numbers of glulam in fixed lengths. For that reason sawmills have preproduced these lengths and served customers from stockpiles. However, nowadays customers are changing their order policies and switch to smaller numbers of non-standard lengths. Cutting these lengths from beams on stock generates a lot of waste material and requires manual handling.

For that reason, sawmills need to switch production from a few standard lengths to individual lengths. This however significantly increases the complexity of production planning.

In this talk we focus on the computation of pressing operations for a single batch and present a heuristic for solving this problem. Before that, we summarize the constraints and the planning goals.

*heiner.ackermann@itwm.fraunhofer.de. Fraunhofer Institute for Industrial Mathematics ITWM, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany.

†andreas.dinges@itwm.fraunhofer.de. Fraunhofer Institute for Industrial Mathematics ITWM, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany.

2 Constraints & Goals

- Within a batch, all lengths requested by a single customer shall be produced in sequence.
- Pressing operations must not fall below (exceed) minimum (maximum) pressing lengths and heights.
- Minimize the number of height changes between subsequent pressing operations, as these generate significant ramp-up times.
- If needed, it is feasible to prolong beams and to fill up pressing operations.
- Minimize the total filling volume and maximize the length of filling pieces.

3 The Heuristic

The heuristic we propose runs in three phases. In the first phase, it computes pressing operations for each customer individually. In the second phase, it combines infeasible precomputed sequences. In the last phase, it determines the processing order of precomputed and combined pressing sequences.

3.1 Single Customer

Assume that a single customer orders beams of identical height but of different lenghts. Furthermore, assume that all beams exceed the minimum pressing length. In this case it is straightforward to construct a sequence of pressing operations with the following properties:

- Their lenghts coincide with ordered lenghts.
- All operations are feasible and have the same height, potentially except the last one.
- Fill-up pieces have maximal lenghts.

If some ordered beams fall below the minimum pressing length, we try to combine them into feasible lenghts. We can achieve this by solving bin-packing problems with varying bin sizes and consider all beams assinged to the same bin as a joint beam.

If there actually is just a single customer, we fill up the potentially infeasible last pressing operation.

3.2 Multiple Customers

Assume now that mutiple customers order beams. For each customer, we precompute a sequence of pressing operations as described above and select those with an infeasible last pressing operation. Since the number of different customers is typically small, we can enumerate all *meaningful* combinations of precomputed pressing sequences. A combination is meaningful if we reverse one sequence and process the two infeasible operations in a joint operation. For each such combination we compute the total fill-up volume.

Finally, we select a set of combinations such that each of the precomputed sequences is contained in one of them. We achieve this via solving a set-partitioning problem on the set of all combinations.

3.3 Determining the Processing Order

Finally, we determine the processing order of the precomputed and combined pressing sequences in order to minimize set-up times between pressing operations of different heights. We achieve this via a reduction to a travelling-salesperson problem.

4 Implementation

We have implemented the heuristic in C# using Google OR Tools to solve both the set-partitioning and travelling-salesperson problem. The heuristic runs efficiently on real-world instances and outperforms solutions generated by human experts. A detailed evaluation and theoretical analysis is subject to future work.

A New Model of Continuous Learning and Its Applications In Scheduling

Bartłomiej Przybylski *

1 Introduction

In many real-world applications of scheduling theory the processing times of jobs are variable, not fixed. Job processing time may then change in response to some environmental factors, such as the amount of resources available, its starting time or its position in a schedule. Recently, a part of the latter case—scheduling problems with *learning effect*—is gaining more and more attention. We refer the reader to Biskup [2], Agnetis et al. [1] and Strusevich and Rustogi [9] for more details on this subject.

The first model of learning effect in scheduling, where the processing time of a job depends on the number of jobs executed earlier, was proposed by Gawiejnowicz [4]. An equivalent variation of the model was introduced by Biskup [3]. In this case the actual processing time of the j-th job placed on the r-th position on a particular machine is the product of its basic processing time $\bar{p}_j$ and the so-called learning factor, that is $p_{j,r} = \bar{p}_j r^a$, where $a < 0$ is a learning index. This model was then extended, for example, by Mosheiov and Sidney [6] who analysed the case, where the learning indices are related to jobs, that is $p_{j,r} = \bar{p}_j r^{a_j}$. One of the models where the actual processing time of a job does not depend on the number of jobs executed earlier, but on the sum of their processing times, is the model introduced by Kuo and Yang [5] who assumed that $p_{j,r} = \bar{p}_j(1 + \sum_{k=1}^{r-1} \bar{p}_{[k]})^a$, where $\bar{p}_{[k]}$ is the basic processing time of a job scheduled on k-th position.

Most of the models with learning effect covered in literature make the processing time of a job depend *linearly* on a function of either the sum of basic processing times or the number of jobs executed earlier (equivalently, the position of a job in a schedule). That means that in most of classical models with learning effect the execution of two unit jobs takes less time than the execution of one job, which is two units long. However, in many real-life situations the process of learning occurs not only before, but also during the execution of a job.

In this presentation we propose a new model, where the processing time of a job is calculated as a Riemann integral of a given positive and non-increasing function on a certain interval. This model takes into account that the process of learning is continuous.

*bap@amu.edu.pl. Faculty of Mathematics and Computer Science, Adam Mickiewicz University in Poznan, Poland, ul. Umultowska 87, 61-614 Poznan, Poland

2 Continuous learning model

Let J_i be any arbitrary job with the basic processing time of $\bar{p}_i$ and let φ be a positive, non-increasing and Riemann integrable function. The actual processing time of J_i assigned to machine M is equal to

$$p_i = \int_{L(M,S_i)}^{L(M,S_i)+\bar{p}_i} \varphi(s)\,\mathrm{d}s,$$

where S_i is the starting time of the job and $L(M, S_i)$ is the sum of basic processing times of jobs executed on machine M up till the moment S_i.

During the presentation, we will illustrate the proposed model by a number of various examples.

3 Results

For the continuous learning model the following properties are hold.

Property 1 *The general problem of parallel-machine scheduling of jobs with continuous learning effect is NP-hard.*

Property 2 *If we apply continuous learning model to problems of scheduling jobs with unit basic processing times, then the proposed model reduces to classical unit jobs scheduling with position-depenedent processing times. In this case*

$$p_{j,r} = \int_{r-1}^{r} \varphi(s)\,\mathrm{d}s = \bar{\varphi}(r)$$

for some discrete and non-increasing function $\bar{\varphi}$ of r.

The case of scheduling unit and position-dependent jobs with learning effect was analysed in detail in [8].

Property 3 *If $c = \int_0^\infty \varphi(s)\,\mathrm{d}s < \infty$, then for any scheduling problem the optimal C_{max} value is less than or equal to c.*

Before we formulate the next two results, we need to introduce some definitions. We will call a schedule a *continuous schedule*, if no machine is idle until all the jobs assigned to it are executed. Let $\Lambda(T, \varphi)$ be a function of a continuous schedule and a learning function that transforms a feasible and continuous schedule T of jobs with fixed processing times into a feasible schedule within a model of continuous learning. The algorithm of such a transformation will be presented and is a part of [7].

Theorem 4 *Let φ be any positive, non-increasing and Riemann integrable function and let I be an arbitrary instance of the $P|prec|C_{max}$ or the $P|prec, pmtn|C_{max}$ problem. If algorithm A generates an optimal schedule T for I and it is a continous schedule, then $T' = \Lambda(T, \varphi)$ is a corresponding schedule optimal within the continuous learning model.*

Corollary 5 *Let φ be any positive, non-increasing and Riemann integrable function and let I be an arbitrary instance of the $P|pmtn|C_{max}$ problem. As McNaughton's algorithm generates an optimal schedule T for I and it is a continous schedule, the $T' = \Lambda(T, \varphi)$ schedule is a corresponding schedule optimal within the continuous learning model.*

References

[1] Alessandro Agnetis, Jean-Charles Billaut, Stanisław Gawiejnowicz, Dario Pacciarelli, and Ameur Soukhal. *Multiagent Scheduling*. Springer Nature, 2014.

[2] Dirk Biskup. A state-of-the-art review on scheduling with learning effects. *European Journal of Operational Research*, 188(2):315–329, 2008.

[3] Dirk Biskup. Single-machine scheduling with learning considerations. *European Journal of Operational Research*, 115(1):173–178, 1999.

[4] Stanisław Gawiejnowicz. A note on scheduling on a single processor with speed dependent on a number of executed jobs. *Information Processing Letters*, 57(6):297–300, 1996.

[5] Wen-Hung Kuo and Dar-Li Yang. Minimizing the total completion time in a single-machine scheduling problem with a time-dependent learning effect. *European Journal of Operational Research*, 174(2):1184–1190, 2006.

[6] Gur Mosheiov and Jeffrey B Sidney. Scheduling with general job-dependent learning curves. *European Journal of Operational Research*, 147(3):665–670, 2003.

[7] Bartłomiej Przybylski. Parallel-machine scheduling with a new model of continuous learning. *In preparation*.

[8] Bartłomiej Przybylski. Precedence constrained parallel-machine scheduling of position-dependent jobs. *Optimization Letters*, 2016, doi:10.1007/s11590-016-1075-8.

[9] Vitaly Strusevich, Kabir Rustogi. *Scheduling with times-changing effects and rate-modifying activities*. Springer, 2017.

Interactive Decision Support for Multi-Goal Operating Theater Scheduling With Different Planning Horizons

Michael Helmling (Speaker) Sebastian Velten

1 Introduction

We consider scheduling of elective surgeries for a set of operating theaters [1]. Our goal is to interactively support surgery scheduling decisions in the following use cases:

Long-term planning: A patient needs a surgery appointment within a given time window, which typically lies at least a few days in the future.

Next day's schedule: After the set of surgeries of the upcoming day has been fixed, they can be rearranged in order to improve the plan with respect to multiple goals (e.g. distribution of workload to theaters or minimization of (expected) delays).

On-line planning: Using live data on the as-is state of current surgeries, propose (small) reschedulings to prevent overtimes or cancellations.

Because surgery planning decisions are generally driven by several objectives, not all of which can be integrated into a formal model but instead reside in the planner's operational know-how, *interactivity* is a major requirement. The system should propose several alternatives and / or allow to evaluate the impact of modifications to the current schedule. We explicitly do not strive to optimize for one single, definite objective function.

Besides the operating theaters, we take arbitrary limited resources such as surgeons, nurses, and special equipment into account by allowing each surgery to specify any number of resource requirements of the form "need k resources among the set $\{r_1, \ldots, r_n\}$".

Constraint programming (CP) (see e.g. [5]) techniques are employed both for modelling and solving the scheduling problems. For each of the abovementioned use cases, a customized search algorithm operating on a common CP model explores the solution space and collects a set of scheduling proposals which are then returned to the user for interactive decision making.

2 The CP Model

We briefly describe the CP variables and constraints used to model our problem.

{michael.helmling, sebastian.velten}@itwm.fraunhofer.de. Fraunhofer Institute for Industrial Mathematics ITWM, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany.

2.1 Surgeries

A point in time is specified by a pair $t = (d, \tau) \in \mathbb{N} \times 1, \ldots, T$, i.e., the τ-th time slot of day d, where T is the number of slots per day. For a surgery s that can potentially be performed on days $D_s \subset \mathbb{N}$, the following CP variables and constraints are created:

- for each $d \in D_s$, an *optional* interval variable [4] I_s^d with $1 \leq \text{start}(I_s^d) \leq \text{end}(I_s^d) = \text{start}(I_s^d) + \text{duration}(s) \leq T$, and a binary indicator variable p_s^d representing the "performedness" of I_s^d,
- if the surgery must not be performed during an interval $[(d, \tau_1), (d, \tau_2)]$, a *no overlap constraint* of I_s^d with the fixed interval $[\tau_1, \tau_2]$,
- the constraint $\sum_{d \in D_s} p_s^d = 1$.

I_s^d represents the execution time of s on day d, and the constraint enforces that s is performed on exactly one day.

2.2 Resource Requirements

If surgery s requires n resources among a resource set R, the CP model is augmented by the following entities:

- for each $r \in R$, optional interval variables $I_{r,s}^d$ for $d \in D_s$ with the same duration and bounds as the I_s^d above and performedness indicators $p_{r,s}^d$,
- if $r \in R$ is unavailable from (d, τ_1) to (d, τ_2) with $d \in D_s$, a *no overlap constraint* of $I_{r,s}^d$ with the fixed interval $[\tau_1, \tau_2]$,
- for $d \in D_s$, the constraint $\sum_{r \in R} p_{r,s}^d = p_s^d \cdot n$ which ensures that the resource requirements are met,
- a *cover constraint* enforcing that all *performed* intervals among the set $\bigcup_{d \in D_s} I_s^d \cup \bigcup_{d \in D_s} \bigcup_{r \in R} I_{r,s}^d$ start and end at the same time.

Finally, for each resource r, each day d and all surgeries $s \in S$ in which r potentially participates on day d, a *temporal disjunctive constraint* on $\cup_{s \in S} I_{r,s}^d$ forbids that the resource is simultaneously occupied by more than one surgery.

2.3 Planning Objectives

Planning objectives, such as preference for a specific time or specific resources, can be added explicitly to the model by introducing a binary indicator variable and constraining its value to 1 if and only if the respective objective is met. These indicator variables are used by the search as described below.

3 Search Strategies

Our general approach is to mimic the searching procedure of a human planner, while the strength of constraint propagation and reasoning allows to quickly detect infeasible choices and quickly process a huge number of alternatives.

A CP solver explores the search space by a tree of *decisions*, each of which narrows down the domain of one or more variables. It is common to group these decisions into *search phases* that each cover a particular class of decisions, such that one can describe the search more abstractly by a sequence of search phases, each of which internally might consist of multiple individual decisions.

In long-term planning mode, the first search phase sets the day of s by fixing the respective p_s^d to 1. The potential days D_s are sorted in such a way that more preferred days are instantiated first. The next search phase further subdivides the selected day, where, similar to the first phase, preferred time intervals (if any time preference is given) are tested first. After that, the third phase choses the operating room (being the most critical resource). The fourth search phase fixes the exact time of the surgery. In order to avoid gaps in the schedule, only that times are considered that let the surgery immediately follow, or be immediately followed by, either another surgery or a calendar boundary (unless a schedule with gaps is the only feasible option). Finally, the remaining resources are assigned in the fifth phase, once again sorted by descending preference.

The search for the remaining use cases is designed in a similar fashion.

3.1 Selection of Solution

In order to obtain a limited number of structurally distinct proposals, we attach a nested solution count limit to each search phase: the maximum number of solutions per day, per time of day, per surgery room etc. can be limited independently of each other. Additionally, the minimum number of achieved objectives is constantly updated to a configurable fraction of the maximum among the current solutions, which reduces the remaining search space and excludes irrelevant proposals.

4 Implementation Within a Commercial Software Tool

The model and search strategies have been implemented as a component for a commercial health care scheduling software [2] that is currently in pilot phase. It allows the planner to work interactively with the solver to find a schedule that best meets the requirements. For the CP parts, we rely on the generic constraint solver from the *or-tools* suite [3].

References

[1] Brecht Cardoen, Erik Demeulemeester, and Jeroen Beliën. Operating room planning and scheduling: A literature review. *European Journal of Operational Research*, 201(3):921–932, 2010.

[2] Imilia Interactive Mobile Applications GmbH. Timerbee 2.0, 2017.

[3] Google Inc. Google optimization tools (or-tools).

[4] Philippe Laborie and Jerome Rogerie. Reasoning with conditional time-intervals. In *FLAIRS conference*, pages 555–560, 2008.

[5] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier Science.

To Augment or Not to Augment: Solving Unsplittable Flow on a Path by Creating Slack

Fabrizio Grandoni* Tobias Mömke (Speaker) † Andreas Wiese ‡
Hang Zhou§

1 Introduction

In the *Unsplittable Flow on a Path* problem (UFP) we are given an undirected path $G = (V, E)$, with edge capacities $u(e) \in \mathbb{N}$ for $e \in E$. Furthermore, we are given a set T of n tasks. Each task $i \in T$ is specified by a subpath $P(i)$ between the start (i. e., leftmost) vertex $s(i) \in V$ and the end (i. e., rightmost) vertex $t(i) \in V$, a demand $d(i) \in \mathbb{N}$, and a profit (or weight) $w(i) \geq 0$. Our goal is to select a subset $T' \subseteq T$ of tasks of maximum total profit such that for each edge e the total demand of the selected tasks using e does not exceed $u(e)$.

UFP attracted a lot of attention in the last few years in the approximation algorithms community. The problem admits a QPTAS [2, 5] which (in some non-trivial sense) builds up on a QPTAS for the *rooted* case of the problem, where all tasks share a common edge. The existence of a QPTAS gives some evidence that UFP might indeed admit a PTAS: finding it is considered as a challenging open problem in the area.

In terms of polynomial-time approximation, the first non-trivial result was an $O(\log n)$-approximation by Bansal et al. [3]. To achieve that, Bansal et al. reduced the general case to the rooted case of the problem and provided an $O(1)$-approximation for the latter. After an improvement to $(7 + \varepsilon)$-approximation by Bonsma et al. [6], the current best approximation ratio is $2 + \varepsilon$ due to Anagnostopoulos et al. [1].

Traditionally, tasks are classified into large and small tasks. A task i is δ-*large* if there is an edge $e \in P(i)$ such that $d(i) \geq \delta \cdot u(e)$ and δ-*small* otherwise. All previous polynomial time approximation algorithms for the problem and its special cases [7, 8, 9, 1, 6] essentially treat these two groups separately, which inherently lose a factor of 2 in the approximation ratio. It is challenging to handle the two groups in a combined way by a polynomial time algorithm. This can be seen from the fact that the ratio of $2 + \varepsilon$ is the best known polynomial time result even for the special case of uniform edge capacities [4, 7, 11] and for the mentioned rooted case. No better result is known even

*IDSIA, University of Lugano, Switzerland. fabrizio@idsia.ch. Partially supported by the ERC Starting Grant NEWNET 279352 and the SNSF Grant APPROXNET 200021_159697/1.

†Saarland University, Saarland Informatics Campus, Germany. moemke@cs.uni-saarland.de. Funded by Deutsche Forschungsgemeinschaft grant MO 2889/1-1.

‡University of Chile, Santiago de Chile, Chile. This research was carried out at the Max Planck Institute for Informatics, Saarland Informatics Campus, Germany. awiese@mpi-inf.mpg.de.

§Max Planck Institute for Informatics, Saarland Informatics Campus, Germany. hzhou@mpi-inf.mpg.de. Research supported in part by the Lise Meitner Award Fellowship.

under resource augmentation, i. e., if we are allowed to increase the capacity of all edges by a factor $1+\varepsilon$ and the compared optimal solution does not have this privilege.

1.1 Our Results and Techniques

In most prior work on UFP the tasks are classified into large and small tasks as defined above, based on the input alone. The *sparsification lemma* by Batra et al. [5, Lemma 3.1] allows us to deviate from this path. Let OPT be the optimal solution. For each edge e, consider the $1/\varepsilon$ tasks in OPT with largest demands using e. We call those tasks *locally large*. Note that this definition depends on OPT. The sparsification lemma states that for any $\kappa \in \mathbb{N}$ one can *globally* remove a set of tasks with total weight $O(\kappa\varepsilon)\cdot w(\mathrm{OPT})$ such that *for each edge* e at least κ of its locally large tasks are removed. Then, intuitively, in a dynamic programming approach the remaining locally large tasks can be guessed and the gained slack can be used to simplify the computation for the locally small tasks.

In this paper we push this approach further. By using the sparsification lemma we remove $\kappa = 2/\varepsilon^4$ out of the $1/\varepsilon^5$ largest tasks on each edge e. Then each locally small task on an edge has a *very* small demand compared to the gained slack. If a task is locally small on all of its used edges we call it *shrinkable*. Using LP-based arguments (critically using that such tasks are locally *very* small) we reduce the total demand of the shrinkable tasks on each edge by a factor $1-\varepsilon$ while losing only a factor $1+O(\varepsilon)$ in the total profit. Hence, we sacrifice on each edge e some fraction of both the locally large and locally small tasks.

From the point of view of the shrinkable tasks, the available capacity on each edge is by a factor $1+\Theta(\varepsilon)$ larger than the actually needed capacity. In some informal sense, we use this to achieve the following goal. We start with an approximation algorithm in the *resource augmentation* setting, where the computed solution is allowed to violate edge capacities by a factor $1+\varepsilon$ (while comparing the profit with the optimal solution that cannot do this). Then we transform it to an algorithm that up to a factor $1+O(\varepsilon)$ achieves the same approximation ratio *without resource augmentation*.

Typically, resource augmentation helps substantially when designing approximation algorithms. Unfortunately, a resource augmentation PTAS for the general case of UFP is not known. Still, we believe that this is a much easier task than achieving (directly) a PTAS without resource augmentation for the same problem. For instance, the additional capacity can be used to round and discretize the demands of tasks, or to allow errors when estimating the required capacities for subsets of tasks (a common step for small tasks of a given instance). Meanwhile, we achieve PTASs for the following special cases and variants of UFP:

- We design a resource augmentation PTAS for the rooted case of UFP, and from there we derive a PTAS for the setting without resource augmentation. We recall that here all task subpaths share a common edge, and the previous best approximation algorithm for this case was the $(2+\varepsilon)$-approximation for the general case. As an extension, we also obtain a PTAS for the case that all tasks use at least one out of a subset of $O(1)$ edges. In turn, this PTAS can be used to obtain a PTAS for the *non-containment* case of the problem, where no two tasks i, j satisfy $P(i) \subseteq P(j)$.
- We consider an unlimited-supply case of UFP (unl-UFP), where we are allowed to include multiple copies of the same task in the solution. This is motivated by scenarios where each task models a type of client, and there are many clients of

each type (enough to saturate the edge capacities). To the best of our knowledge, this unlimited-supply notion was not addressed before in the framework of UFP, while it is common in some related pricing problems [10]. We first design a resource augmentation PTAS for unl-UFP via a structural lemma that shows that there are near-optimal solutions in which the small tasks are chosen in *bundles*. Using the same technique as before, we then obtain a PTAS without resource augmentation.

- We obtain a PTAS for the special case of UFP where the profit of each task i is proportional to its *area* $d(i) \cdot |P(i)|$ (area-UFP). This special case is also well-motivated: for example, considering $P(i)$ as a time interval as mentioned before, $d(i) \cdot |P(i)|$ can be interpreted as the total volume of the considered resource used to process task i (hence it makes sense to charge i by that amount). In this case the PTAS with resource augmentation is already quite involved and requires some careful charging arguments. We obtain a PTAS without resource augmentation using the same technique.

References

[1] Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing 2+ε approximation for unsplittable flow on a path. In *SODA*, pages 26–41, 2014.

[2] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729. ACM, 2006.

[3] N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA*, pages 702–709, 2009.

[4] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *STOC*, pages 735–744, 2000.

[5] Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *SODA*, pages 47–58, 2015.

[6] Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM Journal on Computing*, 43:767–799, 2014.

[7] Gruia Călinescu, Amit Chakrabarti, Howard J. Karloff, and Yuval Rabani. An improved approximation algorithm for resource allocation. *ACM Transactions on Algorithms*, 7:48:1–48:7, 2011.

[8] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47:53–78, 2007.

[9] C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3, 2007.

[10] Fabrizio Grandoni and Thomas Rothvoß. Pricing on paths: A PTAS for the highway problem. In *SODA*, pages 675–684. SIAM, 2011.

[11] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *SODA*, pages 879–888, 2000.

A priori TSP in the Scenario Model

Martijn van Ee (Speaker) [*] Leo van Iersel [†] Teun Janssen [‡]
René Sitters [§]

1 Introduction

In *a priori* routing, we extend our classical routing problems to the case that the set of clients is uncertain or changes regularly. Because reoptimizing might be inconvenient or impossible, we want to find a single tour. Given a tour and a set of clients, the active set, we shortcut the tour to the active set and we want to minimize the expected length of the resulting tour.

Formally, in the *a priori* traveling salesman problem in the scenario model, we are given a complete weighted graph $G = (V, E)$ and a set of scenarios $\mathcal{S}$ with $S_1, \ldots, S_m \subseteq V$. Scenario S_j has probability p_j of being the active set, where $\sum_j p_j = 1$. We begin by finding an ordering on V, called the first-stage tour. When an active set is released, the second-stage tour is obtained by shortcutting the first-stage tour on the vertices of the active set. The goal is to find a first-stage tour that minimizes the expected length of the second-stage tour.

This problem has, for example, a direct application to the photo-lithography processes used in semi-conductor manufacturing to transfer the geometric pattern of a chip onto a wafer [1]. This is done by putting UV light through a photomask on a photoresistant layer on top of the wafer. The entire wafer is not exposed at once, but one square at a time. If certain parts of the square do not need to be exposed, blades are moved in to block the UV light. Moving the blades is a time-consuming, and hence costly, process. Since it often influences the total processing time of a wafer in the lithography machine, minimizing the distance reduces the processing time. The blading positions are defined in a file. The blading positions are obtained from this file by reading it from top to bottom and the positions are used by the machine in order of appearance. A product will visit the photolithography machine multiple times during its fabrication. Every time it will use the same file that defines its blading positions, but it will not use all blading positions defined in the file in every visit. For each visit, there is a given subset of the blading positions that has to be used. Hence minimizing the movement of

[*] m.van.ee@vu.nl. Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands.

[†] l.j.j.vaniersel@tudelft.nl. Department of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands.

[‡] t.m.l.janssen@tudelft.nl. Department of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands.

[§] r.a.sitters@vu.nl. Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands.

the blades comes down to finding an ordering of the blading positions such that the sum over all visits of the total distance between the blading-positions is minimized.

A priori TSP has already been considered in the independent decision and black-box model. In the independent decision model, vertex i is active with probability p_i, independent of the other vertices. Shmoys and Talwar [7] showed that a sample-and-augment approach gives a randomized 4-approximation, which can be derandomized to an 8-approximation algorithm. In the black-box model, we have no knowledge on the probability distribution over the vertices, but we are able to sample from it, i.e. to query the probability of any subset of the vertices. Schalekamp and Shmoys [6] showed that one can obtain a randomized $O(\log n)$-approximation even without sampling. It was shown by [4] that there is an $\Omega(\log n)$ lower bound for deterministic algorithms on general metrics.

The former results give us the first results for *a priori* TSP in the scenario model. First of all, we inherit the randomized $O(\log n)$-approximation. Secondly, we know that a deterministic algorithm that does not use the information given in the scenarios will not achieve an approximation guarantee better than $O(\log n)$. The main question is whether we can use the scenarios to improve upon the $O(\log n)$ upper bound and which restrictions we can put on the scenarios in order to obtain constant-factor approximability. Here, we show that the problem is already difficult when the scenarios are small. We also show positive results when there is a constant number of scenarios, when the scenarios are big and when the scenarios are nested. An extended version of this abstract appeared in [2].

2 Small scenarios

We show that *a priori* TSP in the scenario model is already hard when $|S_j| \leq 4$ for all j by reducing from Max Cut. Note that the problem is trivial when $|S_j| \leq 3$ for all j.

Theorem 1 *A priori TSP is NP-complete when $|S_j| \leq 4$ for all j.*

We can use the same reduction to show inapproximability results. We can strenghten these by assuming the Unique Games Conjecture (UGC).

Theorem 2 *There is no 1.0117-approximation for a priori TSP with $|S_j| \leq 4$, unless P=NP, and no 1.0242-approximation under UGC.*

By using strong results on the inapproximability of Permutation CSPs [5], we can improve our results for larger scenarios.

Theorem 3 *Under UGC, there is no α-approximation for a priori TSP with*

(a) $\alpha < \frac{10}{9}$ *when* $|S_j| \leq 6$,

(b) $\alpha < \frac{7}{6}$ *when* $|S_j| \leq 8$,

(c) $\alpha < \frac{71}{60}$ *when* $|S_j| \leq 10$.

A related question is whether there is a tour such that if we shortcut on the vertices of a scenario, we get the optimal solution for that scenario. This problem is known as the Master Tour problem with scenarios. It turns out that this problem is Δ_2^p-complete [3]. By adjusting the proof of Theorem 1, we can prove that the master tour problem with scenarios is NP-complete when $|S_j| \leq 5$. This is done by reducing from Set Splitting instead of Max Cut. The master tour problem with scenarios is still open for $|S_j| \leq 4$.

3 Other results

We were able to prove constant-factor approximability for certain special cases. The algorithm that solves (approximates) TSP for each scenario and than concatenates the m subtours is a $(2m-1)$-approximation. One might either concatenate the tours in non-increasing order of their lenghts or in non-decreasing order of their probabilities.

Theorem 4 *There is a $(2m-1)$-approximation for a priori TSP in the scenario model, where $m \geq 2$ is the number of scenarios.*

The optimal solution on the entire set of vertices is a good approximation if the scenarios are large, i.e. each scenario contains all but a constant number of the vertices.

Theorem 5 *The optimal solution on V is a $(1+\frac{c}{2})$-approximation for a priori TSP with $|S_i| \geq n-c$, where $1 \leq c \leq \frac{n}{2}$.*

Finally, by carefully choosing a collection of subtours to concatenate, we can prove a 9-approximation if the scenarios are nested.

Theorem 6 *There is a 9-approximation for nested scenarios, i.e. when $S_1 \subseteq \ldots \subseteq S_m$.*

References

[1] Jan Driessen and Teun Janssen. Minimizing the blading in lithography machines: an application of the *a priori* TSP problem. Unpublished manuscript, 2016.

[2] Martijn van Ee, Leo van Iersel, Teun Janssen, and René Sitters. *A priori* TSP in the scenario model. In *Approximation and Online Algorithms: 14th International Workshop, Revised Selected Papers*, pages 183–196. Springer, 2017.

[3] Martijn van Ee and René Sitters. On the complexity of master problems. In *Proceedings of the 40th Symposium on Mathematical Foundations of Computer Science*, pages 567–576. Springer, 2015.

[4] Igor Gorodezky, Robert D. Kleinberg, David B. Shmoys, and Gwen Spencer. Improved lower bounds for the universal and *a priori* TSP. In *Proceedings of the 13th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 178–191. Springer, 2010.

[5] Venkatesan Guruswami, Johan Håstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM Journal on Computing*, 40(3):878–914, 2011.

[6] Frans Schalekamp and David B. Shmoys. Algorithms for the universal and *a priori* TSP. *Operations Research Letters*, 36(1):1–3, 2008.

[7] 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.

Container Dispatching and Conflict-Free Yard Crane Routing in an Automated Container Terminal

Dirk Briskorn (Speaker) * Jenny Nossack † Erwin Pesch ‡

1 Introduction

We consider a crossover gantry crane setting managing a single container block in a seaport container terminal. Two cranes spanning the width of the whole block move on tracks alongside the block. Since they differ in height and width they can pass each other unless the taller crane has its spreader lowered.

We consider a 1-dimensional model of the block only, see also [1], focussing on moves of cranes affecting the whole gantry since these are most time-consuming and can be conducted in parallel to moving the trolley along the gantry. The spreader can be lowered only when gantry and trolley do not move and, therefore, can be represented by pickup times and dropoff times for transport requests. We distinguish between three types of transportation requests, namely inbound requests, outbound requests, and so-called housekeeping requests, see [4]. Origin and destination of a request are typically determined beforehand by a pre-executed stacking algorithm, see [3].

In our problem setting, we address the set of dual cranes $K := \{1, 2\}$ where 1 represents the taller crane and 2 the smaller crane. The block bays are denoted by $Q := \{0, \ldots, l\}$ and the set of transportation requests by $R = \{1, \ldots, n\}$. The origin location of each request $i \in R$ is denoted by $O_i \in Q$ and the destination location by $D_i \in Q$. The service times at the origin and destination are represented by $s_i^O \in \mathbb{N}_0$ and $s_i^D \in \mathbb{N}_0$ for each request $i \in R$, respectively. The minimum travel time from the origin O_i to its destination D_i of request i is represented by $t_{i,i}$ and is set to $t_{i,i} := |O_i - D_i|$. Similarly, the minimum travel time from the destination D_i of request i to the origin O_j of request j is denoted by $t_{i,j}$ and is defined by $t_{i,j} := |D_i - O_j|$. Furthermore, parameter $S^k \in Q$ denotes the initial location of crane $k \in K$.

The CCSP is to assign requests to cranes such that each request is dispatched to exactly one crane. For a given dispatch we, furthermore, consider a permutation of the associated requests. We refer to permutations of requests for both cranes as a *work plan*. For a given work plan, a *schedule* for a crane is the position of the crane over time such that the position is a continuous function (in time) with slope in $-1, 0, 1$ and the crane follows the work plan, i.e., it visits the origin and destination locations in the

*briskorn@uni-wuppertal.de. Department of Production and Logistics, University of Wuppertal, Wuppertal, Germany.

†Jenny.Nossack@hhl.de. Center for Advanced Studies in Management , HHL Leipzig Graduate School of Management, Leipzig, Germany.

‡erwin.pesch@uni-siegen.de. Department of Management Information Science, University of Siegen, Siegen, Germany.

order implied by the work plan and is present in the respective locations long enough to conduct the corresponding operations. Note that the bounds on the slope of the function reflect the maximum speed of cranes of at most one bay per time unit. A *conflict-free pair of schedules* guarantees that at each point of time when crane 1 is operating in bay b, crane 2 is located in bay b' with $b' \leq b-1$ or $b' \geq b+1$. A *feasible solution* gives a work plan and a conflict-free pair of schedules. The makespan of a feasible solution gives the point of time the last container is served at its destination location. The CCSP is to find a feasible solution having minimum makespan.

2 Computational complexity and solution approach

NP-hardness of the CCSP is rather obvious due to, for example, individual service times of requests and the similarity of CCSP and problem $P2||C_{\max}$ to schedule jobs on two parallel machines. However, we can show a stronger result.

Theorem 1 *The decision version of CCSP is strongly NP-complete even if* $l \leq 5$.

We only sketch the proof here. It is by reduction from 3-PARTITION. The idea is to design an instance such that the taller crane is forced to conduct very time consuming pickup operations in fixed time intervals in bay b. The smaller crane, therefore, can pass b only in narrow time intervals and has to have a partition of its requests on both sides of b. The partition implies a yes-certificate to the instance of 3-PARTITION if and only if a given makespan can be reached. Membership in NP on the other hand can be shown using the result from [2] enabling us to employ sequences of requests assigned to both cranes as certifier. The graph-based approach developed in [2], then, can be used in order to determine the optimum right of way in case of conflicting requests in strongly polynomial time.

In order to tackle CCSP we develop a decomposition approach making use of the efficient approach in [2], as well. The basic idea of this decomposition approach is to partition a problem into a master and a subproblem while connecting them via logic-based Benders constraints. Based on cutting plane methods, it then approximates the feasible set of the master problem by only a subset of the constraints. If the resulting solution is infeasible, additional constraints are added. The subproblem operates as separation problem and either proves global optimality of a solution or detects violated constraints reflecting a lower bound on the makespan imposed by the optimum positions over time for given sequence of requests.

We base our master problem on a vehicle routing formulation (refer, e.g., to [5]) and introduce a binary decision variable y_{ij}^k which denotes whether or not request j is conducted after request i by crane $k \in K$. Furthermore, variable $W \in \mathbb{R}_0^+$ identifies the makespan. For simplicity, we treat the initial states of crane 1 and 2 as requests 0_1 and 0_2, respectively, and thus define $t_{0_k,i} := \left|S^k - O_i\right|$, $t_{0_k,0_k} := 0$, $s_{0_k}^O := 0$ and $s_{0_k}^D := 0$ for all $k \in K$ and $i \in R$. Note that we also require a dummy request (denoted by $n+1$) to simulate that a crane ends its route at the destination location of its last request.

$$\min W \tag{1}$$

$$\sum_{i \in R \cup \{n+1\}} y_{0_k,i}^k = 1 \qquad \forall k \in K \tag{2}$$

$$\sum_{k \in K} \sum_{\substack{i \in R \cup \{0_k\} \\ i \neq j}} y_{i,j}^k = 1 \qquad \forall j \in R \tag{3}$$

$$\sum_{\substack{j \in R \cup \{n+1\} \\ i \neq j}} y_{i,j}^k - \sum_{\substack{j \in R \cup \{0_k\} \\ i \neq j}} y_{j,i}^k = 0 \qquad \forall i \in R; k \in K \tag{4}$$

$$\sum_{\substack{i,j \in S \\ i \neq j}} y_{i,j}^k \leq |S| - 1 \quad \forall S \subseteq R \cup \{0_k\}; k \in K \tag{5}$$

$$\sum_{\substack{i \in R \cup \{0_k\}, j \in R \\ i \neq j}} y_{i,j}^k \cdot (t_{i,i} + t_{i,j} + s_i^O + s_i^D) \leq W \qquad \forall k \in K \tag{6}$$

$$\hat{W}^h (1 - \sum_{k \in K} \sum_{i,j \in J_k^h} (1 - y_{i,j}^k)) \leq W \qquad \forall h \in H \tag{7}$$

$$y_{i,j}^k \in \{0,1\} \qquad \forall i \in R \cup \{0_k\}, \; j \in R \cup \{n+1\}, i \neq j; k \in K \tag{8}$$

$$W \in \mathbb{R}_0^+ \qquad \forall k \in K \tag{9}$$

The objective function (1) minimizes the makespan. Equations (2) ensure that both cranes start at their initial location. Constraints (3) to (5) ensure feasible sequences of requests. Constraints (6) guarantee that the makespan equals the maximum finishing time of both cranes. Constraints (7) are logic-based Benders constraints that are obtained by solving the subproblem and bound the makespan from below. Finally, constraints (8) and (9) define the domains of the decision variables. We obtain encouraging results using the branch-and-cut approach as an exact method and a heuristic, respecitvely, tackling instances with up to 40 transport requests.

References

[1] N. Boysen and D. Briskorn and F. Meisel (2017). *A generalized classification scheme for crane scheduling with interference*. European Journal of Operational Research, Vol. 258:343–357, 2017.

[2] D. Briskorn and P. Angeloudis (2016). *Scheduling co-operating stacking cranes with predetermined container sequences*. Discrete Applied Mathematics, Vol. 201:70–85, 2016.

[3] U. Dorndorf and F. Schneider (2010). *Scheduling automated triple cross-over stacking cranes in a container yard*. OR Spectrum, 32:617–632, 2010

[4] A. Ehleiter and F. Jaehn (2016). *Housekeeping: Foresightful container repositioning*. International Journal of Production Economics, 179:203–211, 2016

[5] P. Toth and D. Vigo (2002). The Vehicle Routing Problem. SIAM, Philadelphia, 2002.

A FPTAS of Minimizing Total Weighted Completion Time on Single Machine with Position Constraint

Gruia Călinescu * Florian Jaehn † Minming Li (Speaker) ‡ Kai Wang §

1 Introduction

In general, a major challenge of scheduling problems is the determination of a job sequence for each machine involved. Especially in non-preemptive one machine settings without idle times, this is usually the only task to be performed. In this context, scheduling problems appear without restrictions on this sequence (e.g. $1||\sum T_j$) and with restrictions on the sequence (e.g. $1|r_j, prec|\sum C_j$). Restrictions on the sequence are commonly either time dependent or linked to job pairs. Examples for time dependent restrictions are release dates, deadlines, or time dependent maintenance activities. Precedence constraints are a typical restriction based on job pairs. In this paper, we elaborate on a different restriction based on the position of a job in the sequence. To be more precise, we force one job to have a fixed position within the sequence of jobs.

The practical and theoretical motivation for such a scheduling problem is twofold. Firstly, such a job that has a fixed position in the sequence could be considered as a maintenance operation. Classically, maintenance is also considered to be time dependent, e.g. by modeling predetermined machine unavailability intervals ([1, 2, 3]), by allowing a maximum time between two maintenances, which is often referred to as "tool changes" ([4, 5]), or maintenances may be inserted arbitrarily in order to reduce the processing times of following jobs ([6, 7]). However, just lately, position dependent maintenance operations have been introduced by [8]. Amongst others, they motivate position dependent maintenance activities with wear and tear of jet engines or aircraft wheels, which is rather caused by the number of flights (because of the climb flight and thrust reversal for the engines) than by the length of the flight. So the problem considered here can be seen as the special case in which exactly one position dependent maintenance activity is necessary.

Secondly, our problem is a special case of scheduling with nonnegative inventory constraints, as it was introduced by [9]. Here, each job either delivers or removes items of a homogeneous good to or from an inventory. Jobs that remove items can only be processed if the required number of items is available, i.e. only if the inventory level does

*calinescu@iit.edu. Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA.

†florian.jaehn@wiwi.uni-augsburg.de. Institute of Sustainable Operations and Logistics at the University of Augsburg, Universitaetsstr. 16, D-86159 Augsburg, Germany.

‡minming.li@cityu.edu.hk. Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong SAR, China.

§kai.wang@my.cityu.edu.hk. Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong SAR, China.

not become negative. This problem relates to ours, in which a job is fixed to position k, as follows. The job fixed on position k can be considered as the only job removing items from the inventory, and $k-1$ jobs are required to deliver items before this job can be scheduled. If the parameter settings of the fixed job are chosen such that this job is to be scheduled as early as possible, it is forced to be on position k. Analogously, the fixed job can be modeled as the only one delivering to the inventory so that it must be scheduled the latest on position k. Parameter settings then need to ensure that it is not scheduled earlier.

In this paper we continue the work of [9] on problem $1|inv|\sum w_jC_j$.

Formulation The instance of the problem is a set of n jobs $J = \{1, 2, ..., n\}$, a specified job $c \in J$ and an integer $k \in [1, n]$. Each job $j \in J$ is defined by its weight w_j and its processing time s_j. The goal is to find a schedule that minimizes total weighted completion time on single machine such that there are exactly $k-1$ jobs scheduled before job c, where k is part of the input.

In classical *Smith's Rule* [10], jobs are executed in non-decreasing order of the ratios w_j/s_j. And Smith's Rule has been proven to be optimal when there is no position constraint of job c.

Theorem 1 *In the optimal solution, jobs that are scheduled before (or after) job c must follow* Smith's order.

Approach & Result So far, we were not able to detect any complexity result of this problem. We give a FPTAS of this problem, however it still remains open that whether this problem is NP-hard or not.

To tackle this problem, we first give pseudo-polynomial dynamic programming algorithms to solve this problem, which is polynomial on job processing time or job weight, respectively. Given ϵ as the parameter of the FPTAS, we round the jobs to integer values according to ϵ such that the job processing time (resp. weight) is polynomial on n and $1/\epsilon$. Therefore, the dynamic programming has polynomial running time, and the major issue is to analyze the performance of the dynamic programming. We discover that, a single dynamic programming may not have a good performance. Therefore in the FPTAS algorithm we construct a set of dynamic programmings based on intermediate result and shows that the algorithm is $(1 + 2\epsilon)$-approximation, with time complexity $O(n^{13}/\epsilon^4)$.

2 PTAS Algorithm

Dynamic Programming with Side Constraints Given integer k, job $c \in J$ and sets of jobs J, $H \subset J \setminus \{c\}$, $B \subset J \setminus \{c\}$ such that $H \cap B = \emptyset$, we propose a pseudo-polynomial dynamic programming to find the optimal schedule such that:

1. jobs from H are scheduled before job c, and
2. jobs from B are scheduled after job c, and
3. there are exactly $k-1$ jobs scheduled before job c.

Jobs $H \cup B$ are said to be *assigned* and let $U = J \setminus (B \cup H \cup \{c\})$ be the *unassigned* jobs.

Rounding on Job Processing Time Given ϵ, let $\lambda := poly(n, 1/\epsilon)$ be a variable polynomial on n and $1/\epsilon$, we round the job processing time by a factor λ, $\forall j \in J, s'_j = \lceil s_j \cdot \frac{\lambda}{s_{max}} \rceil$, where s_{max} varies during the iteration of the algorithm.

Algorithm The algorithm has many rounds, in each round we fix the position of a job accordingly and start over (solve the problem without this job accordingly) unless we reach the termination of the algorithm. In each iteration of current round, we have some *assigned* jobs and *unassigned* jobs. We keep assigning the unassigned jobs to be before or after job c until we reach the termination of current round. In each iteration, we take the best of the solution from the dynamic programmings.

References

[1] Lee, Chung-Yee(1996). *Machine scheduling with an availability constraint.* Journal of global optimization 9.3-4: 395-416.

[2] Błażewicz, Jacek and DellOlmo, Paolo and Drozdowski, Maciej and Maczka, Przemysław (2003). *Scheduling multiprocessor tasks on parallel processors with limited availability.* European Journal of Operational Research 149.2: 377–389.

[3] Low, Chinyao and Ji, Min and Hsu, Chou-Jung and Su, Chwen-Tzeng (2010). *Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance.* Applied Mathematical Modelling 34.2: 334–342.

[4] Chen, Jen-Shiang (2008). *Optimization models for the tool change scheduling problem.* Omega 36.5: 888–894.

[5] Costa, A and Cappadonna, FA and Fichera, S (2016). *Minimizing the total completion time on a parallel machine system with tool changes.* Computers & Industrial Engineering 91: 290–301.

[6] Kubzin, Mikhail A and Strusevich, Vitaly A (2006). *Planning machine maintenance in two-machine shop scheduling.* Operations Research 54.4: 789–800.

[7] Rustogi, Kabir and Strusevich, Vitaly A (2012). *Simple matching vs linear assignment in scheduling models with positional effects: A critical review.* European Journal of Operational Research 222.3: 393–407.

[8] Drozdowski, M and Jaehn, F and Paszkowski, R (2016). *Scheduling position dependent maintenance operations.* Submitted.

[9] Briskorn, Dirk and Choi, Byung-Cheon and Lee, Kangbok and Leung, Joseph and Pinedo, Michael (2010). *Complexity of single machine scheduling subject to nonnegative inventory constraints.* European Journal of Operational Research 207.2: 605–619.

[10] Smith, Wayne E (1956). *Various optimizers for single-stage production.* Naval Research Logistics Quarterly 3.1-2: 59–66

Linear-Time Approximation for Minimum Subset Sum and Subset Sum

Liliana Grigoriu *

1 Introduction

The minimum subset sum problem and the subset sum problem appear often as parts of solutions for scheduling problems, see for example [4]. We present a family of approximation algorithms for minimum subset sum with a worst-case approximation ratio of $1+1/k$ and which run in linear time assuming that k is constant. We also present a family of linear-time approximation algorithms for subset sum with worst-case approximation factors of $1-1/(k+1)$ assuming that k is constant. The algorithms use approaches from and improve upon previous literature, where a linear-time 4/5 approximation algorithm for subset sum and a linear-time 5/4 approximation algorithm for minimum subset sum have been provided by Kellerer et al. [7] and by Grigoriu and Briskorn [4] respectively. The maximum number of times a linear-time procedure could be called within the algorithms, which depends on k, is determined computationally for each of the two problems for sample values of k. For example, for $k = 10$, these numbers are 137 for subset sum and 171 for minimum subset sum, and for $k = 30$ they are 28627 and respectively 31023. A very loose bound for the time complexity is, for both algorithms, $O(n * k^k)$, and it can be improved to other loose bounds such as $O(n * (k-1)^{\lfloor k/2 \rfloor})$. The simplicity of the algorithms allows for fast implementation, and they can be parallelized.

In [7] two linear-time 3/4 and 4/5 approximation algorithms for subset sum are presented. Several FPTAS are known for this problem (see [6] for an overview), and recent exact algorithms for it were presented in [1] and [8] . For the minimum subset sum problem, a quadratic approximation algorithm with a worst-case approximation ratio of 5/4 has been proposed in [5], and a linear-time 5/4 approximation algorithm has been presented in [4], where it was used to solve a related scheduling problem. Also, FPTAS for more general problems than minimum subset sum have been developed, for example in [2]. In this work we use some ideas that were also used in [7] and [4]. Next, we outline the ideas used to build the algorithms for minimum subset sum.

2 Approximation algorithms for minimum subset sum

An detailed version of the results we present is given in [3]. The minimum subset sum problem asks, given a multiset J of positive integers and a target sum S which subset of J has elements that add up to a sum that is at least S and as close as possible to S. To

*liliana.grigoriu@uni-siegen.de. Zentrum für Informations- und Medientechnologie and Fakultät für Wirtschaftswissenschaften. Wirtschaftsinformatik und Wirtschaftsrecht, Universität Siegen.

obtain an approximation algorithm that solves the minimum subset sum problem with a worst-case approximation factor of $(k+1)/k$, we first divide the set J into disjoint subsets as follows:

$$J_i = \{j \in J | (i-1)\frac{S}{k} \leq j < \frac{iS}{k}\} \text{ and } J_l = \{j \in J | j \geq S\}.$$

The main algorithm, which has a worst-case approximation factor of $(k+1)/k$, uses two subroutines. The subroutine $AddJ1(K)$ adds, if necessary, elements from (the globally visible set) J_1 to a set of jobs $K \subseteq J \setminus J_1$ with $S - \sum_{j \in J_1} j \leq \sum_{j \in K} j \leq \frac{k+1}{k}S$ until the sum of the elements in K is at least S. Then, $AddJ1(K)$ returns the resulting set K, which has the property that the sum of its elements is at least S and at most $\frac{k+1}{k}S$, as all elements of J_1 are less than $\frac{1}{k}S$. The second subroutine $Candidate(K, Kbest)$ updates the best found solution $Kbest$ with a newly found solution K if K is better. Here, we assume that the procedure can change $Kbest$. We denote with $min_i(J_h)$ and with $max_i(J_h)$ the sets of elements with the i smallest and respectively with the i largest values from J_h (with ties broken arbitrarily). The definition assumes that there are at least i elements in J_h. Any solution that contains an element of J_l is not better than the solution $min_1(J_l)$, which our algorithm will check separately. Any other solution contains a number n_i of elements from J_i for $i \in \{2, 3, \ldots, k\}$, and possibly elements from J_1. We call a tuple $(n_2, \ldots, n_k)$ configuration of all subsets of $J \setminus J_l$ that contain exactly n_i of elements from J_i for all $i \in I$, where $I = \{2, 3, \ldots, k\}$. Our algorithms build sets Q of elements from $J \setminus J_1$ which fulfill one of the following conditions:

1. it can be proved that they are a $\frac{k+1}{k}$ approximation for the problem instance
2. they can be the optimal solution
3. $\sum_{i \in Q} i < S$ and $\sum_{i \in Q} i + \sum_{i \in J_1} i \geq S$. Here, a solution can be obtained by adding elements from J_1 to Q until the sum of elements in Q reaches or exceeds S, at which point we have $S \leq \sum_{i \in Q} \leq \frac{k+1}{k}S$, which implies that Q is a $\frac{k+1}{k}$ approximation of an optimal solution as desired.

We consider all configurations an optimal solution or a $(k+1)/k$ approximation thereof can have, which are described as follows[3]. The item with the smallest value from J_l is considered as a candidate solution. If, in addition to that, all *configurations of interest* $(n_2, \ldots, n_k)$, that is, those which fulfill all of the following properties, are checked, either an optimal solution or a $(k+1)/k$ approximation thereof is found:

1. (a) $\sum_{i \in I} i \cdot n_i + \sum_{j \in J_1} j \geq S$. In addition, we must have
   (b) $\sum_{i \in I} \sum_{j \in max_{n_i} J_i} j + \sum_{j \in J_1} j \geq S$.
2. $\sum_{i \in I} (i-1)n_i < k + \min\{q | n_q > 0\} - 1$.

Here, checking a configuration means that relevant solutions that correspond to these configurations are found and returned if they are known to be a $(k+1)/k$ approximation or stored as a candidate if they may be the optimal solution.

We next present the procedure CheckConfiguration that considers each configuration of interest $T = (n_2, \ldots, n_k)$. We assume that k, the number $sumJ1 = \sum_{j \in J_1} j$, the sets $J_1, J_2, \ldots J_k$ and the current best candidate solution $Kbest$ are globally visible and can be accessed and changed by the procedure:
CheckConfiguration(T)

(0) Let $I = \{2, 3, \ldots, k\}$, let $Q' = \cup_{i \in I} max_{n_i} J_i$ and $Q = \cup_{i \in I} min_{n_i} J_i$; (if for some $i \in I$, J_i does not contain n_i elements return $\emptyset$;)
(1) If $(\sum_{j \in Q'} j + sumJ1 \geq S)\{$
(2) If $(\sum_{j \in Q} j \geq S)\{$
(a) candidate (Q, Kbest);
(b) If $(\sum_{j \in Q} j \leq \frac{k+1}{k} S)$ return Q; // this step is optional
(c) return $\emptyset$;
}
(3) Else {
(a) If $\sum_{j \in Q} \geq S - SumJ1$, return $AddJ1(Q)$;
(b) For all $i \in I$ do: Iteratively replace in Q elements of $min_{n_i} J_i$ with elements of $max_{n_i} J_i$ until the sum of elements in Q reaches or exceeds $S - SumJ1$, in which case return $AddJ1(Q)$, or until all elements from $min_{n_i} J_i$ are replaced in Q.
}
}
(4) return $\emptyset$;
The main algorithm generates all configurations of interest and calls CheckConfiguration, which runs in linear time, for each one of them, in the end resulting in a 1+1/k approximation or in an optimal solution.

References

[1] K. Bringman(2017) *Improved pseudopolynomial time algorithms for Subset Sum*, Proceedings of the 28th ACM-SIAM Symposium on discrete Algorithms (SODA'17)

[2] G.V. Gens and E.V. Levner (1981). *Fast approximation algorithm for job sequencing with deadlines*, Discrete Applied Mathematics, Vol. 3, pp. 313 - 318

[3] L. Grigoriu. *Linear-time approximation for minimum subset sum and subset sum* e-print on Optimization Online, http://www.optimization-online.org/DB_FILE/2016/11/5708.pdf

[4] L. Grigoriu and D. Briskorn (2016). *Scheduling jobs and maintenance activities subject to job-dependent machine deteriorations*, Journal of Scheduling, doi: 10.1007/s10951-016-0502-0

[5] M.M. Güntzer, M.M. and D. Jungnickel (2000). *Approximate minimization algorithms for the 0/1 Knapsack and Subset-Sum Problem*, Operations Research Letters, Vol. 26, pp. 55-66

[6] H. Kellerer, U. Pferschy and D. Pisinger (2004). *Knapsack Problems.* Springer, Berlin

[7] H. Kellerer, R. Mansini and M.G. Speranza (2000). *Two linear approximation algorithms for the subset-sum problem*, European Journal of Operational Research, Vol. 120, pp. 289 - 296

[8] K. Koiliaris, C. Xu (2017) *A Faster Pseudopolynomial Time Algorithm for Subset Sum*, Proceedings of the 28th ACM-SIAM Symposium on discrete Algorithms (SODA'17), pp. 1062-1072

Scheduling Recurring and Optional Activities for Radiotherapy Considering Stable Treatment Starting Times

Petra Vogl (Speaker) * Roland Braune * Karl F. Doerner *

1 Introduction

We consider the scheduling of recurring treatment activities for patients diagnosed with cancer in a radiotherapy treatment facility. A radiation therapy consists of a pre-treatment phase and the treatment phase itself, which in turn consists of multiple daily treatment activities (DTs) [2]. The treatments are performed on separate days and are followed by various control examinations.

Scheduling radiotherapy appointments consists of (1) an assignment problem defining the days on which treatments for a specific patient should take place and fixing the subsequent control examinations and (2) a scheduling problem in which the exact ("to-the-minute") starting times of the activities are determined. The basis of our formulation is the classic disjunctive job shop MIP model. The activity starting time of activity i of patient p is then described by integer decision variable s_{pi}. Additionally, we incorporate day-indexed binary decision variables dt_{pid} in our model. dt_{pid} equals 1 iff $\underline{dw_d} \leq s_{pi} \leq \overline{dw_d}$ with $\underline{dw_d}$ and $\overline{dw_d}$ denoting the start and end time of day d, respectively.

Assignment For each patient p, a release time r_p and a deadline d_p for his first daily treatment (FDT) is given. Additionally, for medical reasons, each patient has to receive a minimum of four treatments within five days. This constraint significantly raises the problem complexity: While for a patient with five planned DTs we do have 5 possible assignments, there are 657 possibilities of scheduling DTs to days for a patient with 20 irradiation appointments. The permutations of feasible assignments of DTs to days for one patient with n_p^{DT} being the known number of treatments to be scheduled is defined by Eq. 1, with h^{max} the maximum number of breaks allowed within a feasible DT to day assignment:

$$(d_p - r_p) \times \sum_{r=0}^{h^{max}} \binom{n_p^{DT} - 3 \cdot r + 2}{r} \tag{1}$$

After each DT activity, so called optional activities appear within the patient-specific "activity chain". For example, patients need to regularly see their assigned radio-oncologist for a weekly control examination (WCE) and some patients additionally need to attend imaging appointments directly after irradiation in order to evaluate the accuracy of treatment (PET-CT, positron emission tomography - computer tomography). It is then for the optimization algorithm to decide, which of these optional activities take place and which ones are omitted. We have to assure, however, that both categories

*Department of Business Administration, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria. {petra.vogl,roland.braune,karl.doerner}@univie.ac.at.

of optional activities (i.e. WCE, PET-CT) are scheduled *at least once* within every five consecutive days during the treatment phase of a patient.

Scheduling The activities within the activity chain of each patient are tied together using minimum and maximum finish-start relations. E.g. the imaging activity may only start maximum 15 minutes after the irradiation has finished, otherwise it would not give the expected results. If, however, one optional activity is omitted, the time window of the next activity to take place depends on the starting time of the previous activity. Figure 1 visualizes this connections between the activities. The optional activities are marked by dashed lines. While scheduling on alternative process paths has already been studied in the literature, to the best of our knowledge, pure optional activities are a new field of research.

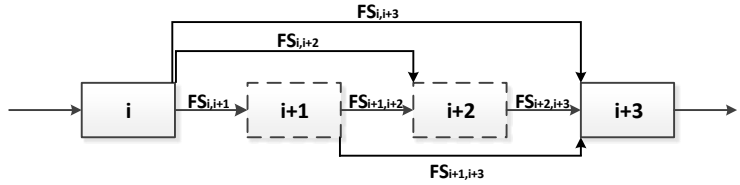


Figure 1: Finish-Start relations among optional activities.

In order to increase convenience for patients, the start of the first activity of patient p on each treatment day should only vary by a maximum of α per week. In order to guarantee this, we introduce decision variable $\widetilde{ds}_{pw}$ as the imaginary stable activity time per week w for patient p. The time at which patient p must come in on day d is denoted by ds_{pd} and is calculated as

$$ds_{pd} = \sum_{i \in \mathcal{DT}} s_{pi} \cdot dt_{pid} - \underline{dw_d} \quad \forall p, d, \tag{2}$$

with $\mathcal{DT}$ the set of all DT activities. Eq. 3 introduces the stable time constraint with $\mathcal{D}^w$ the set of days belonging to week w. Violations to this constraint – denoted by γ^{st}_{pd} – are to be minimized within the objective function.

$$|ds_{dp} - \widetilde{ds}_{pw}| - \gamma^{st}_{pd} \leq \alpha \quad \forall w, d \in \mathcal{D}^w, p \tag{3}$$

Each treatment activity requires numerous resources as well as staff capacity simultaneously. The bottleneck resource is considered to be the particle beam which can only serve one out of three treatment rooms at a time. One DT activity, however, consists of three parts: (1) A set-up time in which the patient is prepared for the treatment (occupying the treatment room only), (2) the irradiation itself where both the beam and the room are blocked and (3) the tear-down time during which again only the treatment room is occupied. Further, in case different particle types are supplied successively, an additional set-up time occurs on the beam resource.

Objective We aim at maximizing the usage of the bottleneck resource – the irradiation beam – by trying to reduce setup effort and idle times. Simultaneously, we focus on the minimization of penalties arising from time window violations, namely violations of the finish-start relations between two consecutive activities and violations of the stable times described above. Due to the complexity of the problem at hand and the lack of space in the extended abstract, we cannot state the full mathematical model formulation here.

2 Solution Approach

Constructive Heuristic Approaches We designed two construction heuristics that sequentially schedule the given activities: The first heuristic schedules activities chronologically, while the second heuristic allows to fill availability gaps on the resources. In order to reduce penalties from time window violations and thereby increase feasibility, we further apply repair strategies to the second heuristic. We use these construction heuristics in two main ways: (1) We apply classical priority rules like "Earliest Starting Time" known from the literature and (2) we schedule activities according to some predefined and order-feasible activity list. The latter use is relevant especially for the metaheuristic solution approaches described below, as it functions as a decoding mechanism of multi-encoded solutions.

The underlying chronological scheduling approach necessitates the formulation of an LP model to evaluate stable time violation penalties *after* the schedule construction has finished, since the stable time per week itself is a variable and not an input to the model (see Eq. 3).

Metaheuristic Solution Approaches We developed a genetic algorithm (GA) as well as a variable neighborhood search (VNS) based heuristic and a combination of both approaches (genetic local search). The representation of one solution is defined by a multi-encoded chromosome (see [1]) which consists of binary vectors indicating the assignment of DT to days for each patient as well as binary encodings displaying which optional activities to schedule and which ones to skip. Finally, a permutation vector including all patient indices depicts the sequence in which the patients are scheduled on each day. This encoding scheme is then transformed into an activity list which is used by the mentioned constructive scheduling schemes to decode and evaluate the solution.

Within the GA approach, we investigate on custom feasibility-preserving crossover and mutation operators tailored to our multi-encoding chromosome. We conclude, that our custom day-wise crossover of the binary encodings outmatches a patient-wise crossover approach.

The VNS process then further uses the mutation operators and focuses the search only on parts of the multi-encoded solution. If we end up in a local optimum, we continue the search on other parts of the solution. Finally, we combine the proposed local search process with the genetic algorithm and thereby further intensify the optimization. We compare the three mentioned approaches on real-world inspired problem instances for the radiotherapy patient scheduling problem of different size (i.e. numbers of patients to be treated).

References

[1] A. Costa, F. Cappadonna, S. Fichera (2013). *A hybrid genetic algorithm for job sequencing and worker allocation in parallel unrelated machines with sequence-dependent setup times.* In: International Journal of Advanced Manufacturing Technology, 69 (9), pp. 2799–2817.

[2] D. Petrovic, M. Morshed, S. Petrovic (2011). *Multi-objective genetic algorithms for scheduling of radiotherapy treatments for categorised cancer patients.* In: Expert Systems with Applications, 38 (6), pp. 6994–7002.

A Generalization of the Knapsack-Cover Inequalities for Linear Functions With Fixed Costs

Ignacio Morales (Speaker) * José Verschae †

1 Introduction

We consider a fundamental covering problem where we are given a demand $D \geq 0$ and a set J of n items that we can use to cover it. In the classical Knapsack-Cover (a.k.a Min-Knapsack) problem, each item i has a cost c_i and a value p_i that determines how much demand we cover if we choose item i to be in our knapsack. In other words, the objective is to determine variables $y_i \in \{0, 1\}$ such that $\sum_i y_i p_i \geq D$ and the cost $\sum_i c_i y_i$ is minimized. We consider a natural generalization of this basic question in which each item has an additional cost g_i and capacity u_i. Whenever we pick item i (which automatically covers p_i units of demand), we must decide for an extra amount of z_i units of demand to cover, each unit costing g_i. Variable z_i can take any fractional value within $[0, u_i]$. In this context we must find a solution minimizing $\sum_i c_i y_i + g_i z_i$ satisfying $\sum_i y_i p_i + z_i \geq D$. We denote this problem as Knapsack-Cover with Fixed and Variable cost (KCFV).

Our main motivation comes from the Unit Commitment Problem (UCP), a prominent problem in the operations of power systems. In its most basic version, a central planner, called Independent System Operator (ISO) must schedule the production of energy from a given set of resources (power plants) in order to satisfy a given demand. A common issue in this setting is that resources incur fixed costs for making the resource available, and after the resource is available a minimum amount of energy must be produced. For the case of one time period, the problem corresponds exactly to KCFV if the production cost scales linearly with power (after paying the fixed cost).

Related Work. Carr et al. [4] studied the Knapsack-Cover problem from a polyhedral perspective and derived a set of inequalities called *knapsack-cover inequalities* (KC-inequalities) that give an integrality gap of 2 when applied over the obvious IP-formulation. This set of inequalities have proven very useful in a myriad of different covering problems. This include, for example, the General Scheduling problem with [1] and without [5] release dates, and the special case corresponding to the unsplittable flow-cover problem on a path [2, 6].

Carnes and Shmoys [3] consider a similar setting to ours, with the difference that the p_i's are all zero, and derive a generalization of the KC-inequalities for this setting,

*inmorales@uc.cl. Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Santiago, Chile.
†jverschae@uc.cl. Facultad de Matemáticas and Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Santiago, Chile.

which then apply to the more general capacitated single-item lot-sizing problem. They obtain an elegant 2-approximation algorithm for this setting via a primal-dual schema.

2 Our Results

We extend the generalized KC-inequalities by Carnes and Shmoys [3] to the KCFV problem, yielding a $(2+\epsilon)$-approximation based on LP-rounding. The basic idea of the original KC-inequalities arise by the following thought experiment: assuming that we have already selected a set $A \subseteq J$ of items for our knapsack, how can the remaining items cover the remaining demand? If A is already selected, we have a residual demand of $D(A) = D - \sum_{j \in A} p_j$, and an item $i \in J \setminus A$ can have p_i replaced to $p_i(A) := \min\{p_i, D(A)\}$. Hence, for any $A \subseteq J$ the inequality $\sum_{i \notin A} x_i p_i(A) \geq D(A)$ is valid, which corresponds to the KC-inequalities.

Similarly to the basic case, consider the thought experiment where we assume that we have already paid the fixed cost for items of a set A. Also, we know that for items in a given set $B \subseteq A$ we have already selected the variable part up to its capacity. That is, $y_i = 1$ for $i \in A$ and $z_i = u_i$ for $i \in B$. Given this, we must still cover a residual demand of $D(A,B) = D - \sum_{i \in A} p_i - \sum_{i \in B} u_i$. As above, we can truncate the values p_i and u_i of every item i to $p_i(A,B) = \min\{D(A,B), p_i\}$ and $u_i(A,B) = \min\{D(A,B) - p(A,B), u_i\}$ respectively. These simple definitions suggest a relaxation with objective function $\sum_{i \in J} c_i y_i + z_i g_i$ and a constraint defined for each pair of subsets $B \subseteq A \subseteq J$,

$$\sum_{i \in J \setminus A} \min\{z_i, y_i u_i(A,B)\} + \sum_{i \in A \setminus B} \min\{z_i, u_i(A,B)\} + \sum_{i \in J \setminus A} p_i(A,B) y_i \geq D(A,B). \quad (1)$$

Here, we have that $z_i \geq 0$ and $y_i \geq 0$ for all $i \in J$. The first summation in the inequality represents the amount of demand covered by the fractional part of items in $J \setminus A$. Notice that in any solution item $i \in J \setminus A$ cannot cover more than $y_i u_i(A,B)$ amount of demand with its variable part, hence the term z_i in the summation can be capped by $y_i u_i(A,B)$. The second summation is similar, with the only difference that variables y_i are set to 1 for $i \in A \setminus B$. Finally, the third summation considers the integral part of items in $J \setminus A$ (using the truncated value of p_i). Notice that the formulation does not include the constraints $z_i \leq u_i$ and $y_i \leq 1$, as we can show that these inequalities are automatically satisfied by any optimal solution. We remark that the relaxation is non-linear. However it can be easily linearized using an exponential number of linear constraints. Moreover, the resulting set of inequalities can be (approximately) separated in polynomial time, and thus the relaxation can be solved in polynomial time up to a $(1+\epsilon)$ factor with the Ellipsoid method. Our main result is the following.

Theorem 1 *There is a poly-time procedure that takes a feasible solution (y,z) of the relaxation and produces a solution of KCFV of cost $\leq 2 \sum_i c_i y_i + z_i g_i$.*

Our rounding scheme works in three steps. In each of them we seek to decrease the number of fractional variables y_i or decrease the number of constraints.

First step: Consider a solution (y,z) of the relaxation above. We will define a new solution (y',z') where y' is integral. Let A^* be the set of items such that $y_i \geq 1/2$. Also consider a set $B^* \subseteq A^*$ such that $z_i \geq u_i(A^*,B^*)/2$. Notice that B^* is defined recursively.

We can show that such set always exists and can be found in poly-time. Moreover, we define part of our final solution (y', z') as $y'_i = 1$ for $i \in A^*$ and $z'_i = u_i(A^*, B^*)$ for $i \in B^*$ without duplicating the cost of (y, z) over these sets.

Second step: We now consider a reduced relaxation by taking constraint (1) using only sets A^* and B^*. Taking our solution $(y_i)_{i \in J \setminus A^*}$ and $(z_i)_{A^* \setminus B^*}$ and amplifying it by a factor of 2, we obtain a solution that is feasible even if the right-hand side of the inequality is changed to $2D(A^*, B^*)$. We consider a new relaxation with this inequality and adding the constraint $y_i \leq 1$. Let us consider an optimal solution $(\tilde{y}, \tilde{z})$ of this new relaxation, which can be solved in poly-time with the Ellipsoid method. This solution can still have many fractional variables $\tilde{y}$ (although we have only one inequality, it is a non-linear one). To handle this we can use the following proposition.

Proposition 2 *Let $(\tilde{y}, \tilde{z})$ be as above, then:*

i) For any $i \in J \setminus A^*$ such that $g_i \geq \frac{c_i}{p_i}$ and $\tilde{z}_i > 0$, then $\tilde{y}_i = 1$.

ii) For any $i \in J \setminus A^*$ such that $g_i < \frac{c_i}{p_i}$, then $\tilde{y}_i u_i(A^*, B^*) = \tilde{z}_i$.

This proposition is based on the fact that if there exists i such that $g_i \geq \frac{c_i}{p_i}$, then the solution will prefer to take $\tilde{y}_i = 1$ rather than increasing $\tilde{z}_i$. Also, for any i such that $g_i < \frac{c_i}{p_i}$, the solution will pay as little as possible of $\tilde{y}_i$ in order to be able to increase $\tilde{z}_i$.

Third step: The proposition allows us to pose a new equivalent relaxation where for each item either y_i is fixed to 1, z_i is fixed to 0, or $y_i u_i(A^*, B^*) = z_i$. This allows us to consider only one variable per item, and reduce the relaxation to a single linear constraint. Then, we take variables $\overline{z}_i$'s and $\overline{y}_i$'s (only one variable for each item in J) as the optimal solution of this reduced LP and denote by k the unique item with a fractional $\overline{y}_k$ (if it exists). Finally, in the case that $i \in J \setminus A^*$ lies in the first case of Proposition 2 and $\overline{y}_i = 1$ or $\tilde{z}_i > 0$, we set $y'_i = 1$ and $z'_i = u_i(A^*, B^*)$. Also, when $i \in J \setminus A^*$ lies in the second case of Proposition 2, we set $y'_i = \overline{y}_i$ and $z'_i = \overline{y}_i u_i(A^*, B^*)$. After this, we set y'_k and z'_k to 0.

References

[1] N. Bansal and K. Pruhs. The Geometry of Scheduling. *SICOMP*, 43:1684–1698, 2014.

[2] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *JACM*, 48:1069–1090, 2001.

[3] T. Carnes and D. B. Shmoys. Primal-dual schema for capacitated covering problems. *Math. Prog.*, 153:289–308, 2015.

[4] R. D. Carr, L. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceeding of SODA 2000*, pages 106–115, 2000.

[5] M. Cheung, J. Mestre, D. B. Shmoys, and J. Verschae. A Primal-Dual Approximation Algorithm for Min-Sum Single-Machine Scheduling Problems. *SIDMA*, To appear.

[6] W. Höhn, J. Mestre, and A. Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. In *Proceedings of ICALP 2014*, pages 625–636, 2014.

The Temp Secretary Problem and Partly-Stochastic Models for Online Scheduling

Thomas Kesselheim * Andreas Tönnis (Speaker) †

1 Introduction

In the secretary problem, candidates of different value arrive over time. After each arrival, the algorithm has to decide whether to permanently accept or reject this candidate. Every decision is final. In many scheduling applications, however, commitments are not eternal but affect only a finite time horizon. They may limit options for the upcoming days but not for the rest of the year or even longer.

Even with such an assumption of limited commitments, traditional worst-case competitive analysis is too strong a benchmark for the secretary problem. Fiat et al. [1] introduced an interesting, partly stochastic model, which allows much more meaningful analyses. First an adversary chooses which items will arrive. However, it does not determine the arrival times, which are instead drawn from a probability distribution, typically the uniform distribution on $[0, 1]$.

In more detail, in the *temp secretary problem*, an adversary defines values of items $v_1, \dots, v_n$. Afterwards, arrival times τ_j are drawn independently uniformly from $[0, 1]$. As time proceeds, the values and arrival times are revealed to the algorithm. Upon each arrival, the algorithm has to decide whether to accept or to reject the respective item. Each item is accepted for a duration of γ, which is assumed to be much smaller than 1. At any point in time t at most B items may overlap, that is, during time $t - \gamma$ and t at most B items may be accepted.

The objective is to maximize $\sum_{j \in \mathrm{ALG}} v_j$, where $\mathrm{ALG} \subseteq [n]$ denotes the selection by the algorithm. By OPT we denote the optimal selection $\mathrm{OPT} \subseteq [n]$, which maximizes $\sum_{j \in \mathrm{OPT}} v_j$. As the arrival times $\tau_1, \dots, \tau_n$ are random, both ALG and OPT are random variables. We evaluate the performance of an algorithm by its competitive ratio, defined as $\mathbf{E}\left[\sum_{j \in \mathrm{ALG}} v_j\right] / \mathbf{E}\left[\sum_{j \in \mathrm{OPT}} v_j\right]$.

2 Algorithmic Ideas and Results

In [3], we introduce a new algorithmic approach to online packing problems with temporal constraints. As key idea, we consider a relaxation to OPT by removing the temporal constraints and exchanging them with global ones. In the special case of the temp secretary problem, we exploit that, for every realization of the arrival dates $\tau_1, \dots, \tau_n$, the

*thomas.kesselheim@mpi-inf.mpg.de. Max Planck Institute for Informatics und Saarland University, Saarbrücken Informatics Campus, Germany.

†atoennis@uni-bonn.de. Department of Computer Science, University of Bonn, Germany. Supported by ERC Starting Grant 306465 (BeyondWorstCase).

```
for every arriving item j do
    Set t := τ_j;                                          // arrival time of j
    Let S^(t) be the ⌊tB/γ⌋ highest-valued items j' with τ_j' ≤ t;
    if j ∈ S^(t) then                                      // if among best items
        if S ∪ {j} is a feasible schedule then             // and if feasible
            Set S := S ∪ {j};                              // then select j
        end
    end
end
```

Algorithm 1: Scaling Algorithm for length γ and capacity B

optimal offline solution OPT never contains more that $B\lceil 1/\gamma \rceil$ elements. Therefore, we exchange the temporal constraints by only requiring $B\lceil 1/\gamma \rceil$ items to be picked throughout the process. An online solution to this relaxation can be found using algorithms for online linear packing problems. It then remains to derive a solution to the original constraints.

For the temp secretary problem, this approach leads to a light-weight, easy to state algorithm, given as Algorithm 1. For an item arriving at time t, our algorithm first checks if it is one of the approximately tB/γ highest valued items so far. The intuition is that a t fraction of $[0, 1]$ has passed and the relaxation selects approximately B/γ items in this entire time. Afterwards, we still have to make sure that it is actually feasible to select the current item. Only at this stage the temporal structure comes into play.

We show it to be $\frac{1}{2}\left(1 - O(\sqrt{\gamma})\right)$-competitive for all values of B. Furthermore, for large values of B, a different analysis shows a better competitive ratio of $1 - O\left(1/\sqrt{B}\right) - O(\sqrt{\gamma})$. Note that $1/2$ is known to be an asymptotic upper bound to the competitive ratio for $B = 1$ [1].

The approach also extends to the generalization in which items stay in the system for different durations. The definition of the set $S^{(t)}$ is then generalized to the solution of a knapsack problem. We also extend the problem from cardinality constraints to arbitrary linear constraints by combining Algorithm 1 with the ideas from [2].

3 Future Work

There are some immediate follow-up questions from our work. For example, we assumed that the arrival rate is constant at all points in time. In fact, it is easy to generalize our algorithm towards general arrival distribution using quantiles, but our analysis does not transfer. Another promising direction is towards different constraint structures, e.g. online flows in a network. In our results, we give general bounds for packing LPs, but for many problems with more structure better bounds should be possible.

In a much broader context, we also think that probabilistic online models have very high potential in scheduling problems. The model used in the temp secretary problem has some interesting features. Like other probabilistic models, it is not as pessimistic as the fully adversarial model, which does not allow any meaningful statements here, while still most parts of the input are assumed to adversarial. Additionally, this model has a very clear notion of time, making it a particularly good fit for scheduling problems. It seems reasonable to apply this kind of model to, for example, minimization objectives

such as flow time or feasibility conditions involving covering constraints or deadlines.

References

[1] Amos Fiat, Ilia Gorelik, Haim Kaplan, and Slava Novgorodov. *The Temp Secretary Problem*. In Proc. 23rd European Symp. Algorithms (ESA), pages 631 - 642, 2015.

[2] Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. *Primal Beats Dual on Online Packing LPs in the Random-Order Model*. In Proc. 46th Symp. Theory of Computing (STOC), pages 303 - 312, 2014.

[3] Thomas Kesselheim, and Andreas Tönnis. *Think Eternally: Improved Algorithms for the Temp Secretary Problem and Extensions*. In Proc. 24th European Symp. Algorithms (ESA), pages 54:1 - 54:17, 2016.

Conflict Graphs and Scheduling in Wireless Networks

Eyjólfur Ingi Ásgeirsson (Speaker) * Tigran Tonoyan †
Magnús M. Halldórsson ‡

1 Introduction

One of the main issue in resource management of wireless networks is assessing their capacity, i.e. how much communication can be achieved in a network, utilizing all the tools available: power control, scheduling, routing, channel assignment and rate adjustment? As pointed out by [5, 3], at the heart of these questions lies the *maximum weight independent set of links (*Mwisl*)* problem: From a given set of links with associated weights/utilities, find an *independent subset of maximum total weight*. This reduction applies to very general settings involving single-hop and multi-hop, as well as fixed and controlled transmission rate networks. Moreover, approximating Mwisl within any factor implies achieving the corresponding fraction of the capacity region, i.e. the set of traffic rates that can be supported by any scheduling policy. This makes Mwisl a central problem in the area.

Unfortunately, solving this problem in its full generality is notoriously hard, since it is well known that Mwisl is effectively inapproximable in models described by general conflict relations or general graphs. Moreover, in general, even approximating the capacity region in polynomial time within a non-trivial bound, while keeping the delays in reasonable bounds, is hard under standard assumptions [4].

We tackle this question in the *physical model* of communication with log-path fading and aim for polynomial time algorithms with approximation guarantees over interference-constrained networks of *arbitrary topology*. Towards this end, we develop a general approximation framework that not only helps us to approximate Mwisl, but can also be used for tackling various other scheduling problems, such as TDMA scheduling, joint routing and scheduling and others.

The crucial feature of the approach, which generalizes a framework of Halldorsson and Tonoyan [2], is that it involves transforming the complex physical model into an unweighted/undirected conflict graph and solving the problems simply on these graphs.

*eyjo@ru.is. ICE-TCS, School of Science and Engineering, Reykjavik University, Menntavegur 1, 101 Reykjavik, Iceland

†tigran@ru.is. ICE-TCS, School of Computer Science, Reykjavik University, Menntavegur 1, 101 Reykjavik, Iceland

‡mmh@ru.is. ICE-TCS, School of Computer Science, Reykjavik University, Menntavegur 1, 101 Reykjavik, Iceland.

2 Model

In scheduling problems, the basic object of consideration is a set L of n communication links, numbered from 1 to n, where each link $i \in L$ represents a communication request from a sender node s_i to a receiver node r_i.

We assume the nodes are located in a metric space with distance function d. We denote $d_{ij} = d(s_i, r_j)$ and $l_i = d(s_i, r_i)$. l_i is called the *length* of link i. Let $d(i, j)$ denote the minimum distance between the nodes of links i and j.

The nodes have adjustable transmission power levels. A *power assignment* for the set L is a function $P : L \to \mathbb{R}_+$. For each link i, $P(i)$ defines the power level used by the sender node s_i. In the physical model of communication, when using a power assignment P, a transmission of a link i is successful if and only if

$$SIR(S, i) = \frac{P(i)/l_i^\alpha}{\sum_{j \in S \setminus \{i\}} P(j)/d_{ji}^\alpha + N} \geq \beta_i, \tag{1}$$

where $\beta_i > 1$ denotes the minimum signal to noise ratio required for link i, $\alpha \in (2, 6)$ is the path loss exponent, N is ambient noise and S is the set of links transmitting concurrently with link i.

A set S of links is called *P-feasible* if the condition (1) holds for each link $i \in S$ when using power assignment P. We say S is *feasible* if there exists a power assignment P for which S is P-feasible.

3 Approximation Method

An independence system $\mathcal{I}$ over a set of vertices V is a pair $\mathcal{I} = (V, \mathcal{E})$, where $\mathcal{E} \subseteq 2^V$ is a collection of subsets of vertices that is closed under subsetting: if $S \in \mathcal{E}$ and $S' \subset S$, then $S' \in \mathcal{E}$. We define a notion of approximation of an independence system $\mathcal{I}_\mathcal{P} = (L, \mathcal{E}_\mathcal{P})$ by another independence system $\mathcal{I}_\mathcal{G} = (L, \mathcal{E}_\mathcal{G})$ over the set L of links. The system $\mathcal{I}_\mathcal{P}$ corresponds to the cumulative interference and physical model, while $\mathcal{I}_\mathcal{G}$ corresponds to a conflict graph $\mathcal{G} = \mathcal{G}(L)$ describing pairwise conflicts between links. The approximation is described by several key properties.

Refinement (Feasibility of Independent Sets): Every independent set in $\mathcal{I}_\mathcal{G}$ must be feasible, i.e. $\mathcal{E}_\mathcal{G} \subseteq \mathcal{E}_\mathcal{P}$. Thus, computing an independent set in $\mathcal{I}_\mathcal{G}$ gives also a feasible set in $\mathcal{I}_\mathcal{P}$.

Tightness (of refinement): There is a small number k such that every feasible set $S \in \mathcal{I}_\mathcal{P}$ is a union of at most k independent sets in $\mathcal{I}_\mathcal{G}$. The smallest such k is called the tightness of refinement. This property guarantees that even an optimal feasible set can be covered with a few independent sets.

Computability: There are efficient (approximation) algorithms for scheduling-related problems such as *vertex coloring* and *maximum weight independent set* in $\mathcal{I}_\mathcal{G}$.

An independence system $\mathcal{I}_\mathcal{G}$ satisfying the properties above is said to be a *refinement* of $\mathcal{I}_\mathcal{P}$. The two first properties establish a tight connection between the two independence systems. That allows us to take nearly *every* scheduling problem in the physical model and reduce it to the corresponding problem in conflict graphs by paying only an approximation factor depending on tightness k.

4 Conflict Graph Approximation of the Physical Model

Let us denote $\mathfrak{l}_i = \beta_i^{1/\alpha} l_i$ and call it the *effective length* of link i. Let $\Delta(L) = \max_{i,j \in L}\{\mathfrak{l}_i/\mathfrak{l}_j\}$ denote the *(effective) length diversity* of instance L. We call a set S of links *equilength* if for every two links $i, j \in S$, $\mathfrak{l}_i \leq 2\mathfrak{l}_j$, i.e., $\Delta(S) \leq 2$.

Definition 1. *Let $f : \mathbb{R}_+ \to \mathbb{R}_+$ be a positive non-decreasing function. Links i, j are said to be* f-independent *if*

$$d_{ij} d_{ji} > \mathfrak{l}_i \mathfrak{l}_j f\left(\frac{\mathfrak{l}_{max}}{\mathfrak{l}_{min}}\right),$$

where $\mathfrak{l}_{min} = \min\{\mathfrak{l}_i, \mathfrak{l}_j\}, \mathfrak{l}_{max} = \max\{\mathfrak{l}_i, \mathfrak{l}_j\}$. Otherwise they are said to be f-adjacent. *A set of links is f-independent (f-adjacent) if they are pairwise f-independent (f-adjacent).*

Let L be a set of links. The conflict graph $\mathcal{G}_f(L)$ is the graph with vertex set L, where two vertices are adjacent if and only if they are f-adjacent.

We are particularly interested in *sub-linear* functions $f(x) = O(x)$. A function f is *strongly sub-linear* if for each constant $c \geq 1$, there is a constant c' such that $cf(x)/x \leq f(y)/y$ for all $x, y \geq 1$ with $x \geq c'y$. Note that if f is strongly sub-linear then $f(x) = o(x)$. For example, the functions $f(x) = x^\delta$ ($\delta < 1$) and $f(x) = \log x$ are strongly sub-linear.

Theorem 2. *There is an $O(\log\log\Delta)$-tight refinement of the physical model by a conflict graph family $\mathcal{G}(L)$.*

The Mwisl problem, where the weights ω_i of links are fixed positive numbers, is a refinable packing problem. Also, it is well known, e.g. [1, 6], that vertex coloring and Mwisl problems are k-approximable in k-inductive independent graphs. Hence there is a constant factor approximation to Mwisl in $\mathcal{G}(L)$, which gives an $O(\log\log\Delta)$-approximation for the Mwisl problem in the physical model.

References

[1] K. Akcoglu, J. Aspnes, B. DasGupta and M.Y. Kao (2000). *Opportunity Cost Algorithms for Combinatorial Auctions.* CoRR, vol cs.CE/0010031.

[2] M.M. Halldórsson and T. Tonoyan (2015). *How Well Can Graphs Represent Wireless Interference?* STOC, pp. 635–644.

[3] X. Lin and N.B. Shroff (2004). *Joint Rate Control and Scheduling in Multihop Wireless Networks* IEEE Conference on Decision and Control, pp. 1484–1489.

[4] D. Shah, D.N.C. Tse and J.N. Tsitsiklis (2011). *Hardness of Low Delay Network Scheduling.* IEEE Trans. Information Theory, vol 57, no 12, pp. 7810–7817.

[5] L. Tassiulas and A. Ephremides (1992). *Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks.* IEEE Trans. Automat. Contr., vol. 37, no. 12, pp. 1936–1948.

[6] Y. Ye and A. Borodin (2012). *Elimination graphs.* ACM Transactions on Algorithms, vol. 8, no. 2, pp. 14:1–14:23.

On Packet Scheduling with Adversarial Jamming and Speedup

Martin Böhm * Łukasz Jeż† Jiří Sgall (Speaker)* Pavel Veselý*

We study an online packet scheduling model recently introduced by Anta *et al.* [1] and Jurdzinski *et al.* [2]. In this model, packets of arbitrary sizes arrive over time. The algorithm schedules one packet at any time, its transmission cannot be interrupted by the algorithm. (In the scheduling jargon, there is a single machine and no preemptions.) There are, however, (instantaneous) jamming errors at times chosen by the adversary and not known to the algorithm. A transmission taking place at the time of jamming is corrupt, and the algorithm learns this fact immediately. The packet whose transmission failed can be retransmitted immediately or at any later time.

The objective is to maximize the total size of packets successfully transmitted. The goal is to develop an online algorithm with the lowest possible competitive ratio. We focus on algorithms with resource augmentation, namely the online algorithms with speedup $s \geq 1$ compared to the offline solution. The main goal is to find the minimal speedup s for which the algorithm is 1-competitive; we also study the tradeoff of the competitive ratio and the speedup.

The distinguishing feature of the model is that the number of different packet sizes (lengths) is a constant, and also the packet sizes are considered to be constants. This means that the additive term in the definition of the competitive ratio may depend on the number and values of the packet sizes.

Jurdzinski *et al.* [2] presented algorithms both with and without speedup. For the important case of divisible packet sizes, they give a 1-competitive algorithm with speedup $s = 2$ and a 2-competitive algorithm with no speedup. For general packet sizes without speedup, the competitive ratio is in the interval $(2, 3)$ and its value depends on the sizes of the packets; from the results of Anta *et al.* [1] it follows that these ratios are optimal.

We develop two algorithms, *MAIN* for general packet lengths and *DIV* for divisible packet lengths. Our main new results concern the analysis of the non-divisible case with speedup. We also develop a uniform framework for the analysis of our algorithms, which significantly simplifies the proofs compared to [2], while we match the competitive ratios for the divisible case and obtain a 3-competitive no-speedup algorithm for the general case as well. In addition, the algorithms are simpler than in [2], and have the following desirable features:

- Our algorithms do not need to know possible packet sizes, while the algorithms in [2] depend on knowing the allowed packet sizes in advance.
- Our algorithms work for any speedup s which does not need to be known in advance, while in [2] the algorithms for the divisible case with and without speedup

*{bohm,sgall,vesely}@iuuk.mff.cuni.cz. Computer Science Institute of Charles University, Prague, Czech Republic. Partially supported by the project 17-09142S of GA ČR.

†lje@cs.uni.wroc.pl. Institute of Computer Science, University of Wrocław, Poland.

are different. I.e., our algorithm is the same no matter what is the speed (slowdown) of the offline solution to which we compare the algorithm.
- Our algorithms are always busy, in the sense that they always transmit some packet if there is any pending packet.

Algorithms

The general idea of the algorithm is that after each error, we start by transmitting packets of small sizes, only increasing the size of packets after a sufficiently long period of uninterrupted transmissions. It turns out that the right tradeoff is to transmit a packet only if it would have been transmitted successfully if started just after the last error. It is also crucial to start with the right packet size, namely to ignore small packet sizes if the total size of not transmitted packets of those sizes is small compared to a larger packet that can be transmitted. This guarantees that if no error occurs, all packets with size equal to or larger than the size of the initial packet are eventually transmitted.

We start by some notations. We assume there are k distinct packet sizes denoted by ℓ_i and ordered so that $\ell_1 < \cdots < \ell_k$. During the run of an algorithm, a packet is pending if it is released and not yet scheduled. Let $P^{<i}$ denote the total size of pending packets of sizes $\ell_1, \ldots, \ell_{i-1}$.

We divide the run of the algorithm into phases. The current time is denoted by t. The time t_B denotes the last invocation of step (2), which is the start of the current phase. We set $\mathrm{rel}(t) = s(t - t_B)$. Since the algorithm does not insert unnecessary idle time, $\mathrm{rel}(t)$ denotes the amount of transmitted packets in the current phase. (We use $\mathrm{rel}(t)$ only when there is no packet running at time t, so there is no partially executed packet.) Thus $\mathrm{rel}(t)$ can be thought as a measure of time relative to the start of the current phase (scaled by the speed s). Note also that the algorithm can evaluate $\mathrm{rel}(t)$ without knowing the speedup, as it can observe the total size of the transmitted packets.

We now give a common description of the two algorithms, *MAIN* for general packet sizes and *DIV* for divisible sizes. The algorithms differ only in that in step (3), *DIV* enforces an additional *divisibility condition*.

Algorithms *MAIN* **and** *DIV*

(1) If no packet is available, stay idle until the next release time.
(2) Let i be the maximal $i \le k$ such that $P^{<i} < \ell_i$. Schedule a packet of size ℓ_i.
(3) Choose the maximum i such that (i) there is a pending packet of size ℓ_i, (ii) $\ell_i \le \mathrm{rel}(t)$ and *(iii) in case the algorithm is DIV, ℓ_i divides rel(t).*
Schedule a packet of size ℓ_i. Repeat Step 3 as long as such i exists.
(4) If no packet satisfies the condition in Step 3, go to Step 1.

Analysis and results

The common scheme for all our proofs is the following. We carefully define critical times $C_k \le C_{k-1} \le \cdots \le C_1 \le C_0$ so that C_0 is the end of the schedule and C_i satisfies that (i) at time C_i, only a bounded number of packets of size ℓ_i is pending, while (ii) during the whole interval $(C_i, C_{i-1}]$, some packet of size ℓ_i is pending. The essential part of the proofs of R-competitiveness is to show that in each phase contained in $(C_i, C_{i-1}]$, the total size of packets of size ℓ_i or larger completed by the online algorithm is at least $1/R$ of the same quantity in the offline solution. The times C_i are defined so that before C_i, the online algorithm completes almost as many of packets of size ℓ_i as the offline solution (since only a few such packets are pending at C_i). These two facts together imply R-competitiveness of the algorithm.

We denote the algorithm ALG with speedup s by $ALG(s)$.

General packet lengths. For general packet lengths we have the following bounds:

Theorem 1 *$MAIN(s)$ is R_s-competitive where:*
$R_s = 1 + 2/s$ *for* $s \in [1, 4)$,
$R_s = 2/3 + 2/s$ *for* $s \in [4, 6)$, *and*
$R_s = 1$ *for* $s \geq 6$.

Our bound is 3 for $s = 1$, giving the same overall bound as [2]. Note that our bound has a discontinuity at $s = 4$. While for $s < 4$, we have examples showing that our algorithm is not 1-competitive, for $s \geq 4$ we are not able to exclude this possibility.

Well-separated packet lengths. We can obtain better bounds on the speedup necessary for 1-competitiveness if the packet sizes are sufficiently different. Namely, we call the packet lengths $\ell_1, \ldots, \ell_k$ α-separated if $\ell_i \geq \alpha \ell_{i-1}$ holds for $i = 2, \ldots, k$.

Theorem 2 *Let $\alpha > 1$. If the packet lengths are α-separated, then $MAIN(s)$ is 1-competitive for any $s \geq S_\alpha$, where*
$S_\alpha = (4\alpha + 2)/\alpha^2$ *for* $\alpha \in [1, \alpha_0]$,
$S_\alpha = 3 + 1/\alpha$ *for* $\alpha \in [\alpha_0, \alpha_1)$,
$S_\alpha = 2 + \frac{2}{\alpha}$ *for* $\alpha \geq \alpha_1$,
for $\alpha_0 = 1/2 + \sqrt{33}/6 \approx 1.46$ *and* $\alpha_1 = (3 + \sqrt{17})/4 \approx 1.78$.

Note that S_α is decreasing in α, with a single discontinuity at α_1. We have $S_1 = 6$, matching Theorem 1). Also, $S_2 = 3$, i.e., $MAIN(3)$ is 1-competitive for 2-separated packet lengths, which includes the case of divisible packet lengths studied below. The limit of S_α for $\alpha \to +\infty$ is 2.

Divisible packet lengths. We say that the packet lengths are divisible if ℓ_{i-1} divides ℓ_i for $i = 2, \ldots, k$. We obtain the following results:

Theorem 3 *For divisible packet lengths the following holds:*
- *$DIV(1)$ is 2-competitive.*
- *$DIV(s)$ is 1-competitive for $s \geq 2$.*
- *$MAIN(s)$ is 1-competitive for $s \geq 2.5$.*

This analysis of the algorithms is tight. We have examples with divisible packet lengths showing that for any $s < 2.5$, *MAIN* is not 1-competitive, and for any $s < 2$, *DIV* is no better than 2-competitive.

References

[1] A. F. Anta, C. Georgiou, D. R. Kowalski, J. Widmer, and E. Zavou. Measuring the Impact of Adversarial Errors on Packet Scheduling Strategies. In *Proc. of the 20th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO), Lecture Notes in Comput. Sci. 8179*, pages 261–273. Springer, 2013.

[2] T. Jurdzinski, D. R. Kowalski, and K. Lorys. Online packet scheduling under adversarial jamming. In *Proc. of the 12th Workshop on Approximation and Online Algorithms (WAOA), Lecture Notes in Comput. Sci. 8952*, pages 193–206. Springer, 2015.

Optimizing Egalitarian Performance in the Side-Effects Model of Colocation for Data center Resource Management

Fanny Pascual * Krzysztof Rzadca (Speaker) †

1 Introduction

The modern data center, the back-bone of cloud computing, redefines how industry and academia use computers. In data centers, up to dozens of tasks are colocated on a single physical machine [1]. Machines are used more efficiently, but, despite significant advances in both OS-level fairness and VM hypervisors, tasks' performance deteriorates [2], as colocated tasks compete for shared resources. Suspects include difficulties in sharing CPU cache or the memory bandwidth. As tasks are heterogeneous (CPU-, memory-, network- or disk-intensive), the resulting performance dependencies are complex. In order to optimize tasks' performance, the data center resource manager should thus try to colocate tasks that are compatible, i.e., that use different kinds of resources. This, however, requires a performance model.

Our side-effects model [3] bridges the gap between colocation in datacenters and classic scheduling, bulk of which has been developed for non-shared machines. Rather than trying to predict tasks' performance from OS-level metrics, we abstract by characterizing a task by two characteristics: its *type* (e.g.: a database, or a computationally-intensive job) and its *load* relative to other tasks of the same type (e.g.: number of requests per second). For each type we then use a performance function mapping a vector of loads (an element being the total load of tasks of a certain type) to a type-relevant performance metric. As datacenters execute multiple instances of tasks we believe that such function can be inferred by a monitoring module matching task's reported performance (such as the 95th percentile response time) with observed or reported loads.

2 Our Results

We consider optimization of the worst-off performance (analogous to makespan in classic multiprocessor scheduling problem, $P||C_{\max}$). We use a linear performance function: on each machine, the influence a type t' has on t's performance is a product of the load of type t' and a coefficient $\alpha_{t',t}$. The coefficient $\alpha_{t',t}$ describes how compatible t' load is with t performance (the coefficient is similar to interference/affinity metrics proposed in [2]). Low values ($0 \leq \alpha_{t',t} < 1$) describe small impact, thus compatible types (e.g.: colocating a memory-intensive and a CPU-intensive task): it is preferable to colocate

*fanny.pascual@lip6.fr. Sorbonne Universités, UPMC (Université Paris 6), LIP6, CNRS, UMR 7606, Paris, France.

†krz@mimuw.edu.pl. Institute of Informatics, University of Warsaw, Warsaw, Poland.

a task t with tasks of the other type t', rather than with other tasks of its own type t. High values ($\alpha_{t'',t} > 1$) denote incompatible types competing for resources, i.e., less incentive to colocate (at least from the tasks' owner's point of view).

Our main results are the following (see [5] for proofs). The notion of type adds complexity, as makespan minimization with unit tasks $P|p_i = 1|C_{\max}$ (a polynomially solvable variant of $P||C_{\max}$) becomes NP-hard and hard to approximate when the number of types T is not constant.

We then show how to cope with that added complexity. First, we propose a PTAS for a constant T. Our PTAS has a similar structure to the PTAS for $P||C_{\max}$ [4]. The two main differences are the treatment of short tasks (which we pack into containers, and not simply greedy schedule); and sizing of long tasks.

We also propose a fast greedy approximation algorithm. The algorithm groups tasks by *clusters*. All the tasks of the same type are in the same cluster. Two tasks of type i and j are in the same cluster iff their types are compatible ($\alpha_{i,j} \leq 1$ and $\alpha_{j,i} \leq 1$). Clusters are processed one by one. Each cluster is dedicated at least one machine. The algorithm puts tasks from a cluster on a machine until machine load reaches $\max\{2L, L + p_{\max}\}$, then opens the next machine ($L = \sum p_i/(m - T)$).

To characterize in detail the optimal schedules in function of the coefficient α, we study a series of special cases with two types. We identify two tipping points, i.e., values of α for which the shape of the optimal schedule changes. For $0 \leq \alpha \leq 1$, all machines should be shared between types (if possible). For $1 < \alpha < 2$, there are some instances that share all machines, but for divisible load (i.e., many small tasks), there is at most one shared machine. Finally, for $\alpha \geq 2$ at most one machine is shared. For each case, we show fast approximation algorithms.

In addition to worst-case performance proofs, we test our algorithm by simulation on a trace derived from one of Google clusters. We show that our algorithms lead to more efficient allocations compared to algorithms designed for $P||C_{\max}$.

Acknowledgements This research has been partly supported by a Google Faculty Research Award, a Polish National Science Center grant Sonata (UMO-2012/07/D/ST6/02440); and a Polonium grant (joint programme of the French Ministry of Foreign Affairs, the Ministry of Science and Higher Education and the Polish Ministry of Science and Higher Education).

References

[1] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *SoCC*, ACM, 2012.

[2] A. Podzimek, L. Bulej, L. Y. Chen, W. Binder, and P. Tuma, "Analyzing the impact of cpu pinning and partial cpu loads on performance and energy efficiency," in *CCGrid Proc.*, 2015.

[3] F. Pascual and K. Rzadca, "Partition with side effects," in *HiPC 2015, Procs.*, 2015.

[4] D. S. Hochbaum and D. B. Shmoys, "Using dual approximation algorithms for scheduling problems theoretical and practical results," *JACM*, vol. 34, no. 1, pp. 144–162, 1987.

[5] F. Pascual and K. Rzadca, "Optimizing egalitarian performance in the side-effects model of colocation for data center resource management," *arXiv:1610.07339*, 2016.

The Itinerant List Update Problem

Neil Olver* Kirk Pruhs (Speaker)† Kevin Schewior‡ René Sitters§
Leen Stougie¶

1 Introduction

We introduce a variation of the online List Update Problem, which we call the Itinerant List Update Problem (ILU). The setting consists of n (data) items, that without loss of generality we will assume are the integers $[n] = \{1, \ldots, n\}$, stored linearly in n locations on a track (tape). The track has a single read/write head. Requests for these items arrive over time. In response to the arrival of request for an item x, the online algorithm can perform an arbitrary sequence of the following unit cost operations:

Move: Move the head to the left, or to the right, one position.

Swap: Swap the item pointed to by the head with the adjacent item on the left, or the adjacent item on the right.

In order to be a feasible response, at some point in this response sequence, the tape head must point to the position holding x. The objective is to minimize the total cost over all requests.

Our motivation for introducing ILU arises from track management within Domain Wall Memory (DWM). Figure 1 depicts a basic DWM track, which consists of a ferromagnetic nanowire, a read/write head, and access transistors [2, 4]. The nanowire holds multiple domains separated by ultra narrow domain walls. Each domain represents one binary bit using its magnetic polarity. All domains of one nanowire share one read/write head that can extract or modify the domain under the head. By injecting current pulses through the two ends of the nanowire, the domains can be "shifted" in both directions such that they can be moved under the head to be read and written. We envision that in an architecture with w bit words, tracks will be grouped into super-tracks, each consisting of w tracks (so technically ILU is about super-track management).

*Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam, and CWI, Amsterdam. n.olver@vu.nl. Supported in part by an NWO Veni grant.

†Computer Science Department, University of Pittsburgh. kirk@cs.pitt.edu. Supported in part by NSF grants CCF-1421508 and CCF-1535755, and an IBM Faculty Award.

‡Universidad de Chile, Santiago de Chile. kschewior@gmail.com. Supported by DFG grant GRK 1408, Conicyt Grant PII 20150140, and the Millennium Nucleus Information and Coordination in Networks ICM/FIC RC130003.

§Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam, and CWI, Amsterdam. r.a.sitters@vu.nl

¶Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam, and CWI, Amsterdam. l.stougie@vu.nl

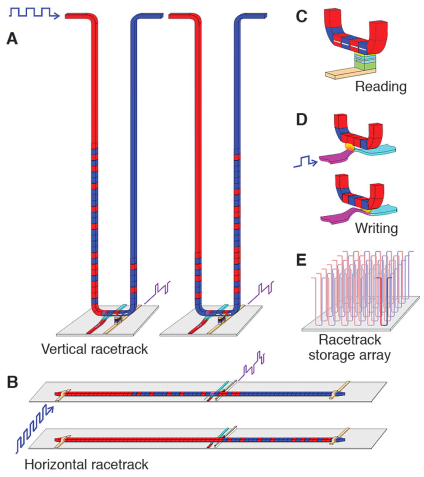


Figure 1: Domain Wall Memory [2].

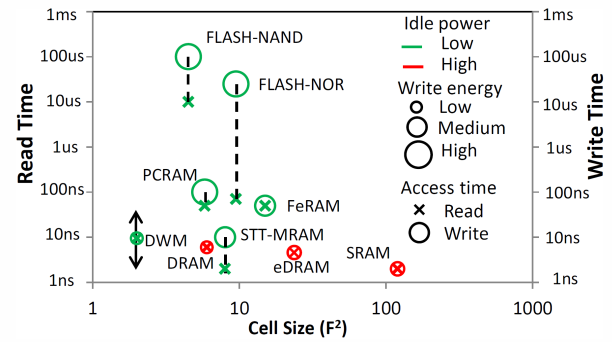


Figure 2: Comparing Technologies [4].

Figure 2 compares different memory technologies, including the emerging memory technologies, such as Phase-change Memory (PCM), Spin-Transfer Torque magnetic RAM (STT-MRAM), Ferroelectric RAM (FeRAM), and Domain Wall Memory (DWM), that can be potentially integrated in mobile devices. From Figure 2, one can see that among these technologies, DWM has the highest density. Density is one of the major constraints on enlarging on-chip memory in mobile devices. But one can also see that the access time for DMW is somewhat variable, with the low end being competitive with the access time of the fastest technologies, and the high end being an order of magnitude slower. This variability is due to the time for shifting. So good track management policies may enable access times for DMW to be competitive with the fastest memory technologies.

The main difference between ILU and the standard list update problem is that in the standard list update problem there is the additional feasibility constraint that at the end of each response sequence, the head has to return to a fixed home position. If the head has a home position, the algorithm/policy Move-To-Front (MTF), which moves the last accessed item to the home position (and moves intermediate items one position further from the home) is $O(1)$-competitive (This can be shown by simple modifications to the analysis of MTF in [3], in which the home position is the first position, and costs are defined somewhat differently). One natural adaptation of MTF for ILU would be: Move the currently requested item to a position adjacent to the location of the previously requested item. However this algorithm is $\Omega(n)$-competitive on the following simple request sequence, which assumes the items are initially ordered as $1, \ldots, n$ and that the head is initially in the first position: $1, n, n-1, n-2, \ldots, 2$. The algorithm moves item $n-i$, $0 \leq i \leq n-2$, to position $i+2$ in the track, at a cost of $\Omega(n^2)$, while the cost would only be $2n$ if the items are not reordered. There are slightly more complicated instances that show that the other natural adaptations of MTF also $\tilde{\Omega}(n)$-competitive. These lower bounds hint at an additionally difficulty of ILU relative to the standard list update problem. In both problems it seems natural for the online algorithm to aggregate recently accessed items together. However, in the standard list update problem it is obvious where to aggregate these items, near the home location, while in ILU, it seems unclear where these items should be aggregated.

We show that the expected competitive ratio of every randomized online algorithm for ILU is $\Omega(\log n)$. In the lower bound instance, the adversary picks an initial random permutation π for the items in its track, and never swaps any items. Thus intuitively to be competitive, the online algorithm must "learn" π. The requests are designed in such a way that the adversary can handle them relatively cheaply, without revealing too much information about π. So the lower bound shows that every online algorithm may have to spend an $\Omega(\log n)$ factor more than optimal to "learn" an ordering. Intuitively this shows that the "where to aggregate problem" for ILU is insurmountable, at least in terms of an online algorithm achieving $O(1)$-competitiveness.

We show that ILU is essentially equivalent to a variation of the standard Minimum Linear Arrangement Problem (MLA), which we call the Dynamic Minimum Linear Arrangement (DMLA) problem, in that a polylog(n)-competitive algorithm for one can be used to obtain an polylog(n)-competitive algorithm for the other. The setting for DMLA is the same as the setting for ILU, a linear track of items. In DMLA a sequence of graphs arrive over time, where the vertices of these graphs are the items stored in the track. In response to the arrival of a graph H_t at time t, the algorithm can perform an arbitrary sequence of swaps of adjacent items in the track. The objective is to minimize the swap cost plus the service cost. The *swap cost* is the sum over all times t, of the number of swaps made in response to the arrival of H_t. The *service cost* is the sum over all times t, of the sum over the edges $\{x, y\} \in H_t$ of the distance between x and y in the track after the swaps made in response to H_t. Note that in DMLA there is no concept of track head, and swaps can be made anywhere on the track. The standard MLA problem is essentially a special case of the DMLA problem in which all of the many arriving graphs are identical (so there is nothing to be gained from reordering the track).

We give an offline polynomial-time algorithm for DMLA assuming that every graph given as input consists of a single edge, and show that it has an approximation ratio of $O(\log^2 n)$. By applying our reductions, we obtain a polynomial-time $O(\log^2 n)$-approximation for ILU. Our algorithm for DMLA is similar to the divide-and-conquer algorithm for MLA in [1], except that it works on a time expanded graph, which creates some additional complications. This shows that computational complexity is not a barrier to achieving a poly-log approximation for ILU.

References

[1] M. D. Hansen. Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems. In *FOCS*, pages 604–609, 1989.

[2] S. S. P. Parkin, M. Hayashi, and L. Thomas. Magnetic domain-wall racetrack memory. *Science*, 320(5873):190–194, 2008.

[3] D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *CACM*, 28(2):202–208, 1985.

[4] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan. STAG: spintronic-tape architecture for GPGPU cache hierarchies. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 253–264, 2014.

The Greedy Algorithm for Capacitated Covering Problems

Britta Peis * José Verschae † Andreas Wierz (Speaker) ‡

1 Introduction

Integer programs of the form $\min\{c^T x : Ax \geq r, x \in \mathbb{Z}^n_+\}$ (P) with $A \in \mathbb{Z}^{\mathcal{L}\times E}_+$, $r \in \mathbb{Z}^{\mathcal{L}}$ and $c \in \mathbb{Z}^E_+$ are called covering problems. Here, we assume that $\mathcal{L}$ is a family of subsets of the elements which, for each row of A, describes the columns which are zero. That is, $S \in \mathcal{L}, e \in S \Rightarrow A_{S,e} = 0$. The name stems from the following observation: solutions to such problems use (multiple) copies of the columns in order to cover the righthandside value for each constraint. Since all coefficients are non-negative, adding additional copies of a column can not render a solution vector infeasible.

There are interesting connections between covering and scheduling problems. In fact, several single machine scheduling problems such as $1|pmnt|\sum_j f_j(C_j)$, with f_j non-decreasing, or $1|r_j|\sum_j w_j C_j$ can be formulated as covering problems in a relatively simple way. The former was pointed out in [11] to be a generalization of unsplittable flow cover on a path, which can be 4-approximated [1].

Besides this, many interesting combinatorial optimization problems can also be formulated in this manner. Famous examples are subset cover, cut covering and contra-polymatroids. Contra-polymatroids are of the form above with $\mathcal{L}$ being the Boolean lattice and constraints of the form $\sum_{e\notin S} x_e \geq r(S)$ for all $S \in \mathcal{L}$. The function $r : \mathcal{L} \to \mathbb{Z}_+$ is supermodular, monotone decreasing and non-negative. For contra-polymatroids, a very simple dual greedy algorithm is able to obtain an optimum solution. The dual to the linear relaxation of the covering problem above can be stated as $\max\{y^T r : y^T A \geq c, y \geq 0\}$ (D). Set $S = \emptyset$, increase the corresponding dual variable y_S until some element e gets tight. Set $x_e = r(S + e) - r(S)$, add e to S and iterate until $r(S) = 0$. Polymatroids, and the depicted greedy algorithm, are known for almost half a century [4].

For the knapsack cover problem, and many other problems as well, the exact same algorithm was used in order to obtain constant factor approximations [1, 2, 12]. In most of these known approximation results, a problem specific analysis was used. Many times, these analyses look very similar and use essentially the same techniques.

If $A \in \{0,1\}^{m\times n}$, there are fairly general conditions on the system (A, r) known, which ensure that the greedy algorithm computes an optimum solution [4, 5, 6, 8, 9, 10]. Most results have in common that the constraints form some kind of lattice on which r is either submodular or supermodular. We discuss similar conditions in case of $A \in \mathbb{Z}^{m\times n}_+$ and approximate solutions.

*britta.peis@oms.rwth-aachen.de. RWTH Aachen University
†jverschae@uc.cl. Pontificia Universidad Católica de Chile
‡andreas.wierz@oms.rwth-aachen.de. RWTH Aachen University

2 Our results

We consider integer covering problems (P) with constraints for subsets of elements which form a ring-family. A family $\mathcal{F}$ of subsets of groundset E is called a ring-family if any two sets $S, T \in \mathcal{F}$ imply $S \cup T, S \cap T \in \mathcal{F}$. That is, a family is called a ring-family, if it is closed under union and intersection. We also assume that $\emptyset, E \in \mathcal{F}$. A ring-family forms a lattice $\mathcal{L} = (\mathcal{F}, \subseteq, \cup, \cap)$. The Boolean lattice is an example for a ring-family.

We analyze the following simple dual greedy algorithm. Initialize $y \equiv 0, x \equiv 0$. Select the minimal element in $\mathcal{L}$ and increase y_S until $\sum_S y_S A_{S,e} = c_e$ some element $e \notin S$. Let S' be the minimal element in $\mathcal{L}$ which also contains e. Set $x_e = \lceil \frac{r(S)^+ - r(S')^+}{A_{S,e}} \rceil$. Remove all elements from $\mathcal{L}$ which do not contain S' and iterate until $r(S) \leq 0$.

In general, we may not assume that the greedy algorithm will terminate with a feasible solution for (P). But we can show that it will, if the system (A, r) satisfies the following conditions.

(P1) r is monotone: $r(S) \geq r(T)$ for all $S \preceq T$ and

(P2) supermodular: $r(S) + r(T) \leq r(S \vee T) + r(S \wedge T)$ for all $S, T \in \mathcal{L}, \in \mathcal{T}$.

(P3) For each element $e \in E$, $A_{*,e}$ is monotone: $A_{S,e} \geq A_{T,e}$ for all $S \preceq T$.

(P4) For every two sets $S \preceq T \in \mathcal{L}$ with $r(T) > 0$ and $e \notin T$, let $S' = \min(\mathcal{L} \setminus S \setminus e)$ and $T' = \min(\mathcal{L} \setminus T \setminus e)$. Then $\frac{r(S) - r(S')}{A_{S,e}} \geq \frac{r(T) - r(T')}{A_{T,e}}$.

Unfortunately, these properties do not suffice in order to obtain good approximation guarantees. Even with two elements, we can construct instances which have an unbounded integrality gap. Moreover, we can show that subset cover can be formulated as a problem of type (P) satisfying (P1) - (P4). Hence, no $o(1 - \log n)$ approximation for (P) exists unless $NP = DTIME(n^{O(\log \log n)})$ [7]. The former issue can be handled by a careful truncation of coefficients of the matrix A.

Definition 1. *Given a system (A, r) on a ring-family satisfying (P1) - (P4), we define the system (A', r) as below to be the* truncated system. *For $S \in \mathcal{L}$ and $e \notin S$, let $S' = \min(\mathcal{L} \setminus S \setminus e)$. Then set $A'_{S,e} = \min\{A_{S,e}, r(S)^+ - r(S')^+\}$.*

We can show that the truncated system (A', r) contains the same integer feasible points as the system (A, r). Although the truncation no longer satisfies (P4), we can also show that the greedy algorithm applied to system (A', r) still obtains a feasible solution. The approximation factor depends on the following parameter δ. Given $S \in \mathcal{L}$ with $e \notin S$, let $S' = \min(\mathcal{L} \setminus S \setminus e)$ and define $\delta_{S,e} = \frac{A'_{\emptyset,e}}{A'_{S,e}}$, if $r(S') \geq 0$ and $A'_{S,e} > 0$, and $\delta_{S,e} = 1$, otherwise. Let $\delta = \max_{S,e} \delta_{S,e}$ be the maximum of these terms.

The value of δ is the maximal ratio between coefficients in A' which may occur on a chain in the dual constructed by the greedy algorithm before the rank becomes negative. Intuitively, the following happens: If the algorithm selects many elements with small coefficient $A'_{S,e}$, these may have a very large contribution $A'_{\emptyset,e} \leq \delta A'_{S,e}$ towards the constraint for $S = \emptyset$. Hence, a small value of δ guarantees that no constraint is oversubscribed by a large factor. This, in-turn, results in a good approximation guarantee using standard arguments. Our main result is the following.

Theorem 2. *The greedy algorithm applied to the truncation (A', r) of a system (A, r) on a ring-family satisfying (P1) - (P4) obtains a solution with cost no larger than $b(\delta + a)OPT$. Where $a = 0$, if r is non-negative and $a = 1$, otherwise. Moreover, $b = 1$, if $\frac{r(S)^+ - r(S')^+}{A'_{S,e}} \in \mathbb{Z}_+$ for all $S \in \mathcal{L}, e \notin S, S' = \min(\mathcal{L} \setminus S \setminus e)$, and $b = 2$, otherwise.*

The approximation factor coincides with many well-known results, for example, for polymatroids, vertex cover, or knapsack cover with item multiplicity and separable convex cost- and concave utility functions. In terms of lower bounds, we can construct a family of instances whose truncation (A', r) has an integrality gap of $o(\log \delta)$.

References

[1] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *JACM*, 2001.

[2] T. Carnes and D. Shmoys. Primal-dual schema for capacitated covering problems. *IPCO*, 2008.

[3] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 1979.

[4] J. Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial structures and their applications*, 1970.

[5] U. Faigle and S. Fujishige. A general model for matroids and the greedy algorithm. *Mathematical programming*, 2009.

[6] U. Faigle and W. Kern. An order-theoretic framework for the greedy algorithm with applications to the core and weber set of cooperative games. *Order*, 2000.

[7] U. Feige. A threshold of ln n for approximating set cover. *JACM*, 1998.

[8] A. Frank. Increasing the rooted-connectivity of a digraph by one. *Mathematical Programming*, 1999.

[9] S. Fujishige. A note on frank's generalized polymatroids. *Discrete Applied Mathematics*, 1984.

[10] S. Fujishige. Dual greedy polyhedra, choice functions, and abstract convex geometries. *Discrete Optimization*, 2004.

[11] W. Höhn, J. Mestre, and A. Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. *ICALP*, 2014.

[12] S. McCormick, B. Peis, J. Verschae, and A. Wierz. Primal–dual algorithms for precedence constrained covering problems. *Algorithmica*, 2016.

[13] A.Schulz and J. Verschae. Min-sum scheduling under precedence constraints. *LIPIcs-Leibniz International Proceedings in Informatics*, 2016.

A Parallel Machine Lot-Sizing and Scheduling Problem With a Secondary Resource and Cumulative Demand

Murat Güngör (Speaker) * Ali Tamer Ünal †

1 Introduction

Motivated by foundry planning in aluminum alloy wheel production, we consider a novel parallel machine lot-sizing and scheduling problem: A number of items are to be produced on identical parallel machines. In order to produce an item, a piece of equipment (mold) particular to that item must be installed in the machine. Planning horizon is divided into buckets for which all-or-nothing assumption is valid. The objective is to minimize the number of setups (mounts) and teardowns (dismounts). The novelty lies in the assumption that demands are given for the entire planning horizon rather than for every single period. In this respect, our problem differs from those in the literature (Lasdon and Terjung, 1971; Eppen and Martin, 1987; Vanderbeck and Wolsey, 1992; Jans and Degraeve, 2004; Gicquel et al., 2012). Cumulative demand is not only a realistic assumption as far as automotive industry is concerned, but it also leads to a problem bearing an interesting structure.

2 Formulation

We formulate two equivalent mixed-integer programs (cf. Gicquel et al., 2012). In the first one, machines are modeled explicitly, whereas in the second, aggregate variables are used. Let x_{it} be the total number of molds of type i used in period t. Let s^+_{it} be the number of setups required for molds of type i in period t; similarly define s^-_{it} for teardowns ($t > 1$). We denote by T and M the number of periods and machines, respectively, and by $\overline{x_i}$ the demands as average number of molds to be used in a period.

*murat.gungor@boun.edu.tr. Industrial Engineering Department, Boğaziçi University, 34342, Istanbul, Turkey.

†unaltam@boun.edu.tr. Industrial Engineering Department, Boğaziçi University, 34342, Istanbul, Turkey.

The second (aggregate) formulation is given below.

$$
\begin{aligned}
\min \quad & \sum_{i,t} (s^+_{it} + s^-_{it}) & \\
\text{s.t.} \quad & \sum_i x_{it} \le M & \text{for all } t \\
& \frac{1}{T} \sum_t x_{it} \ge \overline{x_i} & \text{for all } i \\
& x_{i,t-1} + s^+_{it} - s^-_{it} = x_{it} & \text{for all } i, t \\
& s^+_{it}, s^-_{it} \ge 0 & \text{for all } i, t \\
& x_{it} \in \mathbb{Z}_{\ge 0} & \text{for all } i, t
\end{aligned}
$$

3 Results

Our theoretical results on the problem are as follows. Proofs are omitted except the outline provided for that of computational complexity.

Theorem 1. *(i) The problem is feasible if and only if* $\sum_i \overline{x_i} \le M$*. In this case, the optimal objective value* z^* *must be an even integer.*
(ii) If $\sum_i \lceil \overline{x_i} \rceil \le M$*, then* $z^* = 0$*.*
(iii) z^* *cannot be greater than* $2I - 2$*, where* I *denotes the number of mold types.*

Theorem 2. *The problem is NP-hard.*

Proof. Let $a_1, \ldots, a_m$ and $b = \frac{1}{2} \sum_{k=1}^m a_k$ be positive integers. Consider an instance of the problem where $M = 2$, $I = m$, $T = b$, and $\overline{x_i} = \frac{a_i}{T}$. Then PARTITION has a positive answer if and only if $z^* \le 2I - 4$. □

We conjecture that the following two sets of inequalities are optimality cuts:

$$\lfloor \overline{x_i} \rfloor \le x_{it} \le \lceil \overline{x_i} \rceil \qquad \text{for all } i, t \tag{1}$$

$$\sum_t s^+_{it} \le 1, \quad \sum_t s^-_{it} \le 1 \qquad \text{for all } i. \tag{2}$$

Proving this claim turns out to be tricky. The difficulty is, making one mold type fit the conditions possibly spoils the situation for other mold types, so it is unlikely that an argument based purely on local manipulations will do. Nevertheless, we can give two partial results. First, note that any one of (1) and (2) implies

$$s^+_{it} \le 1, \quad s^-_{it} \le 1 \qquad \text{for all } i, t. \tag{3}$$

Theorem 3. *Assuming* (3)*, the inequalities* (1) *are optimality cuts for* $T \le 3$*.*

Theorem 4. *Assuming* (1)*, the inequalities* (2) *are optimality cuts for* $T \le 4$*.*

We propose a polynomial-time heuristic algorithm for the problem. Feeding the solver with an initial heuristic solution slightly improves solution performance. This is also true of setting the absolute gap tolerance to $2 - \varepsilon$ (recall that z^* must be even). However, the cuts (1) and (2) do not seem to be useful according to our computational study.

Finally, we discuss two possible extensions to the problem related to real-life applications. First, we consider "vertical constraints," limiting the number of setups and teardowns in each period (cf. Lasdon and Terjung, 1971). Second, we study "minimum lot size restrictions," which translates in our problem as a lower bound on the number of periods a mold should remain within a machine after being installed. Such a restriction might stem from shop floor peculiarities (e.g. materials to be processed may only be stored and retrieved in certain large batches), or a managerial decision based on the observation that scrap rates gradually decrease as one produces more and more after a new setup. We show how these two sets of constraints can be incorporated into the formulations and elaborate on some properties of the new problems arising thereby. Computational tests indicate that it is significantly more time-consuming to solve these extended models.

References

Gary D. Eppen and R. Kipp Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35(6):832–848, 1987. doi:10.1287/opre.35.6.832.

C. Gicquel, L. A. Wolsey, and M. Minoux. On discrete lot-sizing and scheduling on identical parallel machines. *Optimization Letters*, 6(3):545–557, 2012. doi:10.1007/s11590-011-0280-8.

Raf Jans and Zeger Degraeve. An industrial extension of the discrete lot-sizing and scheduling problem. *IIE Transactions (Institute of Industrial Engineers)*, 36(1):47–58, 2004. doi:10.1080/07408170490247296.

L. S. Lasdon and R. C. Terjung. An Efficient Algorithm for Multi-Item Scheduling. *Operations Research*, 19(4):946–969, 1971. doi:10.1287/opre.19.4.946.

Francois Vanderbeck and Laurence A Wolsey. Valid Inequalities for the Lasdon-Terjung Production Model. *The Journal of the Operational Research Society*, 43(5):435–441, 1992. doi:10.1057/jors.1992.70.

Scheduling Demand Response on the French Spot Power Market for Water Distribution Systems by Optimizing the Pump Scheduling

Chouaib Mkireb (Speaker) * Abel Dembele † Antoine Jouglet ‡
Thierry Denoeux §

1 Introduction

The control of peak electricity demand is more and more important in a context of massive integration of renewable energies and some new uses of electricity: electric vehicles, heat pumps, etc. Balancing in real time load and generation is a difficult task for Transmission System Operators (TSO) because they are facing several uncertainties: weather change, unavailability of production plants, network congestions, etc. In France, electricity consumption is highly driven by weather conditions, especially in winters because of the preponderance of electric heating in households. In cold winters, a decrease of 1 degree Celsius of temperature implies an increase of 2300 MW of electricity demand, it is the Thermo-Sensibility phenomenon. The peak of consumption occurred on the 08^{th} February 2012, estimated to 102 GW at 7 pm, alerted the French TSO RTE (Réseau Transport d'Electricité), and showed the need to develop efficient methods for the active management of demand. Demand Response (DR), defined as the change in the power consumption of an electric utility customer in response to a given signal, brings flexibility to the electric grid by adapting consumption to production. Industrial processes are believed to be the best candidates for Demand Response, especially those with storage units: warehouse, electric batteries. They can adapt their energy consumption to the needs of the electric network, in return for remuneration. Water systems, thanks to reservoir storage and pipe transfer capacities, are electrically flexible industrial processes that can contribute to balance the electric grid, allowing electricity production with low greenhouse gas emissions.

In this talk, we present the opportunities and constraints for water systems to participate in efficient Demand Response mechanisms in France, providing flexibility for

*c.mkireb@suez.com. Suez Smart Solution, 38 Avenue du Président Wilson, 78320 Le pecq, France. chouaib.mkireb@hds.utc.com. Sorbonne Universités, Université de Technologie de Compiègne, CNRS, UMR 7253 Heudiasyc, France

†abel.dembele@suez.com. Suez Smart Solution, 38 Avenue du Président Wilson, 78320 Le pecq, France.

‡antoine.jouglet@hds.utc.fr. Sorbonne Universités, Université de Technologie de Compiègne, CNRS, UMR 7253 Heudiasyc, France

§thierry.denoeux@hds.utc.fr. Sorbonne Universités, Université de Technologie de Compiègne, CNRS, UMR 7253 Heudiasyc, France

the electric grid by reducing peak consumptions. We evaluate the economic benefits for water utilities for optimizing pump scheduling and Demand Response trading operations on the French spot power market (NEBEF Mechanism), under time of use (ToU) electric contracts. We formulate the problem as a non-linear mixed integer optimization problem, and propose a linearization approach to make it tractable. Then, we solve the problem using a branch and bound algorithm for a range of water demand scenarios and electricity prices on the French spot market. Using a real water system in France, the financial reliability of the NEBEF Demand Response mechanism is shown to allow for up to 50% of savings on the total energy bill.

2 Context and Objective

The NEBEF mechanism is applicable since April 2016 in France. It allows to sell energy curtailment of an energy consumer, called a DR bloc, in day ahead on the spot market via a DR operator. The DR bloc is sold at the market price, which is the intersection of the supply and the demand curves. The DR operator must then compensate financially the supplier of the site with energy curtailment for the energy injected into the network and valued by the DR operator on the market [1]. The amount of compensation is regulated and depends on the type of the day: season, time, working or non-working days. The final incentive for the DR operator is the difference between the spot price and the compensation. Since it is a recent mechanism, no study has been carried out in this field to our knowledge. Furthermore, France is the only country in Europe to allow DR to participate directly to Day Ahead (D-1) markets as a resource [2]. However, some authors have been interested in water systems and their participation to some other DR mechanisms around the world: Demand Response and water distribution systems in UK [3], Demand Response in California Agricultural Irrigation [4]. These studies present the design of the local Demand Response Markets, and evaluate the potential, the benefits and the constraints for water systems to take part of these markets.

In this talk, we formalize the optimization problem resulting in participation of water systems in the Demand Response NEBEF mechanism. We model Demand Response in the objective cost function of a water utility aiming at:

1. minimizing the total electricity cost due to pumping operations (*) ;
2. maximizing the revenues earned from trading DR blocs on the spot market (**) ;
3. respecting all the physical and operational constraints of the water distribution system;
4. respecting all the NEBEF constraints as described by RTE [5].

3 Mathematical formulation

We denote by :

1. $x_{i,t}$: the binary variable indicating the state of the pump i at time t.
2. $C_{i,t}$: the electric cost when pump i in ON at time t.

3. d_t: the binary variable indicating if we participate in the spot market at time t or not.

4. $P_{dr,t}$: the electric power (DR bloc) put on sale at time t.

5. r_t: the market spot price at time t (in €/ MWh).

We write the objective function ensuring (*) and (**) as follows:

$$\min_{x_{i,t},d_t,P_{dr,t}} \sum_{i,t} C_{i,t} * x_{i,t} - P_{dr,t} * d_t * r_t \ .$$

The objective function aims at making, for each step time t, a trade-off between electric consumption by activation of pumps, and energy curtailment by selling the energy not consumed on the spot market in day ahead.

In this talk, we will first deal with the modeling aspect of the NEBEF constraints: minimum DR blocs per event, shape of the load curve for a DR event, maximum duration of an event, etc. Then, a mathematical approach will be proposed to linearize some of these constraints and the second term of the objective function. We will use a branch and bound algorithm to solve the optimization problem for a range of water demand and electricity price scenarios. The model has been tested on a real network composed of more than 120 pumps and 70 storages. The numerical results show that the NEBEF mechanism is the most reliable in winter for both the water utility and the electric grid, between 18:00 and 20:00:

1. the DR blocs are the most profitable for the water utility in terms of economic gain: 3% to more than 50% of gain compared to the normal optimal pump planning, according to scenarios;

2. 10 to 15 MW of peak power demand reduction per DR event, which corresponds to 7 to 10 Kg of CO2 emissions avoided per event.

References

[1] Thomas, R. (2016). *Contruire la valeur d'un service énergtique : la trajectoire de l'effacement diffus en France*. Université de Grenoble - PACTE.

[2] Thomas, V. (2014). *Market design for Demand Response : the French experience*. International Energy Agency, Demand Response Workshop.

[3] Ruben, M., Edo,A., Panos, P., Ivan, S. (2015). *Demonstrating demand response from water distribution system through pump scheduling*. Journal of Applied Energy, Volume 170 pp 377–387.

[4] Daniel, O., Goli, S. (2013). *Opportunities for Demand Response in California Agricultural Irrigation : A scoping study*. Lawrence Berkeley National Laboratory.

[5] RTE. (2016). *Règles pour la valorisation des effacements de consommation sur les marchés de l'énergie NEBEF 2.1* .

An Exact Method for Solving the Two-Machine Flow-Shop Problem With Time Delays

Mohamed Amine Mkadem (Speaker) * Aziz Moukrim * Mehdi Serairi *

1 Introduction

We address the flow-shop scheduling problem with two machines and time delays, which is denoted by $F2|l_j|C_{max}$. $F2|l_j|C_{max}$ can be described as follows. Given a set $J = \{1, 2, \ldots, n\}$ of n jobs and two machines (M_1, M_2), each job $j \in J$ has two operations $O_{1,j}$ and $O_{2,j}$. The operation $O_{1,j}$ (resp. $O_{2,j}$) must be executed without preemption during $p_{1,j}$ (resp. $p_{2,j}$) time units on M_1 (resp. M_2). In addition to the resource constraints of the machines and the non-preemption of jobs, a schedule S is considered feasible if, for each job $j \in J$, a time delay of at least l_j time units must elapse between the completion of $O_{1,j}$ and the start of $O_{2,j}$. The goal is to find a feasible schedule S that minimizes the makespan, i.e, the maximum job completion time.

The $F2|l_j|C_{max}$ problem is NP-hard in the strong sense even with unit-time operations [5]. Therefore, it has been the scope of a variety of investigations. As far we know, [2] and [4] proposed lower bound methods. Moreover, [2] investigated heuristic approaches where he introduced four constructive heuristics and a Tabu Search algorithm. It should be noted that [2] implemented an exact method based on the branch-and-bound method of [1], which is originally made for the job-shop problem. Another branch-and-bound method was proposed by [3] for the unit-time operations case.

The objective of this paper consists in introducing a branch-and-bound algorithm for $F2|l_j|C_{max}$. To our aim, we propose a set of lower bounds and an upper bound. In addition, a set of dominance rules is proposed in order to reduce the search space.

2 The Branch-and-Bound Algorithm

In this section, we present an exact method for $F2|l_j|C_{max}$ based on a branch-and-bound enumeration scheme. At first, let us introduce the following observation.

Observation 1 *Let σ be a fixed job sequence on M_1 of all jobs, then schedules in which the job sequence σ is fixed first on M_1 and the jobs are executed on M_2 according to the nondecreasing order of their arrival times are dominant.*

According to this observation, our branch-and-bound enumerates job sequences on M_1 as follows. At a given node N_{σ_1} of the search tree of the branch-and-bound, a partial

*{mohamed-amine.mkadem, aziz.moukrim, mehdi.serairi}@hds.utc.fr. Sorbonne universités, Université de technologie de Compiègne, CNRS, UMR 7253 Heudiasyc - CS 60 319 - 60 203 Compiègne cedex.

job sequence σ_1 of $|\sigma_1|$ jobs is fixed on M_1. In order to reduce the number of explored nodes, we invoke at each node N_{σ_1} the following features:

- A preprocessing procedure that aims to fix some precedence relationships between jobs on each machine.
- Several lower bounds of [2] and [4] that have been adapted to be used inside the branch-and-bound search tree.
- An upper bound based on a local search technique applied on the current subsequence.
- Three dominance rules that allow to discard nodes from additional expansions in order to reduce the computational time of the proposed branch-and-bound algorithm.

If these features fail to prune the current node N_{σ_1}, then for each unscheduled job j a *son node* is created from N_{σ_1}, in which j is fixed after σ_1. Note that we adopted the depth-first node selection strategy.

3 Computational results

A computer simulation of the branch-and-bound algorithm, which was carried out on a set of 360 instances of [2], shows that our branch-and-bound method outperforms the state of the art exact method. In particular, we manage to solve 358 instances among 360 possible ones.

References

[1] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1):107 – 127, 1994.

[2] M. Dell'Amico. Shop problems with two machines and time lags. *Operations Research*, 44(5):777–787, 1996.

[3] A. Moukrim, D. Rebaine, and M. Serairi. A branch and bound algorithm for the two-machine flowshop problem with unit-time operations and time delays. *RAIRO - Operations Research*, 48(2):235–254, 2014.

[4] W. Yu. *The two-machine flow shop problem and the one-machine total tardiness problem.* PhD thesis, Eindhoven University of Technology, The Netherlands, 1996.

[5] W. Yu, H. Hoogeveen, and J.K. Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard. *Journal of Scheduling*, 7(5):333 – 348, 2004.

A New Necessary Condition of Optimality for a Single Machine Scheduling Problem With Deteriorating Jobs

Stanisław Gawiejnowicz (Speaker)* Wiesław Kurc†

1 Introduction

We consider the following open scheduling problem [5]. The set of $n+1$ independent jobs $J_0, J_1, \ldots, J_n$, where $n > 2$, has to be scheduled on a single machine, starting from time 0. The processing time of job J_j linearly deteriorates in time and is equal to $p_j(t) = 1 + \alpha_j t$, where t is the starting time of the job, $\alpha_j > 0$ is its deterioration rate and $0 \leq j \leq n$. Let $a^\circ = (a_0, a_1, \ldots, a_n)$ denote the initial sequence of coefficients $a_j = 1 + \alpha_j > 1$, where $0 \leq j \leq n$. Since the order of jobs with the same deterioration rate has no impact on schedule optimality, we assume that $a_i \neq a_j$ whenever $i \neq j$ and $0 \leq i, j \leq n$. Let $\mathcal{P}(a^\circ)$ denote the set of all permutations of sequence a°. Then any sequence $a^\pi \in \mathcal{P}(a^\circ)$, in which the elements of sequence a° are ordered according to a permutation $\pi = (\pi_0, \pi_1, \ldots, \pi_n)$ of indices $0, 1, \ldots, n$, corresponds to a schedule in which the job with index π_0 starts at time 0 and all remaining jobs are scheduled one by one without idle times. Hence, for a given $a^\pi = (a_{\pi_0}, a_{\pi_1}, \ldots, a_{\pi_n}) \in \mathcal{P}(a^\circ)$, where $a_{\pi_j} = 1 + \alpha_{\pi_j}$ for $0 \leq j \leq n$, the completion time of the jth job in the schedule corresponding to sequence a^π is equal to $C_j(a^\pi) = C_{j-1}(a^\pi) + p_{\pi_j}(C_{j-1}(a^\pi)) = 1 + a_{\pi_j} C_{j-1}(a^\pi)$, with $C_0(a^\pi) = 0 + 1 + \alpha_{\pi_0} \cdot 0 = 1$, since job J_{π_0} starts at time $t = 0$. Then, the problem under consideration is to find a sequence $a^\sigma \in \mathcal{P}(a^\circ)$ such that $\sum_{j=0}^{n} C_j(a^\sigma) = \min_{a^\pi \in \mathcal{P}(a^\circ)} \left\{ \sum_{j=0}^{n} C_j(a^\pi) \right\}$, where job completion times $C_j(a^\pi)$ are defined as above. This problem will be called *problem* (P).

Problem (P) is a dynamic, time-dependent scheduling problem in which deteriorating job processing times are non-decreasing functions of the starting times of the respective jobs. We refer the reader to monographs [1, 2, 6] for the most recent discussion on different types, properties and applications of dynamic scheduling problems.

Research on problem (P) has a 25-year-old history that was initiated by paper [5], where the problem was formulated and its basic properties were proved. The most important property proved there, which we will call *the first necessary condition* of schedule optimality for problem (P), says that if $(a_{\pi_0}, a_{\pi_1}, \ldots, a_{\pi_n}) \in \mathcal{P}(a^\circ)$ is an optimal schedule for (P), then it is V-shaped. (A sequence $(a_{\pi_0}, a_{\pi_1}, \ldots, a_{\pi_n})$ is said to be V-shaped, if there exists $0 \leq m \leq n$ such that $a_{\pi_k} \geq a_{\pi_{k+1}}$ for any $0 \leq k \leq m-1$ and $a_{\pi_k} \leq a_{\pi_{k+1}}$ for any $m \leq k \leq n-1$.) Let us also notice that a similar property holds for more general variations of problem (P) [3].

*stgawiej@amu.edu.pl. Faculty of Mathematics and Computer Science, Adam Mickiewicz University in Poznań, Umultowska 87, 61-614 Poznań, Poland.

†wkurc@amu.edu.pl. Faculty of Mathematics and Computer Science, Adam Mickiewicz University in Poznań, Umultowska 87, 61-614 Poznań, Poland.

The first necessary condition implies the bound $O(2^n)$ on the number of possible optimal schedules for problem (P). In this talk, we present a new *second necessary condition* of schedule optimality for (P), decreasing the bound $O(2^n)$ by the factor $O(n^{-\frac{1}{2}})$, and a few other results related to this new condition [4].

2 Auxiliary results

Below we present the results we use in the proofs of the main results given in Section 3.

Given $a = (a_1, \ldots, a_r, \ldots, a_q, \ldots, a_n)$, where $1 \leq r < q \leq n$, let sequence b be in the form of $\overline{a}^{q-r}$ or $\underline{a}^{q-r}$, where $\overline{a}^{q-r} = (a_1, \ldots, a_{r-1}, a_q, a_r, \ldots, a_{q-1}, a_{q+1}, \ldots, a_n)$ and $\underline{a}^{q-r} = (a_1, \ldots, a_{r-1}, a_{r+1}, \ldots, a_q, a_r, a_{q+1}, \ldots, a_n)$.

Let $a = (a_1, \ldots, a_m, \ldots, a_n) \in \mathcal{P}(a^\circ)$ be a V-shaped sequence such that a_m is the minimal element in a and $1 < m < n$. Index r is said to be *m-conjugated* to index q, if $m < q \leq n$, $1 \leq r < m$ and $\overline{a}^{q-r}$ is V-shaped. Similarly, index q is said to be *m-conjugated* to index r, if $1 \leq r < m$, $m < q \leq n$ and $\underline{a}^{q-r}$ is V-shaped. Let

$$\Delta_k(r,q) = \sum_{i=1}^{q-k-1} \prod_{j=i}^{q-k-1} a_j - \sum_{i=q-k+1}^{q-1} \prod_{j=q-k+1}^{i} a_j - \frac{1}{a_q} \sum_{i=q+1}^{n} \prod_{j=q-k+1}^{i} a_j$$

and

$$\nabla_k(r,q) = \frac{1}{a_r} \sum_{i=1}^{r-1} \prod_{j=i}^{r+k-1} a_j + \sum_{i=r+1}^{r+k-1} \prod_{j=i}^{r+k-1} a_j - \sum_{i=r+k+1}^{n} \prod_{j=r+k+1}^{i} a_j,$$

where $1 \leq r < q \leq n$ and $k = 1, 2, \ldots, q-r$.

Lemma 1 *If $a = (a_1, \ldots, a_r, \ldots, a_q, \ldots, a_n)$ with $1 \leq r < q \leq n$, then we have $||C(\overline{a}^{q-r})||_1 - ||C(a)||_1 = \sum_{k=1}^{q-r}(a_q - a_{q-k})\Delta_k(r,q)$ and $||C(\underline{a}^{q-r})||_1 - ||C(a)||_1 = \sum_{k=1}^{q-r}(a_{r+k} - a_r)\nabla_k(r,q)$.*

3 Main results

The first two main results are equivalent formulations of the new necessary condition of schedule optimality for problem (P). In view of space limitations we skip details of an iterative procedure which effectively verifies the inequalities given in Theorem 3.

Theorem 2 *Let $a = (a_1, a_2, \ldots, a_n)$ be an optimal schedule for problem (P). Then (i) a is V-shaped, the minimal element in a is a_m, where $1 < m < n$, and (ii) for any r and q such that $1 \leq r < m < q \leq n$ we have*

$$\Delta_1(r,q) = \sum_{j=1}^{q-2} \prod_{k=j}^{q-2} a_k - \sum_{i=q+1}^{n} \prod_{k=q+1}^{i} a_k \;\geq\; 0$$

and

$$\nabla_1(r,q) = \sum_{j=1}^{r-1} \prod_{k=j}^{r-1} a_k - \sum_{i=r+2}^{n} \prod_{k=r+2}^{i} a_k \;\leq\; 0.$$

Theorem 3 *Let $a = (a_1, a_2, \ldots, a_n)$ be an optimal schedule for problem (P). Then (i) a is V-shaped, the minimal element in a is a_m, where $1 < m < n$, and (ii) there hold inequalities*

$$\Delta_1(m-1, m+1) = \sum_{j=1}^{m-1} \prod_{k=j}^{m-1} a_k - \sum_{i=m+2}^{n} \prod_{k=m+2}^{i} a_k \geq 0$$

and

$$\nabla_1(m-1, m+1) = \sum_{j=1}^{m-2} \prod_{k=j}^{m-2} a_k - \sum_{i=m+1}^{n} \prod_{k=m+1}^{i} a_k \leq 0.$$

Given an instance a° of problem (P), let $V_I(a^\circ)$ and $V_{II}(a^\circ)$ denote the sets of all $a \in \mathcal{P}(a^\circ)$ that satisfy the old and the new necessary condition, respectively. Definitions of sets $\mathcal{P}(a^\circ)$, $V_I(a^\circ)$ and $V_{II}(a^\circ)$ imply that $V_{II}(a^\circ) \subset V_I(a^\circ) \subset \mathcal{P}(a^\circ)$. Moreover, since $|\mathcal{P}(a^\circ)| = n!$ and $|V_I(a^\circ)| = 2^n$, we have $|V_{II}(a^\circ)| \leq |V_I(a^\circ)| < |\mathcal{P}(a^\circ)|$.

Let $u = \min\{a_i^\circ : i = 1, 2, \ldots, n\}$ and $v = \max\{a_i^\circ : i = 1, 2, \ldots, n\}$. Let us notice that $1 < u < v$. Let $d_n = n\frac{\log u}{\log u + \log v}$ and $g_n = n\frac{\log v}{\log u + \log v} + 1$.

The next result describes possible indices of the minimal element a_m in $a \in \mathcal{P}(a^\circ)$.

Theorem 4 *Let $a = (a_1, \ldots, a_m, \ldots, a_n) \in \mathcal{P}(a^\circ)$ be a V-shaped schedule for problem (P) such that a_m is the minimal element in a. Then (i) if $u < v$ are arbitrary, then $d_n < m < g_n$; (ii) if $v \to u$, then $\frac{n}{2} \leq m \leq \frac{n}{2}+1$; (iii) if $v \to +\infty$, then $m \in \{1, 2, \ldots, n\}$.*

Let us notice that since $1 \leq \lceil d_n \rceil \leq m \leq \lfloor g_n \rfloor \leq n$, Theorem 4 implies that the set of possible values of m is $D = \{\lceil d_n \rceil, \ldots, m, \ldots, \lfloor g_n \rfloor\} \subset \{1, \ldots, n\}$.

Let $V_D(a^\circ)$ denote the set of all $a = (a_1, \ldots, a_m, \ldots, a_n) \in \mathcal{P}(a^\circ)$ such that a is V-shaped and $m \in D$. Let us notice that $V_{II}(a^\circ) \subset V_D(a^\circ) \subset V_I(a^\circ)$ and the cardinality of $V_D(a^\circ)$ equals $|V_D(a^\circ)| = \sum_{k=\lceil d_n \rceil}^{\lfloor g_n \rfloor} \binom{n}{k}$.

Our last main result gives bounds on the cardinality of the set $V_D(a^\circ)$.

Theorem 5 *Let us denote $c(n) = \sqrt{\frac{2}{\pi n}}\, 2^n \left(1 + O\left(\frac{1}{n}\right)\right)$ and let u and v, $1 < u < v$, be defined as above, respectively. Then $|V_{II}(a^\circ)| \leq |V_D(a^\circ)| \leq \left(1 + \frac{\log v - \log u}{\log v + \log u} n\right) \times c(n)$ and, if v is sufficiently close to u, $|V_D(a^\circ)| \geq c(n)$.*

References

[1] A. Agnetis, J-C. Billaut, S. Gawiejnowicz, D. Pacciarelli and A. Soukhal (2014). *Multiagent Scheduling: Models and Algorithms*. Springer, Berlin-Heidelberg.

[2] S. Gawiejnowicz (2008). *Time-Dependent Scheduling*. Springer, Berlin-New York.

[3] S. Gawiejnowicz and W. Kurc (2015). Structural properties of time-dependent scheduling problems with the l_p norm objective. *Omega*, **57**, 196–202.

[4] S. Gawiejnowicz and W. Kurc (2016). A new necessary condition of optimality for a single machine scheduling problem with deteriorating jobs, submitted.

[5] G. Mosheiov (1991). V-shaped policies in scheduling deteriorating jobs. *Operations Research*, **39**, 979–991.

[6] V.A. Strusevich and K. Rustogi (2017). *Scheduling with Times-Changing Effects and Rate-Modifying Activities*. Springer, Berlin-Heidelberg.

Scheduling When You Don't Know the Number of Machines

Clifford Stein * Mingxian Zhong (Speaker) †

1 Introduction

For many problems, one does not know the entire input accurately and completely in advance. There are different ways of addressing such uncertainty, e.g. via online algorithms (assuming that the input arrives over time), dynamic algorithms (assuming the input changes over time) or robust optimization (assuming that there is bounded uncertainty in the data). Another way of addressing uncertainty is to require one solution that is good against all possible values of the uncertain parameters. Examples of work in this direction include the universal traveling salesman problem (one tour that is good no matter which subset of points arrive) [9], robust matchings (one matching is chosen and then evaluated by its top k edges, where k is unknown) [7], a knapsack of unknown capacity (one policy of packing that is good irrespective of the actual capacity) [5] and 2-stage scheduling (some decisions must be made before the actual scenario is known) [10]. In scheduling problems, there are many ways to model uncertainty in the jobs, including online algorithms [1], in which the set of jobs is not known in advance, and work on schedules that are good against multiple objective functions [4]. But there is much less work studying the possibility of uncertainty in the machines, and the work we are aware of studies uncertainty in speed or reliability (breakdowns) [2].

Motivated by the need to understand how to make scheduling decisions without knowing how many machines we will have, we consider a different notion of uncertainty, that is, a scenario in which you don't know how many machines you are going to have, but you still have to commit (partially) to a schedule by making significant decisions about partitioning the jobs before knowing the number of machines.

This type of decision arises in a variety of settings. For example, many scheduling problems are fundamentally about packing items onto machines and there are many examples of problems that concern packing items where there are multiple levels of commitment to be made with partial information. For example, in a warehouse, you may need to take a large order and place it into multiple boxes, without knowing exactly how many trucks you will have to ship the items. You therefore want to be able to pack the items well, given the various possible number of trucks. Another example involves problems in modern data centers. In data centers, there are some systems which require you to group work together into "bundles" without knowing exactly how many machines you will have. For example, in a map-reduce type computation, the mapping function

*cliff@ieor.columbia.edu. Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027, USA.

†mz2325@columbia.edu. Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027, USA.

you define naturally breaks the data into some number of groups g. However, you actually have some number of machines m at your disposal, and you typically have to design your mapping function, choosing a g and associated grouping, without knowing m. You may know a range of possible values for m, or it may vary widely depending on the availability of machines at the time you run the map reduce computation (and the availability is typically not under your control). As more and more computing moves to the "cloud", that is, moves to large shared data centers, we anticipate that this problem of grouping work without knowing the number of machines will become more widespread.

In this paper, we consider one of the simplest scheduling problems – minimizing makespan on identical parallel machines. We choose this problem as a proof of concept and many other objectives could have been chosen. We consider the following specific model. We are given a set of n jobs, J, with known processing times $p(j)$, and a number M, which is an upper bound on the number of machines we might have. An algorithm must commit, before knowing how many machines there are, to grouping the jobs into M *packs*, where each job is assigned to exactly one of the packs and some packs are possibly empty. We call this step the *packing* step. Only then, do we learn the number of machines m. We now need to compute a schedule, with the restriction that we must keep the packs together, that is, we will assign one or more packs to each machine. We call this step the *scheduling* step. As in other robust problems, we want to do well against all possible numbers of machines. We therefore evaluate our schedule by the ratio of the makespan of our schedule, $ALG(m, M)$, to the makespan of a schedule that knew m in advance, opt_m, taking the worst case over all possible values of m. If an algorithm always provides a ratio of at most α, where $\alpha = \max_{1 \le m \le M} \frac{ALG(m,M)}{opt_m}$, we call it α-robust. Note that when the number of machine is given, then there exists a polynomial time approximation scheme [3, 8] and a simple 4/3-approximation algorithm [6] (longest processing time).

1.1 The Main Results

Our first result is on an introductory problem. We assume that we have an optimal schedule on M machines, $OPT(M)$, as our packs. We then ask how well these packs can be scheduled on $M/2$ machines, compared to $opt_{M/2}$, the unconstrained optimal schedule for $M/2$ machines. We show that this ratio is at most 2 and give an example where it is 2. This ratio can be improved if we choose among all optimal schedule, the one that has the lowest L_q norm of the machine loads, for finite $q > 1$. Starting from these packs, we show that the ratio is now 3/2, and that this analysis is tight.

We next consider the general problem of scheduling the packs not on $M/2$ machines, but on any number $m \in [1, M]$. Here we give an algorithm that is 2-robust, where we use $OPT(M)$ as our packing. The 2 is tight, given that $OPT(M)$ is our packing, but it is possible that a better bound could be found using a different initial packing. We explore this possibility and provide positive results in two directions. First, we consider the case where the jobs are all infinitesimal, and give a 3/2-robust algorithm. This result shows that some of the difficulty comes from dealing with large jobs. Finally, we consider the case where we have an upper and a lower bound on the number of machines. In particular, we consider the case where we know that the eventual number of machines will be between $M/2$ and M, and give an algorithm which gives a robust ratio of $\frac{9}{5}(1+\epsilon)$, breaking the simple $2(1+\epsilon)$ bound. This algorithm is more involved than the previous

ones, and demonstrates how a more careful investigation of the packing and scheduling steps can lead to improved bounds.

In this algorithm, we first calculate the optimal schedule for every fixed $m \in [M/2, M]$ within a factor of $1 + \epsilon$ using the algorithm in [3]. Denote opt'_k the approximation to the optimal makespan for a fixed k. Then we follow three steps. First we choose one or more of these schedules as *bases.* We then take these bases, which are on some number $k \leq M$ machines and *separate* the jobs that were scheduled on some machines into a larger number of sets, which we call *packs.* After learning how many machines we have, we place the packs on the machines. We remark that in separation procedure we provide three separation algorithms with good properties which could be useful in related problems. For each of these steps, there are multiple cases to consider. We first handle two of the easier cases – when opt'_M is large relative to $opt'_{M/2}$ and when there is a K such that opt'_{K+1} is much smaller than opt'_K. Note that the two cases can be easily extended to the case when $m \in [\alpha M, M]$, where $\alpha \in (0, 1)$. We then handle the remaining cases by considering the value of $opt'_{M/2}$ relative to $opt'_{3M/4}$.

References

[1] S. Albers, Recent advances for a classical scheduling problem. In Proceedings of ICALP, Springer LNCS 7966, 4-14, 2013.

[2] S. Albers and G. Schmidt, Scheduling with unexpected machine breakdowns. Discrete Applied Mathematics, 110(2-3):85-99, 2001.

[3] N. Alon, Y. Azar, G. J. Woeginger and T. Yadid, Approximation schemes for scheduling on parallel machines. Journal on Scheduling, 1:55-56, 1998

[4] J. A. Aslam, A. Rasala, C. Stein and N. E. Young, Improved Bicriteria Existence Theorems for Scheduling. In Proceedings of SODA, 846-847, 1999.

[5] Y. Disser, M. Klimm, N. Megow, S. Stiller, Packing a Knapsack of Unknown Capacity. In the Proceedings of STACS, 276-287, 2014.

[6] R. L. Graham, Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. 45 , 1563-1581, 1966.

[7] R. Hassin and S. Rubinstein, Robust matchings. SIAM Journal on Discrete Mathematics 15(4):530-537, 2002.

[8] D. S. Hochbaum and D. B. Shmoys, Using dual approximation algorithms for scheduling problems: theoretical and practical results. Journal of the ACM, 34:144-162, 1987.

[9] L. K. Platzman and I. John J. Bartholdi, Spacefilling curves and the planar travelling salesman problem. Journal of the ACM, 36, 719-737, 1989.

[10] D. B. Shmoys and M. Sozio, Approximation algorithms for 2-stage stochastic scheduling problems. In Integer Programming and Combinatioral Optimization, 145-157, 2007.

Faster Approximation Schemes for the Two-Dimensional Knapsack Problem

Sandy Heydrich (Speaker) * Andreas Wiese †

1 Introduction

We study the two-dimensional geometric knapsack problem for squares: Given a set of input items, which are squares of side lengths between 0 and 1, each with an associated weight or profit, and a knapsack of size $N \times N$, our goal is to find an axis-aligned packing of a maximum weight subset of the items into the knapsack without overlappings. This is a natural generalization of the fundamental, widely applicable one-dimensional knapsack problem. There is a polynomial time $(1+\epsilon)$-approximation algorithm for it (i.e., a PTAS) but the running time of this algorithm is triple exponential in $1/\epsilon$, i.e., $\Omega(n^{2^{2^{1/\epsilon}}})$ [4]. A double or triple exponential dependence on $1/\epsilon$ is inherent in how this and several other algorithms for other geometric problems work. In this paper, we present an EPTAS for knapsack for squares, i.e., a $(1+\epsilon)$-approximation algorithm with a running time of $O_\epsilon(1) \cdot n^{O(1)}$. In particular, the exponent of n in the running time does not depend on ϵ at all. Since there can be no FPTAS for the problem (unless $\mathsf{P} = \mathsf{NP}$, [6]) this is the best kind of approximation scheme we can hope for. To achieve this improvement, we introduce two new key ideas: We present a fast method to guess the $\Omega(2^{2^{1/\epsilon}})$ relatively large squares of a suitable near-optimal packing instead of using brute-force enumeration. Secondly, we introduce an *indirect guessing* framework to define sizes of cells for the remaining squares. In the previous PTAS each of these steps needs a running time of $\Omega(n^{2^{2^{1/\epsilon}}})$ and we improve both to $O_\epsilon(1) \cdot n^{O(1)}$.

2 Techniques

Our result builds on the existing PTAS for the problem given by Jansen and Solis-Oba [4]. This PTAS (and many other algorithms for geometric problems, e.g., [1, 5, 2]) applies the well-known shifting technique to distinguish squares into large and small squares. Eventually, it is shown that there is a $(1+\epsilon)$-approximative packing which is structured into $O_\epsilon(1)$ large squares and $O_\epsilon(1)$ cells. However, due to the shifting step both quantities can be $\Omega(2^{2^{1/\epsilon}})$. Roughly speaking, the PTAS in [4] then guesses the large squares and the sizes of the cells in time $O(n^{2^{2^{1/\epsilon}}})$ each. Then it assigns the small

*heydrich@mpi-inf.mpg.de. Max Planck Institute for Informatics and Graduate School of Computer Science, Saarland Informatics Campus, Germany. Supported in part by the *Google European Fellowship in Market Algorithms*.

†awiese@mpi-inf.mpg.de. University of Chile, Santiago de Chile, Chile. This research was carried out while the author was at the Max Planck Institute for Informatics.

squares to the cells via an instance of the generalized assignment problem [7] and finally packs large squares and cells into the knapsack. We need to improve the guessing of the large squares as well as the guessing of the cells in order to obtain an EPTAS.

Guessing large squares fast. Since there can be up to $\Omega(2^{2^{1/\epsilon}})$ large squares we do not want to simply enumerate over them since there are $n^{\Omega(2^{2^{1/\epsilon}})}$ possibilities. Instead, we show that by losing a factor $1+\epsilon$ in the approximation guarantee we can assume that the squares have only $O_\epsilon(\log n)$ different profits and that each profit class with large squares in the solution contributes at most $O_\epsilon(1)$ squares to the solution in total. Then, we can show that there are only $O_\epsilon(\log n)$ squares in total that can potentially be large. This reduces the guessing time to $(\log n)^{O_\epsilon(1)} = O_\epsilon(1) \cdot n^{O(1)}$.

Guessing cell decomposition via indirect guessing. Roughly speaking, there are two types of cells in the decomposition given by [4]. The first type are *block cells* that contain only squares that are much smaller than the cell itself (in both dimensions). For those we show that by losing a factor of $1+\epsilon$ we can round down their heights and widths to powers of $1+\epsilon$. This allows us to reduce the number of possibilities for those quantities to $O_\epsilon(\log n)$ and thus to $(\log n)^{O_\epsilon(1)}$ for all cells in parallel.

The other cells (*elongated cells*) are further structured and the most complicated parts are *row subframes.* Each of them has the property that its height equals the height of some input square and inside it squares are lined up next to each other but never on top of each other. Moreover, for the height of the row subframes there are only $O_\epsilon(1)$ (unknown) values $k_1, k_2, ...$ arising in the whole instance (as they stem from a harmonic grouping step). If we knew these values k_j then we would be essentially done. The PTAS in [4] just guesses them directly and *afterwards* assigns all squares into all subframes and all block cells. We cannot afford to guess all values k_j directly since a priori there are n possibilities for each of them and we cannot afford $n^{O_\epsilon(1)}$ guesses. Instead, we guess them *indirectly.* Intuitively, for a given value of x we ask ourselves: how much profit would we obtain from subframes of height k_1 if k_1 was equal to x? Let this value be described by the function $p_1(x)$. The larger x is, the more profit we could achieve since then more input squares would fit into these subframes. By losing a factor of $1+\epsilon$ in the objective it suffices to allow only values of x where this profit increases by a factor of $1+\epsilon$; we can find these values by inspecting the function $p_1(x)$. This allows us to iteratively guess the values k_j such that for each of them there are only $O_\epsilon(\log n)$ possibilities. In particular, this interlaces the guessing of the subframe heights with the packing of the squares, instead of doing one after the other like in [4].

If we only have one such elongated cell, we can directly compute the function $p_1(x)$ and find the candidate values for k_1 from it. However, in the general case, we need to compute an approximation for this function using an LP and a rounding scheme inspired by [3], as the problem of computing $p_1(x)$ is a generalization of the multi-dimensional knapsack problem.

References

[1] Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the two-dimensional geometric knapsack problem. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 1491–1505. SIAM, 2015.

[2] Nikhil Bansal, Alberto Caprara, Klaus Jansen, Lars Prädel, and Maxim Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *Algorithms and Computation (ISAAC 2009)*, volume 5878 of *LNCS*, pages 77–86. Springer, 2009.

[3] Wiebke Höhn, Julián Mestre, and Andreas Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 625–636. Springer, 2014.

[4] Klaus Jansen and Roberto Solis-Oba. A polynomial time approximation scheme for the square packing problem. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008, Bertinoro, Italy, May 26-28, 2008, Proceedings*, volume 5035 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2008.

[5] Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pages 204–213. SIAM, 2004.

[6] JYT Leung, TW Tam, CS Wong, GH Young, and FYL Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 1990.

[7] David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1-3):461–474, 1993.

On Minimizing the Makespan With Bag Constraints

Syamantak Das (Speaker) * Andreas Wiese †

1 Introduction

Minimizing the makespan is a classical problem in scheduling [2, 3]. Given a set of machines M and set of jobs J, we seek to assign each job to a machine. In the setting where all machines are *identical*, the processing time of each job j is given by a value p_j for each job j. For *unrelated* machines the processing time of a job j can depend on the machine i on which it is scheduled. In this case the input contains a value $p_{ij} \in \mathbb{R}_0^+ \cup \{\infty\}$ for each combination of a machine i and a job j. The objective is to minimize the makespan, i.e., the maximum load of a machine i which is the total processing time of jobs assigned to i. The problem is well-studied, for identical machines it is strongly NP-hard and there are PTASs [4] and even EPTASs, e.g., [5, 6]. For unrelated machines there is a 2-approximation algorithm due to Lenstra, Shmoys, and Tardos [7], an improvement to $2 - 1/m$ due to Shchepin and Vakhania [8], and a lower bound of $3/2 - \epsilon$ [7].

In practice, one often finds side constraints in addition to the above scheduling setting that make the problem harder. A typical constraint is that some jobs have to be assigned on different machines. For instance, on-board computers of aeroplanes typically have several CPUs (modeled as machines) and for system stability considerations some tasks need to be executed on different CPUs [1]. The idea is that if one CPU fails then the plane still continues to operate safely. To model this, in this paper we assume that the input jobs are partitioned into *bags* $J = B_1 \dot\cup B_2 \dot\cup ... \dot\cup B_b$ and that no two jobs from the same bags are allowed to be assigned on the same machine. We call these new requirements the *bag-constraints*. In this paper we study the problem of minimizing the makespan on identical and unrelated machines with bag-constraints.

1.1 Identical machines

The known (E)PTASs [5, 6, 4] for minimizing the makespan on identical machines follow the idea of enumerating the solution for large jobs, e.g., jobs that are larger than $\epsilon \cdot OPT$, and then adding the small jobs via a greedy algorithm. More precisely, one enumerates patterns for the large jobs that indicate how many large jobs of each size are assigned on each machine. The large jobs are then assigned according to these patterns and it does not matter which exact job is assigned to which slot of each pattern as long as the size of the slot is respected. In the case of bag-constraints this unfortunately does not work directly anymore. One can still enumerate the mentioned patterns and, with some

*syamanta@uni-bremen.de. Universität Bremen, Bremen, Germany.

†awiese@dii.uchile.cl. Departamento de Ingenieria Industrial, Universidad de Chile, Santiago de Chile, Chile.

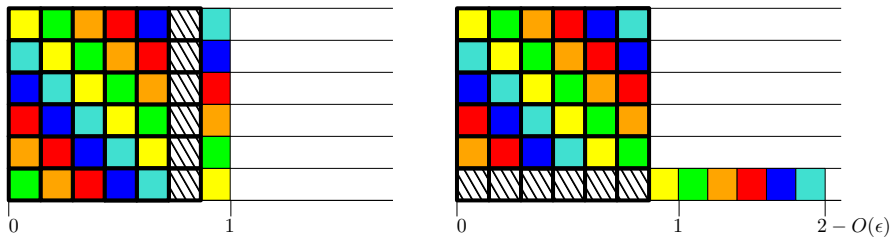


Figure 1: Left: an optimal schedule for the given instance. The bold lines indicate the enumerated pattern for the big jobs, all of them having size ϵ. The colors show the different bags of the jobs. Each white (striped) job j is in a (private) bag that contains only j. Right: a schedule in which the big jobs are assigned according to the same pattern but differently than in OPT. Thus, all non-big jobs have to be assigned to the last machine in order to satisfy the bag-constraints. This yields an approximation ratio of $2 - O(\epsilon)$.

reasonable additional effort, assign the large jobs to them such that they respect the bag-constraints. However, we cannot guarantee that the large jobs are assigned exactly like in the optimal solution. It could be that the jobs from the different bags are distributed completely differently on the machines than in the optimal solution (while still respecting the enumerated slots). In fact, there are instances for which the above procedure can lead to an assignment of the large jobs such that *any* solution for the remaining jobs has a makespan of at least $(2 - O(\epsilon))OPT$, see Figure 1 for an example.

Hence, we need additional ideas for the setting with bag-constraints. First, we observe that in the mentioned example many bags have relatively many large jobs (more than $\epsilon \cdot m$ many). There can be only $O_\epsilon(1)$ such *large bags* and hence we can afford to be more careful for them when we enumerate their large jobs. Indeed, we manage to assign the large jobs in such bags like some optimal solution. We assign the all other large jobs according to the enumerated pattern such that we respect the bag-constraints. To assign the remaining jobs, we split them into medium and small jobs such that the medium jobs have small total processing time (at most $\epsilon OPT \cdot m$). We find a way to assign the latter to the machines such that via some swapping and charging arguments we can guarantee that for the remaining small jobs there exists a solution with small overall makespan. For the small jobs the argumentation is again not as easy as without the bag-constraint since some machines already have jobs from some bags which prevents small jobs of such bags to be assigned to them. We solve the remaining problem with a combination of a non-trivial dynamic programming algorithm and a modified greedy routine.

Theorem 1 *There is a PTAS for minimizing the makespan on identical machine with bag-constraints.*

1.2 Unrelated machines

For makespan minimization on unrelated machines, there exists the LP-based 2- and $(2-1/m)$-approximation algorithms [7, 8]. The bag-constraints induce a linear constraint for each combination of a bag and a machine. Thus, it seems natural to enhance the normal LP by these constraints and try to adapt one of the known rounding algorithms.

However, in this paper show that this is deemed to fail. On the other hand, we show that a randomized rounding algorithm yields a $O(\log n / \log \log n)$-approximation.

Theorem 2 *For minimizing the makespan under restricted assignments with bag-constraints there can be no* $(\log n)^{1/4-\epsilon}$*-approximation algorithm for any* $\epsilon > 0$ *unless* $\mathsf{NP} \subseteq \mathsf{ZPTIME}(2^{(\log n)^{O(1)}})$.

Thus, in contrast to the case of identical machines we see here an increase in complexity due to the bag-constraints. However, we identify a special case of the restricted assignment setting where we can do better than in the general case: if all jobs from each bag can be assigned to exactly the same set of machines then we can give a 8-approximation algorithm based on the above mentioned LP. For this we need several novel ideas on top of the above mentioned known rounding techniques.

Theorem 3 *There is a* 8*-approximation algorithm for minimizing the makespan in the restricted assignment case if for each bag all jobs in the bag can be assigned to the same set of machines.*

References

[1] F. Eisenbrand, K. Kesavan, R. S. Mattikalli, M. Niemeier, A. W. Nordsieck, M. Skutella, J. Verschae, and A. Wiese. *Solving an Avionics Real-Time Scheduling Problem by Advanced IP-Methods*, pages 11–22. Springer, 2010.

[2] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[3] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[4] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

[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(2):457–485, 2010.

[6] K. Jansen, K.-M. Klein, and J. Verschae. Closing the Gap for Makespan Scheduling via Sparsification Techniques. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 72:1–72:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[7] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *SFCS '87: Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 217–224, Washington, DC, USA, 1987. IEEE Computer Society.

[8] E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33:127–133, 2005.

Hybrid Adaptive Particle Swarm Optimization Algorithm for the Permutation Flowshop Scheduling Problem

Yannis Marinakis (Speaker) * Magdalene Marinaki †

1 Introduction

The finding of suitable values for all parameters of a Particle Swarm Optimization (PSO) algorithm is a crucial issue in the design of the algorithm. A trial and error procedure is the most common way to find the parameters but, also, a number of different procedures have been applied in the past. In this paper, an adaptive strategy is used where random values are assigned in the initialization of the algorithm and, then, during the iterations, the parameters are optimized together and simultaneously with the optimization of the objective function of the problem. This approach is used for the solution of the Permutation Flowshop Scheduling Problem. The algorithm is tested in 120 benchmark instances and it is compared with a number of algorithms from the literature.

Particle Swarm Optimization (PSO) is a population-based swarm intelligence algorithm that was originally proposed by Kennedy and Eberhart [3]. In the literature and for the various variants of the PSO algorithm, authors have proposed different ways for calculating the main parameters of the algorithms. The most common way is to use the parameters that most researchers have used in the literature. Another way is to test for some instances a number of different sets of parameters, find the best values of the parameters for these instances and, then, use these values for the rest of the instances. Nowadays, a number of algorithms have been proposed for automated tunning of the parameters inside the procedure. Most of the papers are using an adaptive way to increase or decrease through the iterations the inertia factor, the acceleration coefficients or both. In all researches, the equations used to adapt the selected parameters are not the same, however, the main idea is the same. Another way to adapt some parameters (usually the inertia weight) is by using a fuzzy system. Other more efficient adaptive strategies where, also, the population size is adapted, have been presented.

In this paper, a new adaptive strategy denoted as **Hybrid Adaptive Particle Swarm Optimization (HAPSO)** algorithm is presented. All parameters of the Particle Swarm Optimization algorithm (acceleration coefficients, iterations, local search iterations, upper and lower bounds of the velocities and of the positions and number of particles) are optimized during the procedure and, thus, the algorithm works independently and without any interference from the user. All parameters are randomly initialized and, afterwards, during the iterations the parameters are adapted using three

*marinakis@ergasya.tuc.gr. School of Production Engineering and Management, Technical University of Crete, Greece.

†magda@dssl.tuc.gr. School of Production Engineering and Management, Technical University of Crete, Greece.

different conditions, the first is used for all parameters except the number of particles, the second is used for the increase of the number of particles and the third is used for the decrease of the number of particles. We applied the idea in a classic Constriction Particle Swarm Optimization algorithm [1].

The proposed algorithm is tested for the solution of the Permutation Flowshop Scheduling Problem (PFSP) [2, 5]. The selection was made for a number of reasons. One reason is that it is an interesting NP-hard problem, another reason is that there is a number of benchmark instances in the literature that could be used for testing the algorithm and as the main subject of this paper is the experimental analysis of the algorithm in a problem with as many instances as possible, the existence of a benchmark set of 120 instances it gave us the opportunity to test our algorithm in data sets with varying number of variables and, thus, to see the performance of the algorithm in instances with different sizes. The third reason is that we have published in the past a paper [4] for the same problem with PSO variants and parameters of the algorithms calculated in the beginning of the process and, thus, it would be interesting to make comparisons with these approaches. Finally, there is a large number of publications in the literature for solving this problem with heuristic, metaheuristic and evolutionary algorithms and, thus, it was easy to compare our results with the results of other algorithms from the literature.

2 Results

The algorithm (implemented in Fortran 90) was tested on the 120 benchmark instances of Taillard. In these instances, there are different sets having 20, 50, 100, 200 and 500 jobs and 5, 10 or 20 machines. There are 10 problems inside every size set. In total, there are 12 sets and these are: 20×5 (i.e. 20 jobs and 5 machines), 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20 and 500×20. The efficiency of the HAPSO algorithm is measured by the quality of the produced solutions. The quality is given in terms of the relative deviation from the best known solution, that is $\omega = \frac{(c_{HAPSO}-c_{BKS})}{c_{BKS}}\%$, where c_{HAPSO} denotes the cost of the solution found by HAPSO and c_{BKS} is the cost of the best known solution. To test the performance of the proposed algorithm we applied HAPSO (and the other algorithms used in the comparisons) 10 times to each test instance.

In Table 1, except of the results of the proposed algorithm, the results of other five algorithms based on Particle Swarm Optimization are presented. All algorithms are variants of PSO with local and global neighborhood topologies and they all presented and analyzed in [4]. In Table 1, the average quality (Q) and the average CPU times (T) in seconds of the proposed algorithm and of the other five algorithms are presented. In Table 2, a comparison with other algorithms from the literature is performed. There is a number of heuristic and metaheuristic algorithms that have been applied for finding of the makespan in a PFSP. Table 2 presents the average quality of the solutions of the proposed algorithm (HAPSO) and the average quality of other 11 algorithms from the literature. The algorithms are: a variant of the classic NEH algorithm (NEHT), five genetic algorithms (SGA, MGGA, ACGA, GA-VNS and SGGA), three versions of the Particle Swarm Optimization (PSO1, CPSO, PSOENT) and a Differential Evolution (DDE) algorithm. For more informations about these algorithms please see [4]. From these tables we can see that the proposed algorithm gives very satisfactory and, in most

Table 1: Comparisons of the results (average qualities) of HAPSO with the other five algorithms in Taillard benchmark instances for the PFSP

	HybPSO		RGPSO		LNPSO		RLNPSO		PSOENT		HAPSO	
	Q	T	Q	T	Q	T	Q	T	Q	T	Q	T
20 × 5	**0.00**	4.15	**0.00**	4.08	**0.00**	4.05	**0.00**	4.02	**0.00**	3.45	**0.00**	*1.58*
20 × 10	0.15	15.15	0.15	18.12	0.08	16.35	0.11	15.5	**0.07**	15.25	0.09	*11.21*
20 × 20	0.31	24.85	0.08	24.5	0.09	24.65	0.11	25.15	0.08	24.52	**0.07**	*18.35*
50 × 5	0.20	8.25	0.07	7.45	0.05	7.52	0.06	7.39	**0.02**	6.15	0.05	*5.24*
50 × 10	2.20	23.45	2.34	23.54	2.05	24.18	2.29	26.21	2.11	23.55	**2.01**	*20.15*
50 × 20	3.81	45.28	3.59	44.15	3.50	46.5	3.51	48.5	3.83	44.25	**3.20**	*39.22*
100 × 5	0.19	22.15	0.13	22.58	0.11	23.15	0.11	24.18	**0.09**	22.85	0.14	*18.12*
100 × 10	1.33	65.25	1.21	64.35	1.23	62.28	1.29	61.28	1.26	60.35	**1.17**	*51.11*
100 × 20	4.36	125.35	4.37	135.48	4.55	140.24	4.21	139.28	4.37	131.15	**4.13**	*118.21*
200 × 10	1.37	122.85	1.12	125.48	1.15	126.35	1.20	125.44	**1.02**	124.18	1.06	*111.21*
200 × 20	4.62	258.46	4.37	255.24	4.45	259.35	4.66	261.28	**4.27**	255.42	**4.27**	*242.34*
500 × 20	3.21	410.18	2.95	408.25	2.74	415.35	2.97	412.24	**2.73**	409.54	3.43	*380.31*
Average	1.81	93.78	1.70	94.43	1.67	95.83	1.71	95.87	1.65	93.38	**1.63**	*84.75*

cases, better results compared either with other PSO algorithms or with algorithms from the literature.

Table 2: Comparisons of the results (average qualities) of HAPSO with other algorithms from the literature in Taillard benchmark instances for the PFSP

	HAPSO	PSOENT	NEHT	SGA	MGGA	ACGA	SGGA	DDE	CPSO	GMA	PSO2	GA-VNS
20 × 5	0.00	0.00	3.35	1.02	0.81	1.08	1.1	0.46	1.05	1.14	1.25	0
20 × 10	0.09	0.07	5.02	1.73	1.4	1.62	1.9	0.93	2.42	2.3	2.17	0
20 × 20	0.07	0.08	3.73	1.48	1.06	1.34	1.6	0.79	1.99	2.01	2.09	0
50 × 5	0.05	0.02	0.84	0.61	0.44	0.57	0.52	0.17	0.9	0.47	0.47	0
50 × 10	2.01	2.11	5.12	2.81	2.56	2.79	2.74	2.26	4.85	3.21	3.6	0.77
50 × 20	3.20	3.83	6.26	3.98	3.82	3.75	3.94	3.11	6.4	4.97	4.84	0.96
100 × 5	0.14	0.09	0.46	0.47	0.41	0.44	0.38	0.08	0.74	0.42	0.35	0
100 × 10	1.17	1.26	2.13	1.67	1.5	1.71	1.6	0.94	2.94	1.96	1.78	0.08
100 × 20	4.13	4.37	5.23	3.8	3.15	3.47	3.51	3.24	7.11	4.68	5.13	1.31
200 × 10	1.06	1.02	1.43	0.94	0.92	0.94	0.8	0.55	2.17	1.1	-	0.11
200 × 20	4.27	4.27	4.41	2.73	3.95	2.61	2.32	2.61	6.89	3.61	-	1.17
500 × 20	3.43	2.73	2.24	-	-	-	-	-	-	-	-	0.63
Average	1.63	1.65	3.35	1.93	1.82	1.84	1.85	1.37	3.40	2.35	2.40	0.4

References

[1] M. Clerc and J. Kennedy (2002). The particle swarm: explosion, stability and convergence in a multi-dimensional complex space, *IEEE Transactions on Evolutionary Computation*, 6, 58-73.

[2] S. Johnson (1954). Optimal two-and-three stage production schedules with setup times included, *Naval Research Logistics Quarterly*, 1, 61-68,

[3] J. Kennedy and R. Eberhart (1995). Particle swarm optimization, *Proceedings of 1995 IEEE International Conference on Neural Networks*, 4, 1942-1948.

[4] Y. Marinakis and M. Marinaki (2013). Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem, *Soft Computing*, 17 (7), 1159-1173.

[5] M. Pinedo (1995). *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, New Jersey.

On Scheduling Coflows*

Saba Ahmadi Samir Khuller (Speaker) Manish Purohit
Sheng Yang

1 Introduction

Chowdhury and Stoica [3] introduced coflows as an effective abstraction to represent the collective communication requirements of a job. We consider the problem of scheduling coflows to minimize weighted completion time and give improved approximation algorithms for this basic problem.

The communication phase for a typical application in a modern data center may contain hundreds of individual flow requests, and the phase ends only when all of these flow requests are satisfied. A coflow is defined as the collection of these individual flow requests that all share a common performance goal. The underlying data center is modeled as a single $m \times m$ *non-blocking switch* that consists of m input ports and m output ports. We assume that each port has unit capacity, i.e. it can handle at most one unit of data per unit time. Modeling the data center itself as a simple switch allows us to focus solely on the scheduling task instead of the problem of *routing* flows through the network. Each coflow j is represented as a $m \times m$ integer matrix $D^j = [d^j_{io}]$ where the entry d^j_{io} indicates the number of data units that must be transferred from input port i to output port o for coflow j. Figure 1 shows a single coflow over a 2×2 switch. For instance, the coflow depicted needs to transfer 2 units of data from input a to output b and 3 units of data from input a to output d. Each coflow j also has a weight w_j that indicates its relative importance and a release time r_j.

A coflow j is available to be scheduled at its release time r_j and is said to be completed when all the flows in the matrix D^j have been scheduled. More formally, the completion time C_j of coflow j is defined as the earliest time such that for every input i and output o, d^j_{io} units of its data have been transferred from port i to port o. We assume that time is slotted and data transfer within the switch is instantaneous. Since each port can process at most one unit of data in each time slot, a feasible schedule for a single time slot can be described as a matching. Our goal is to find a feasible scheduling that minimizes the total, weighted completion time of the coflows, i.e. minimize $\sum_j w_j C_j$.

Related Work: Although coflow aware network schedulers have been found to perform very well in practice in both the offline [5] and online [4] settings, no $O(1)$ approximation algorithms were known even in the offline setting until recently. Since the coflow scheduling problem generalizes the well-studied concurrent open shop scheduling problem, it is NP-hard to approximate with a factor better than $(2 - \epsilon)$ [1, 11].

*University of Maryland, College Park MD 201742, USA.

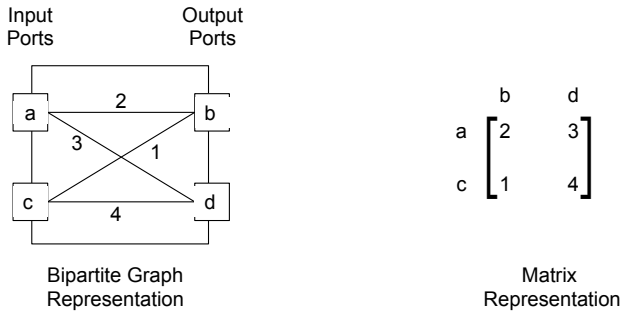


Figure 1: An example coflow over a 2×2 switch. The figure illustrates two equivalent representations of a coflow - (i) as a weighted, bipartite graph over the set of ports, and (ii) as a $m \times m$ integer matrix.

For the special case when all coflows have zero release time, Qiu, Stein and Zhong [10] obtain a deterministic $\frac{64}{3}$ approximation and a randomized $(8 + \frac{16\sqrt{2}}{3})$ approximation algorithm for the problem of minimizing the weighted completion time. For coflow scheduling with arbitrary release times, Qiu et al. [10] claim a deterministic $\frac{67}{3}$ approximation and a randomized $(9 + \frac{16\sqrt{2}}{3})$ approximation algorithm. We demonstrate a subtle error in their proof that deals with non-zero release times. We show that their techniques in fact only yield a deterministic $\frac{76}{3}$-approximation algorithm for coflow scheduling with release times. However their result holds for the case with no release times.

By exploiting a connection with the concurrent open shop scheduling problem, Luo et al. [8] claim a 2-approximation algorithm for coflow scheduling when all the release times are zero. Unfortunately, their proof too is flawed and the result does not hold.

2 Our Contributions and Techniques

The main algorithmic contribution of our paper is a combinatorial algorithm for the offline coflow scheduling problem with improved approximation guarantees.

Theorem 1 *There exists a deterministic, combinatorial, polynomial time 5-approximation algorithm for coflow scheduling with release times.*

Theorem 2 *There exists a deterministic, combinatorial, polynomial time 4-approximation algorithm for coflow scheduling without release times.*

The first stage of our algorithm exploits the connection between the well-studied concurrent open shop scheduling [9, 2] (also called as order scheduling [7]) and the coflow scheduling problem. In the concurrent open shop problem, we have a set of m machines and each job j with weight w_j is composed of m tasks $\{t_i^j\}_{i=1}^m$, one on each machine. A job j is said to be completed once all its tasks have completed. A machine can perform at most one unit of processing at a time and the goal is to find a feasible schedule that minimizes the total weighted completion time of jobs. Indeed, the concurrent open shop scheduling problem can be viewed as a special case of coflow scheduling when all the demand matrices D^j are diagonal [5]. At a first glance, it appears that coflow scheduling is much harder than concurrent open shop. Surprisingly, we show

that via a similar LP relaxation as for the concurrent open shop problem, we can design a primal-dual algorithm to obtain a good permutation of the coflows. Our primal-dual algorithm is inspired by Davis et al. [6] and Mastrolilli et al. [9]. As a side effect, our algorithm also gives the first known *combinatorial* 3-approximation for concurrent open shop scheduling with release times.

However, simply obtaining a good permutation of the coflows is not sufficient to obtain a good schedule as sequentially scheduling these coflows independently can lead to large completion times for the later coflows. In the second stage of our algorithm, we *move* some flows from coflows that are later in the permutation to an earlier coflow. By carefully moving only those flows that would not delay the total completion time of the preceding coflow, we show that sequentially scheduling these postprocessed coflows leads to a provably good coflow schedule.

References

[1] N. Bansal and S. Khot. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In *ICALP*, pages 250–261. Springer, 2010.

[2] Z.-L. Chen and N. G. Hall. Supply chain scheduling: Conflict and cooperation in assembly systems. *Operations Research*, 55(6):1072–1089, 2007.

[3] M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In *ACM Workshop on Hot Topics in Networks*, pages 31–36. ACM, 2012.

[4] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *SIGCOMM*, pages 393–406. ACM, 2015.

[5] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *SIGCOMM*, SIGCOMM '14, pages 443–454, New York, NY, USA, 2014. ACM.

[6] J. M. Davis, R. Gandhi, and V. H. Kothari. Combinatorial algorithms for minimizing the weighted sum of completion times on a single machine. *Operations Research Letters*, 41(2):121–125, 2013.

[7] N. Garg, A. Kumar, and V. Pandit. Order scheduling models: Hardness and algorithms. In *FSTTCS*, pages 96–107. Springer, 2007.

[8] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li. Towards practical and near-optimal coflow scheduling for data center networks. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1, 2016.

[9] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010.

[10] Z. Qiu, C. Stein, and Y. Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *SPAA*, SPAA '15, pages 294–303, New York, NY, USA, 2015. ACM.

[11] S. Sachdeva and R. Saket. Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover. In *IEEE Conference on Computational Complexity*, pages 219–229. IEEE, 2013.

A Branch and Bound Approach for Single Machine Scheduling in the Automotive Supply Chain

Paul Göpfert (Speaker) * Stefan Bock †

1 The Scheduling Problem

In this talk, we consider a single machine scheduling problem that can be classified by the three field notation of [4] as follows:

$$1|setup, rm, chains, \delta_j| \sum w_j C_j. \tag{1}$$

The scheduling problem is 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 set of *chain*-based job-precedence relations. As the supplier has to guarantee a timely delivery, deadlines (δ_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 deadlines [7] or the need for availability of raw materials [3] are proven to be $\mathcal{NP}$-hard, this also applies to the extended 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.

2 A Branch and Bound Approach

The aim of our research is the development of an efficient Branch and Bound procedure for this problem. The enumeration process is done in forward direction, in every new node of the tree a further job is added to the partial schedule represented by the currently branched node. Dominance criteria and tests for the fulfillment of necessary conditions for the existence of a feasible extension to a complete schedule speed up the enumeration process.

For the purpose of the calculation of lower bounds, we may resort on subproblems included in this scheduling problem. Branch and Bound approaches from the literature for problems such as $1|\delta_j|\sum w_jC_j$ ([9]), $1|r_j|\sum w_jC_j$ ([1]) and $1|r_j,\delta_j|\sum w_jC_j$ ([8]) therefore provide lower bounds also for the scheduling problem under consideration.

The chain restrictions together with the time windows resulting from the material availability limitations and deadlines allow us to derive an arbitrary precedence graph G. As the scheduling problem $1|prec|\sum w_jC_j$ with arbitrary precedences is $\mathcal{NP}$-hard ([6]), we may seek for polynomial-time solvable classes of precedence relations contained in G to obtain further bounds for the problem. An example for this technique is the construction of a vertex-series-parallel (VSP) suborder, that allows us to use the $\mathcal{O}(n \log n)$-polynomial-time algorithm of Lawler [6] for this class of precedence relations. This idea is also mentioned in [2], whereas we take another way to generate the needed VSP-suborder.

Another possibility to obtain a lower bound for scheduling problems with an total cost objective such as $\sum(w_j)C_j$ or $\sum(w_j)T_j$ is the calculation of an estimate $C^{LB}_{j,p}$ on the completion time for every job $j \in J$, if job j is placed on position $p \in 1,\ldots|J|$ of the schedule. The application of these estimates in order to calculate a lower bound by consideration of the resulting linear assignment problem has already been proposed by [5].

All the lower bounds mentioned so far differ in the addressed properties of the problem, the effort for their calculation and their quality.

We investigate the impact of the different bounding techniques on the overall efficiency of the Branch and Bound procedure for the considered optimization problem. Results on artificial instances with up to 75 jobs and 15 products as well as different settings for time windows and setup times are presented.

References

[1] H. Belouadah, M. E. Posner, C. N. Potts (1992) Scheduling with release dates on a single machine to minimize total weighted completion time, *Discrete Applied Mathematics*, Vol. 36, pp. 213-231.

[2] M. Davari, E. Demeulemeester, R. Leus, F. Talla Nobibon (2015) Exact algorithms for single-machine scheduling with time windows and precedence constraints, *Journal of Scheduling*, pp. 1-26.

[3] A. Grigoriev, M. Holthuijsen and J. van de Klundert (2005). Basic Scheduling Problems with Raw Material Constraints, *Naval Research Logistics*, Vol. 52, pp. 527-535.

[4] 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.

[5] A. H. G. Rinnooy Kan, B. J. Lageweg and J. K. Lenstra (1975). Minimizing Total Costs in One-Machine Scheduling *Operations Research*, Vol. 23, pp. 908-927.

[6] E. L. Lawler (1978). Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints, *Annals of Discrete Mathematics*, Vol. 2, pp. 75-90.

[7] J. K. Lenstra, and A. H. G. Rinnooy Kan, P. Brucker (1977). Complexity of Machine Scheduling Problems, *Annals of Discrete Mathematics*, Vol. 1, pp. 343362.

[8] Y. Pan, L. Shi (2005). Dual constrained single machine sequencing to minimize total weighted completion time, *IEEE Transactions on Automation Science and Engineering*, Vol. 2, pp. 344-357.

[9] M. E. Posner (1985). Minimizing Weighted Completion Times with Deadlines, *Operations Research*, Vol. 33, pp. 562-574.

Delayed-Clairvoyant Scheduling

Sorrachai Yingchareonthawornchai (Speaker) * Eric Torng †

1 Introduction

In this paper, we introduce a new model of partial clairvoyance which we call *delayed clairvoyance* for online scheduling. In traditional clairvoyant scheduling, an online scheduler learns a job's processing time once it is released. In nonclairvoyant scheduling, an online scheduler only learns a job's processing time once it is completed. Bender *et al.* introduced an intermediate model of clairvoyance which they called "semi-clairvoyant scheduling" where the online scheduler learns the class or approximate processing time of a job once it is released and only learns the exact processing time once it is completed [2].

In delayed clairvoyance, an online scheduler is initially nonclairvoyant and becomes clairvoyant once a job has been processed for αp_i time where $0 \leq \alpha \leq 1$. That is, once a job has been processed for an α fraction of its processing time, the online scheduler learns its processing time, and we do not assume that the online scheduler knows α. Delayed clairvoyance with $\alpha = 0$ is equivalent to clairvoyance, and delayed clairvoyance with $\alpha = 1$ is equivalent to nonclairvoyance. Delayed clairvoyance can be generalized in several ways. For example, each job could have its own α_i parameter. That said, we start with a single α parameter for all jobs and assume the scheduler is nonclairvoyant initially.

We perform a very preliminary study of delayed clairvoyance using the problem of online scheduling on a single machine with the goal of minimizing total flow time. For this problem, for clairvoyant scheduling, it is well known that the Shortest Remaining Processing Time (SRPT) algorithm is optimal. On the other hand, no nonclairvoyant scheduling algorithm can be $O(1)$-competitive [4]. This is because the nonclairvoyant scheduler can fall behind by processing the wrong jobs and thus builds up too many extra jobs compared to the optimal algorithm. For semi-clairvoyant scheduling, Bender *et al.* proved the surprising result that SRPT is not $O(1)$-competitive [2], and Becchetti *et al.* later proved that a modified algorithm is $O(1)$-competitive for semi-clairvoyant scheduling [1].

Given that SRPT is optimal when $\alpha = 0$ and no online algorithm is $O(1)$-competitive when $\alpha = 1$, the natural question is for what value of α does this problem become hard. Is it hard for any $\alpha > 0$? Is there a threshold where we can be constant competitive and then suddenly not?

We make initial progress towards answering this question. We consider the following algorithmic framework. At any time t, let $A(t)$ denote the remaining jobs. We divide

*yingchar@cse.msu.edu. Department of Computer Science and Enginereing, Michigan State Univeristy, East Lansing, MI, USA.

†torng@cse.msu.edu. Department of Computer Science and Enginereing, Michigan State Univeristy, East Lansing, MI, USA.

$A(t)$ into two classes: *early jobs* where we do not know their processing times and *late jobs* where we do know their processing times. Let $e(t)$ and $l(t)$ denote the number of early jobs and late jobs, respectively, at time t. We focus on algorithms that apply $e(t)/|A(t)|$ fraction of the processor to early jobs and $l(t)/|A(t)|$ fraction of the processor to late jobs. For the late jobs, we use SRPT. For the early jobs, we use a nonclairvoyant algorithm. In this paper, we consider round robin (RR). We realize this is not the best nonclairvoyant algorithm and expect to have better results with better nonclairvoyant algorithms by the time of MAPSP.

The resulting algorithm we study is RR+SRPT. We then use the following notation. For any time t, let $A_{RR}(t)$ and $A_C(t)$ be the set of early jobs and late jobs, respectively. Note that $A(t) = A_{RR}(t) \sqcup A_C(t)$, where $\sqcup$ is disjoint union.

Proposition 1. *RR+SRPT is $\Omega(n^{2-\frac{1}{\alpha}})$-competitive when $\alpha \geq 0.5$.*

Proof Sketch. Consider only RR with α fraction of jobs' processing time. This model is equivalent to speed augmentation where RR' has speed $\frac{1}{\alpha}$ with full jobs' processing time. It is known that there exists an instance such that RR is $\Omega(n^{2-s})$-competitive with speed $s < 2$ (Theorem 2.19 in [5]). The special model for RR cannot handle this instance, and hence RR+SRPT cannot be competitive. □

In this work, we then focus on finding a smaller α so that SPRT can catch up. We set $\alpha < 0.25$ and show that it is indeed possible. Open question is whether or not RR+SPRT is $O(1)$-competitive for $\alpha \in [0.25, 0.5)$.

Nonclairvoyant jobs have initial processing time to be $\alpha p_i(t)$ by definition of our model. Let RR' be a RR algorithm with $\frac{1}{\alpha}$ speed and job's processing time is $p_i(t)$. The following key Lemma is useful in our main result, which can be easily verified.

Lemma 2. *At any time t, $|A_{RR'}(t)| = |A_{RR}(t)|$*

Theorem 3. *RR+SRPT is $O(1)$-competitive when $\alpha < 0.25$.*

Proof. We consider RR'+SRPT where RR' has speed $\frac{1}{\alpha}$ but works with full size of job. After RR' finished processing, such job will be processed with SRPT with ordinary speed with processing time of $1-\alpha$ fraction. By Lemma 2, the competitiveness of RR'+SRPT implies RR+SRPT.

Let $\alpha = \frac{1}{4+\epsilon}$ for $\epsilon > 0$.Denote $p_i^a(t), p_i^o(t)$ as remaining processing time of job i in the algorithm RR'+SRPT and optimal algorithm, OPT, respectively. Let $z_i(t) = \max(p_i^a(t) - p_i^o(t), 0)$ be the lag behind OPT. We give the following potential function. A detailed potential function analysis technique can be found at [3].

$$\Phi(t) = \frac{4}{\epsilon} \sum_{J_i, J_j \in A_{RR}(t)} \min(z_i(t), z_j(t)) + 8(1 + \frac{1}{\epsilon}) \sum_{\substack{J_i \in A_C(t) \\ \text{increasing order}}} (n - i + 1) z_i(t)$$

Job Arrival Condition. There is no change in potential function.
Job Completion Condition. When RR finishes job J_i on its queue, J_i's remaining processing time of $1-\alpha$ fraction is revealed to SRPT's queue. In this case, the potential will increase by the rank of J_i relative to the remaining size of the queue in SRPT. One can show that total change of such increase is at most $(1-\alpha)OPT$.

Running Condition. We consider the change in potential function due to RR's processing (We denote RR' as RR for convenience). There are $|A_{RR}(t)|^2$ terms in the first

term of $\Phi(t)$ where each term is processing with speed $(4+\epsilon)/A(t)$. So the change due to RR is at most $-\frac{4(4+\epsilon)}{\epsilon}\frac{|A_{RR}(t)|^2}{|A(t)|}$. We will consider the change of potential function due to $SPRT$'s processing later when necessary. Now we consider the change of $\Phi(t)$ due to OPT's processing. Without loss of generality, assume that OPT processes one process at a time. Hence, the potential function will increase by either first term or second term. We consider the maximum of both. That is, for the first term in $\Phi(t)$, OPT will affect at most $2|A_{RR}(t)|$ terms. So, $\Phi(t)$ will increase by $\frac{8}{\epsilon}|A_{RR}(t)|$. For second term, OPT will process on the least remaining jobs in $OPT(t)$. Hence, the change is $8(1+\frac{1}{\epsilon})|A_C(t) \cap OPT(t)|$. The total change due to OPT is then $\max\{\frac{8}{\epsilon}|A_{RR}(t)|, 8(1+\frac{1}{\epsilon})|A_C(t) \cap OPT(t)|\}$ We consider three cases. Case (1): if $|A_{RR}(t)| \geq A(t)/2$. We have that the change due to RR's processing is $-\frac{4(4+\epsilon)}{\epsilon}\frac{|A_{RR}(t)|^2}{|A(t)|} \leq -\frac{2(4+\epsilon)}{\epsilon}|A_{RR}(t)|$. The change due to OPT's processing in this case is $\max\{\frac{8}{\epsilon}|A_{RR}(t)|, 8(1+\frac{1}{\epsilon})|A_C(t) \cap OPT(t)|\} \leq \frac{8}{\epsilon}|A_{RR}(t)| + 8(1+\frac{1}{\epsilon})|OPT(t)|$. Hence, the change of $\Phi(t)$ is at most

$$\frac{d\Phi(t)}{dt} \leq -\frac{2(4+\epsilon)}{\epsilon}|A_{RR}(t)| + \frac{8}{\epsilon}|A_{RR}(t)| + 8(1+\frac{1}{\epsilon})|OPT(t)|$$
$$\leq -2|A_{RR}(t)| + 8(1+\frac{1}{\epsilon})|OPT(t)| \leq -|A(t)| + 8(1+\frac{1}{\epsilon})|OPT(t)|$$

Hence, in case (1) we have: $|A(t)| + \frac{d\Phi(t)}{dt} \leq 8(1+\frac{1}{\epsilon})|OPT(t)|$

We omit the detail of case (2) and (3), but provide high level ideas. Case (2): if $|A_C(t)| \geq 3|A(t)|/4$ and $|A_{RR}(t)| \leq |A(t)|/4$. The case is similar to case (1), but we use only change due to SRPT's processing against OPT. SRPT can catch up OPT because the amount of processing power is at least $3/4$ which is a constant factor. We can show that $|A(t)| + \frac{d\Phi(t)}{dt} \leq 8(1+\frac{1}{\epsilon})|OPT(t)|$.

Case (3) is $|A(t)|/4 \leq |A_{RR}(t)| \leq |A(t)|/2$. The crux is that we need to use both RR and SRPT to pay for the potential function to obtain the same bound. This step requires more calculation and using appropriate inequalities. $|A(t)| + \frac{d\Phi(t)}{dt} \leq 8(1+\frac{1}{\epsilon})|OPT(t)|$.

Totaling from arrival condition, completion and running condition we have

$$|A(t)| + \frac{d\Phi(t)}{dt} \leq (1 - \frac{1}{4\epsilon+1} + 8(1+\frac{1}{\epsilon}))|OPT(t)| = O(1)|OPT(t)|$$

Finally, the result follows from integrating both side from $t = 0$ to ∞. □

References

[1] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs. Semi-clairvoyant scheduling. *Theoretical Computer Science*, 324(2):325 – 335, 2004.

[2] M. A. Bender, S. Muthukrishnan, and R. Rajaraman. Improved algorithms for stretch scheduling. SODA '02, pages 762–771, 2002.

[3] S. Im, B. Moseley, and K. Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42(2):83–97, June 2011.

[4] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. SODA '93, pages 422–431, Philadelphia, PA, USA, 1993.

[5] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. pages 115–124. CRC Press, 2003.

Schedulability Analysis of Dependent Probabilistic Real-Time Tasks

Slim Ben-Amor (Speaker) * Dorin Maxim † Liliana Cucu-Grosjean ‡

1 Introduction

The complexity of modern architectures has increased the timing variability of programs (or tasks). New probabilistic approaches decrease the resulting pessimism by associating probabilities to different worst case execution time values of the programs (tasks). In this paper, we extend the original work of Chetto et al. [3] on precedence constrainted tasks by describing the worst case execution times by probability distributions. We provide probabilistic schedulability conditions that decrease pessimism due to worst case reasoning on highly variable systems.

2 Definition

Definition 1 *Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be two independent probability distributions and $\mathcal{Z} = \mathsf{Max}(\mathcal{X}_1, \mathcal{X}_2)$*

$$\begin{aligned} p(\mathcal{Z} \leqslant t) &= p(\mathsf{Max}(\mathcal{X}_1, \mathcal{X}_2) \leqslant t) \\ &= p(\mathcal{X}_1 \leqslant t, \mathcal{X}_2 \leqslant t) \\ &= p(\mathcal{X}_1 \leqslant t) p(\mathcal{X}_2 \leqslant t) \\ &= \sum_{i=min(\mathcal{X}_1)}^{t} p(\mathcal{X}_1 = i) \sum_{j=min(\mathcal{X}_2)}^{t} p(\mathcal{X}_2 = j) \end{aligned}$$

If $\mathcal{X}_1$ and $\mathcal{X}_2$ are finite discrete distributions, then we obtain

$$p(\mathcal{Z} = t) = \sum_{max(i,j)=t} p(\mathcal{X}_1 = i) p(\mathcal{X}_2 = j)$$

We note that $\mathsf{Max}(\mathcal{X}_1, \mathcal{X}_2) \succeq \mathcal{X}_i$, $\forall i \in \{1, 2\}$ with $\succeq$ as defined in [2].
Similarly, we define the minimum distribution $\mathcal{Z} = \mathsf{Min}(\mathcal{X}_1, \mathcal{X}_2)$ with

$$p(\mathcal{Z} = t) = \sum_{min(i,j)=t} p(\mathcal{X}_1 = i) p(\mathcal{X}_2 = j)$$

We note that $\mathcal{X}_i \succeq \mathsf{Min}(\mathcal{X}_1, \mathcal{X}_2)$, $\forall i \in \{1, 2\}$.

*INRIA 2, rue Simone Iff Paris, France slim.ben-amor@inria.fr
†LORIA - University of Lorraine Nancy, France dorin.maxim@inria.fr
‡INRIA 2, rue Simone Iff Paris, France liliana.cucu@inria.fr

3 Motivational Example: The Non-applicability of the Diaz relation

A natural candidate for replacing the comparison relation $\leq$ in Chettos' schedulability test is the comparison relation $\succeq$ introduced by Diaz et al. [2]. We provide an example of the non-applicability of this operator to the schedulability analysis of a task set. Let $\tau = \{\tau_1\}$ be a task set with a single task τ_1 defined by

$$(\mathcal{C}_1 = \begin{pmatrix} 1 & 3 \\ 0.9 & 0.1 \end{pmatrix}, \mathcal{D}_1 = \begin{pmatrix} 2 & 4 \\ 0.8 & 0.2 \end{pmatrix}).$$

In the deterministic worst case for $C_1 = 3$ and $D_1 = 2$, we conclude that the task set is not schedulable ($C_1 > D_1$). Nevertheless, if we compare the CDFs $F_{\mathcal{C}_1}(t) \geq F_{\mathcal{D}_1}(t), \forall\, t \in \mathbb{R}$, then $\mathcal{D}_1 \succeq \mathcal{C}_1$ implying the task set is schedulable. In conclusion, the Diaz relation cannot replace the relation $\leq$ in Chetto schedulability test. Thus, we propose a new relation in Section 3.1 that justifies Definition 1.

3.1 A new relation of comparison between two probability distributions

Based on the conclusion of the previous section we propose a new relation to compare two probability distributions in Chetto schedulability test as follows:

$$p(\mathcal{X}_1 \leqslant \mathcal{X}_2) = \int_{-\infty}^{\infty} p(\mathcal{X}_1 \leqslant t)p(\mathcal{X}_2 \geqslant t)dt = \sum_{t=min(\mathcal{X}_1)}^{max(\mathcal{X}_2)} p(\mathcal{X}_1 \leqslant t)p(\mathcal{X}_2 \geqslant t)$$
$$= \sum_{t=min(\mathcal{X}_1)}^{max(\mathcal{X}_2)} \sum_{i=min(\mathcal{X}_1)}^{t} p(\mathcal{X}_1 = i) \sum_{j=t}^{max(\mathcal{X}_2)} p(\mathcal{X}_2 = j) = \sum_{i \leqslant j} p(\mathcal{X}_1 = i)p(\mathcal{X}_2 = j)$$

Thus, the probabilistic comparison operation is given by:

$$\mathcal{X}_1 \leqslant \mathcal{X}_2 = \begin{pmatrix} \mathcal{X}_1 \leqslant \mathcal{X}_2 & \mathcal{X}_1 > \mathcal{X}_2 \\ \sum\limits_{i \leqslant j} p(\mathcal{X}_1 = i)p(\mathcal{X}_2 = j) & 1 - \sum\limits_{i \leqslant j} p(\mathcal{X}_1 = i)p(\mathcal{X}_2 = j) \end{pmatrix} \tag{1}$$

By applying the proposed comparison to the task set $\tau = \{\tau_1\}$ with τ_1 defined in Section 3, then the task set is schedulable with an associated probability 92%. A non-probabilistic schedulability analysis declares the system non-schedulable while its probability of meeting the deadline is high.

4 Probabilistic model

In this section we propose a probabilistic description of the model of a task τ_i defined by $\mathcal{R}_i, \mathcal{C}_i, \mathcal{D}_i$. Our probabilistic description considers the release times, the worst case execution times and the deadlines described by discrete probability distributions and thus a probabilistic task τ_i.

We provide a probabilistic formulation of Chetto equations as follows. Let τ^* be the task system obtained as follows:

$$\mathcal{R}_i^* = \mathsf{Max}(\mathcal{R}_i, \mathcal{R}_j^* \otimes \mathcal{C}_j : j \to i), \quad \mathcal{D}_i^* = \mathsf{Min}(\mathcal{D}_i, \mathcal{D}_j^* \otimes (-\mathcal{C}_j) : i \to j) \tag{2}$$

Let τ^* be the task set obtained by applying Equation (2) to all tasks of a task set τ. From Definition 1 we obtain that $\mathcal{R}_i^* \succeq \mathcal{R}_i$, and $\mathcal{D}_i \succeq \mathcal{D}_i^*$, $\forall i \in \{1, \cdots, n\}$.

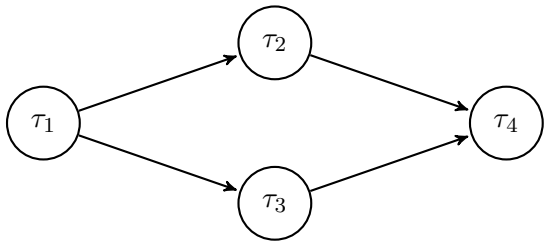


Figure 1: A graph describing the precedence constraints of a task system

Task	r_i	$\mathcal{C}_i$	d_i
τ_1	0	$\begin{pmatrix} 1 & 2 \\ 0.9 & 0.1 \end{pmatrix}$	3
τ_2	1	2	5
τ_3	0	2	4
τ_4	4	3	8

Table 1: Timing parameters

After transforming this task set with the defined Min and Max operations we get an independent probabilistic task set. The proposed transformation is applied as follows:

$$\begin{cases} \mathcal{R}_1^* = \mathcal{R}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \mathcal{R}_2^* = \mathsf{Max}\left(\mathcal{R}_2, \mathcal{R}_1^* \otimes \mathcal{C}_1\right) \\ \mathcal{R}_3^* = \mathsf{Max}\left(\mathcal{R}_3, \mathcal{R}_1^* \otimes \mathcal{C}_1\right) \\ \mathcal{R}_4^* = \mathsf{Max}\left(\mathcal{R}_4, \mathcal{R}_2^* \otimes \mathcal{C}_2, \mathcal{R}_3^* \otimes \mathcal{C}_3\right) \end{cases} \quad \begin{cases} \mathcal{R}_1^* = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \mathcal{R}_2^* = \mathsf{Max}\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 2 \\ 0.9 & 0.1 \end{pmatrix}\right) \\ \mathcal{R}_3^* = \mathsf{Max}\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 2 \\ 0.9 & 0.1 \end{pmatrix}\right) \\ \mathcal{R}_4^* = \mathsf{Max}\left(\begin{pmatrix} 4 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 & 4 \\ 0.9 & 0.1 \end{pmatrix}, \begin{pmatrix} 3 & 4 \\ 0.9 & 0.1 \end{pmatrix}\right) \end{cases}$$

Moreover deadline modification Equation 2 is applied to deadlines. Then, we obtain a transformed task set described in the table below.

Task	$\mathcal{R}_i^*$	$\mathcal{C}_i$	$\mathcal{D}_i^*$
τ_1	0	$\begin{pmatrix} 1 & 2 \\ 0.9 & 0.1 \end{pmatrix}$	2
τ_2	$\begin{pmatrix} 1 & 2 \\ 0.9 & 0.1 \end{pmatrix}$	2	5
τ_3	$\begin{pmatrix} 1 & 2 \\ 0.9 & 0.1 \end{pmatrix}$	2	4
τ_4	4	3	8

References

[1] S. Ben-Amor, D. Maxim and L. Cucu-Grosjean (2016). *Schedulability analysis of dependent probabilistic real-time tasks*. the 24th International Conference on Real-Time and Networked Systems (RTNS).

[2] J. López, J. Díaz, J. Entrialgo and D. García (2008). *Stochastic analysis of real-time systems under preemptive priority-driven scheduling*. Real-Time Systems, pages 180–207.

[3] H. Chetto, M. Silly and T. Bouchentouf (1990). *Dynamic Scheduling of Real-Time Tasks under Precedence Constraints*. Real-Time Systems, 2(3): 181–194.

Beating the Harmonic Lower Bound for Online Bin Packing

Sandy Heydrich* Rob van Stee (Speaker)†

1 Introduction

In the online bin packing problem, items of sizes in $(0, 1]$ arrive online to be packed into bins of size 1. The goal is to minimize the number of used bins. In this paper, we present an online bin packing algorithm with asymptotic performance ratio of 1.5815, which constitutes the first improvement over the algorithm Harmonic++ in fifteen years and reduces the gap to the lower bound by roughly 15%. Within the well-known Super Harmonic framework, no competitive ratio below 1.58333 can be achieved.

The lower bound of Ramanan et al. [1] is based on inputs like the one shown in Fig. 1, which contains a *medium* item (size in $(1/3, 1/2]$) and a *large* item (size in $(1/2, 1]$). Both of these items arrive N times for some large number N, and although they fit pairwise into bins, the algorithm never combines them like this. No matter how fine the item classification of an algorithm, pairs of items such as these, that the algorithm does not pack together into one bin, can always be found.

We make two crucial changes to the Super Harmonic framework. We avoid the lower bound construction by defining the algorithm so that it simply combines medium and large items *whenever* they fit together in a single bin. Essentially, we use Any Fit to combine such items into bins (under certain conditions specified below).

In order to benefit from using Any Fit, it is important to ensure that for each medium type, as much as possible, *it is the smallest items that are colored red.* This is because the red items are the ones that are packed alone into bins, in order to possibly be combined with large items later. The general weakness of all Super Harmonic algorithms is that they do not distinguish between any two items that have the same type; after classifying the items by their size, the size is ignored. This means that the items of any given type could arrive in such an order that the items which are colored red are slightly *larger* than the blue ones. Then, when large items arrive later, they may be too large to fit in bins with red medium items. The online algorithm then has to pack them into new bins, even though they would have fit with blue medium items.

We will avoid this situation (as much as possible) by initially packing *each* medium item alone into a bin and giving it a provisional color. After several items of the same type have arrived, we will color the smallest one red and start packing additional medium items of the same type together with the other items, that are now colored blue. In this way, we can ensure that at least half of the blue items (namely, the ones that have already arrived at the time when we select the smallest to be red) are at least as large as the smallest red item.

*heydrich@mpi-inf.mpg.de. Max Planck Institute for Informatics, 66123 Saarbrücken, Germany.

†rob.vanstee@uni-siegen.de. Department of Mathematics, University of Siegen, 57068 Siegen, Germany. Work performed while this author was at the University of Leicester, UK.

Figure 1: Part of the lower bound construction from Ramanan et al. [1]. The figure shows how one bin is packed in the *optimal* solution. Both of these items arrive many times. The central idea of our algorithm is that we limit the number of times that these "bad" patterns can be used in the optimal solution. This is how we beat the ratio of 1.58333.

Technical difficulties arise because postponing coloring decisions is not always possible or even desirable, depending on the existing packing. We carefully mark medium items depending on how they end up packed. This allows us to bound the number of so-called critical bins (bins that are packed as in Figure 1) in the optimal solution. This will allow us to give a stronger lower bound on the optimal solution.

We simplify the analysis of Super Harmonic and extend it to be able to analyze algorithms in our new framework, which we call Extreme Harmonic. We reduce the problem of analyzing these algorithms to solving a set of knapsack problems. We implemented a computer program which solves the knapsack problems and also does the other necessary work, including the automated setting of many parameters, like item sizes. As a result, our algorithm Son Of Harmonic requires far less manual settings than Harmonic++.

Our program uses an exact representation of fractions, with numerators and denominators of potentially unbounded size, in order to avoid rounding errors. We provide a certificate and a verifier program, and we also output the final set of knapsack problems directly to allow independent verification.

Our approach can also be applied to the original Super Harmonic framework. Surprisingly, we find that the algorithm Harmonic++ is in fact 1.58879-competitive. We suspect that Seiden did not prove this ratio because of the prohibitive running times of his heuristic approach; he mentions that it took 600 hours to prove the upper bound of 1.58889. Our program completes in less than a minute. (Computers became faster since 2000, but not by a factor of 1000!)

Another benefit of using our approach is that this result becomes more easily verifiable as well. Furthermore, we were able to improve and simplify the parameters of Harmonic++ to obtain a competitive ratio of 1.5886 in the old framework.

Finally, we give a lower bound of 1.5762 for any interval classification algorithm, including ones that pack medium and large items like our algorithm. Thus fundamentally different ideas will be needed to get much closer to the lower bound of 1.54037, which we believe is closer to the true competitive ratio of this problem.

References

[1] Prakash V. Ramanan, Donna J. Brown, Chung-Chieh Lee, and D. T. Lee. Online bin packing in linear time. *J. Algorithms*, 10:305–326, 1989.

On Energy Efficient Scheduling of Parallel Jobs with Preemption

Alexander Kononov (Speaker) * Yulia Kovalenko †

1 Introduction

We are given a set of parallel jobs $\mathcal{J} = \{1, \ldots, n\}$, each job $j \in \mathcal{J}$ is specified by its release date r_j, its deadline d_j and its processing volume (work) W_j, and a set of m speed-scalable processors. In our paper we consider two basic variants of scheduling multiprocessor jobs. In the first variant, processing of job j simultaneously requires precisely $size_j$ processors. In the second variant, execution of job j simultaneously needs a prespecified subset fix_j of dedicated processors. Note that the parallel execution of parts of the same job is not allowed. Moreover the execution of each job can be interrupted and resumed without incurring any costs or delays. According to the definitions in the literature on scheduling theory we consider rigid tasks (jobs) and single mode multiprocessor tasks [7].

We consider the standart model in speed-scaling in which if a processor runs at speed s then the energy consumption is s^α units of energy per time unit, where $\alpha > 1$ is a constant (practical studies show that $\alpha \leq 3$). We assume that if processors execute the same job simultaneously then all these processors run at the same speed. For each job $j \in \mathcal{J}$, we say that j is alive during the interval $[r_j, d_j]$. Since processors may change their speed, a job j may be completed faster (or slower) than the time W_j it needs to be executed at speed 1. It is supposed that a continuous or discrete spectrum of processor speeds is available. The goal is to find a feasible schedule respecting the release dates and deadlines of the jobs so that the total energy consumption is minimized.

2 Related results

For the preemptive single-processor case, Yao et al. [10] proposed an optimal algorithm for finding a feasible schedule with minimum energy consumption. The case, where there are m available parallel processors and single-processor jobs, has been solved optimally in polynomial time when both the preemption and the migration of jobs are allowed [1, 3, 5, 9]. The works [1, 3, 9] are based on different reductions of the problem with migration to maximum flow problems. As far as we know, the algorithm presented in [9] has the best running time among the above-mentioned algorithms. A schedule is called migratory if a job may be interrupted and resumed on the same or on another processor.

*alvenko@math.nsc.ru. Sobolev Institute of Mathematics, ak Koptyuga 4, 630090, Novosibirsk, Russia.

†julia.kovalenko.ya@yandex.ru Sobolev Institute of Mathematics, ak Koptyuga 4, 630090, Novosibirsk, Russia.

We note that the migration of jobs is equivalent to the possibility to execute a parallel job in different modes.

Albers et al. [2] considered the problem on parallel processors where the preemption of jobs is allowed but not their migration. They proved that instances with agreeable deadlines and unit-work jobs are solvable in polynomial time. For general instances with unit-work jobs, they proved that the problem becomes strongly NP-hard and they proposed an $(\alpha^\alpha 2^{4\alpha})$-approximation algorithm. For the case where the jobs have arbitrary processing volumes, the problem was proved to be NP-hard even for instances with common release dates and common deadlines. Albers et al. [1] proposed a $2(2-\frac{1}{m})^\alpha$-approximation algorithm for instances with common release dates, or common deadlines, and an $(\alpha^\alpha 2^{4\alpha})$-approximation algorithm for instances with agreeable deadlines. Greiner et al. [8] gave a generic reduction transforming an optimal schedule for the problem on parallel processors with migration to a $B_{\lceil\alpha\rceil}$-approximate solution for the problem on parallel processors with preemptions but without migration, where $B_{\lceil\alpha\rceil}$ is the $\lceil\alpha\rceil$-th Bell number. This result holds only when $\alpha \leq m$. Cohen-Addad et al. [6] showed that the nonmigratory variant of the problem with processor dependent work is APX-hard even for jobs with common life intervals and work volumes in $1, 3, 4$.

Bampis et al. [4] studied the heterogeneous preemptive problem on parallel processors where every processor i has a different speed-to power function, $s^{\alpha(i)}$, and both a life interval and a processing volume of each job are processor dependent. For the migratory variant they proposed a polynomial in problem size and $\frac{1}{\varepsilon}$ algorithm returning a solution within an additive error ε, and for nonmigratory variant an $(1+\varepsilon)^\alpha \tilde{B}_\alpha$-approximation algorithm, where $\tilde{B}_\alpha$ is the generalized Bell number [4].

To the best of our knowledge no one considered the speed scaling scheduling of parallel jobs. For more information on scheduling problems with parallel jobs, we refer the reader to the survey book by M. Drozdowski [7].

3 Our results

In this paper we present exact and approximate algorithms for preemptive speed scaling problems with rigid jobs and single mode two-processor jobs.

Theorem 1 *A $\left(2-\frac{1}{m}\right)^{\alpha-1}$-approximate schedule can be found in $O(n^3)$ time for the speed scaling problem of rigid jobs with migration and continuous spectrum of processing speeds.*

Theorem 2 *1. An optimal schedule can be found for the speed scaling problem of rigid jobs with migration and discrete spectrum of processing speeds in time polynomial in m and the input size.*
2. A schedule of energy consumption $OPT + \varepsilon$ can be found for the speed scaling problem of rigid jobs with migration and continuous spectrum of processing speeds in time polynomial in m, $1/\varepsilon$ and the input size.

We note that the running time mentioned in Theorem 2 is polynomial in the size of the input if m is fixed and pseudo-polynomial if m is a part of the input. We also note that the results of Theorem 2 can be adopted for problems with malleable jobs.

Theorem 3 *1. An optimal schedule can be found for the speed scaling problem with single mode two-processor jobs and discrete spectrum of processing speeds in time polynomial in the input size*
2. A schedule of energy consumption $OPT + \varepsilon$ can be found for the speed scaling problem with single mode two-processor jobs and continuous spectrum of processing speeds in time polynomial in $1/\varepsilon$ and the input size.

We also determine the computational hardness for preemptive speed scaling problems with rigid jobs and single mode multi-processor jobs and claim that most of the NP-hardness proofs for scheduling problems with the minimization maximum lateness criterion may be easily transformed to their speed scaling counterparts.

This research is supported by the Russian Science Foundation grant 15-11-10009.

References

[1] Albers, S., Antoniadis, A., Greiner, G.: On multi-processor speed scaling with migration: extended abstract. In 23rd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2011, ACM, 279–288 (2011)

[2] Albers, S., Müller, F., Schmelzer, S.: Speed scaling on parallel processors. In 19th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2007, ACM, 289–298 (2007)

[3] Angel, E., Bampis, E., Kacem, F., Letsios, D.: Speed scaling on parallel processors with migration. In 18th International European Conference on Parallel and Distributed Computing, Euro-Par 2012. LNCS, Vol. 7484, 128–140, Springer (2012)

[4] Bampis, E., Kononov, A., Letsios, D., Lucarelli, G., Sviridenko, M.: Energy efficient scheduling and routing via randomized rounding. In FSTTCS, 449–460 (2013)

[5] Bingham, B.D., Greenstreet, M.R.: Energy optimal scheduling on multiprocessors with migration. In International Symposium on Parallel and Distributed Processing with Applications, ISPA 2008, IEEE, 153–161 (2008)

[6] Cohen-Addad, V., Li, Z., Mathieu, C., Milis, I.: Energy-efficient algorithms for non-preemptive speed-scaling. In WAOA 2014. LNCS, Vol. 8952, 107–118 (2015)

[7] Drozdowski, M.: Scheduling for Parallel Processing. Springer-Verlag, London (2009)

[8] Greiner, G., Nonner, T., Souza, A.: The bell is ringing in speed-scaled multiprocessor scheduling. In 21st ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2009, ACM, 11–18 (2009)

[9] Shioura, A., Shakhlevich, N., Strusevich, V.: Energy saving computational models with speed scaling via submodular optimization. In Proceedings of Third International Conference on Green Computing, Technology and Innovation (ICGCTI2015), Serdang, Malaysia, 7–18 (2015)

[10] Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In 36th Annual Symposium on Foundation of Computer Science, 374–382 (1995)

Robust Assignments: Hardness, Approximability and Algorithms

David Adjiashvili * Viktor Bindewald (Speaker) † Dennis Michaels ‡

1 Introduction

Many real-life planning problems require making a priori decisions before all parameters of the problem have been revealed. An important special case of such problem arises in scheduling or staff rostering problems, where a set of tasks needs to be assigned to the available set of machines or personnel (resources), in a way that all tasks have assigned resources, and no two tasks share the same resource. In its nominal form, the resulting computational problem becomes the *assignment problem* on bipartite graphs.

This paper deals with a robust variant of the assignment problem modeling situations where the structure of the corresponding graph is subject to uncertainty, i.e. some edges are *vulnerable* and may become unavailable after a solution has been chosen. An optimal solution to the robust assignment problem (RAP) guarantees that all tasks can be performed (independent of the effective structure shift) and minimizes the cost.

We conclude the introduction with a more precise description of an applicaton in the field of *Staff Scheduling and Rostering.* Rostering is an important subtask of Human Resource Management and has four different aspects: strategic, tactical, operational and retrospective. RAP addresses the tactical dimension of rostering. The objective thereby is to deliver a prototype schedule for a certain planning horizon (e.g. a month), which can be used as a starting point for operative assignment decisions on a daily (or weekly) basis. Naturally a roster is subject to uncertainty because employees may get sick or not be able to work in a certain shift due to legal work load regulations or non-availability of crucial equipment. Uncertainty of this kind can be incorporated into our model. A solution to RAP enables the decision maker to choose an assignment from the solution for the upcoming day (or week), meeting the latest needs of both, the company and personnel.

2 Problem Definition

Given a undirected biparite graph $G = (U \dot\cup W, E)$ we call the nodes in U jobs and the nodes in W machines. The edge set E is subject to uncertainty, described via an

*addavid@ethz.ch. Institute for Operations Research, Department of Mathematics, ETH Zurich, Rämistrasse 101, 8092 Zurich, Switzerland

†viktor.bindewald@math.tu-dortmund.de. Department of Mathematics, TU Dortmund University, Vogelpothsweg 87, 44227 Dortmund, Germany

‡dennis.michaels@math.tu-dortmund.de. Department of Mathematics, TU Dortmund University, Vogelpothsweg 87, 44227 Dortmund, Germany

uncertainty set $\mathcal{U} = \{F_1, \ldots, F_k\} \subseteq 2^E$. Each element F of $\mathcal{U}$ describes one possible failure scenario. Upon emerging of scenario F, the corresponding edges are removed from the graph G. Eventually a non-negative cost function $c : 2^E \to \mathbf{R}_{\geq 0}$ is provided. An assignment in G is a set of non-adjacent edges from E covering every job node in U. The Robust Assignment Problem is defined as follows.

$$\begin{array}{ll} \min & c(X) \\ \text{s.t.} & \forall F \in \mathcal{U} : X \setminus F \text{ contains an assignment,} \\ & X \subseteq E \end{array} \tag{RAP}$$

In other words $X \subseteq E$ is a solution to RAP if for each failure scenario F the subgraph $G[X] - F$ contains an assignment. This makes RAP a bulk-robust problem, a robustness concept introduced in [1]. Note that unlike in many other robust optimization settings, solutions to RAP are in general not feasible to the nominal assignment problem in the setting considered here.

3 Our Contribution

To the best of our knowledge this optimization problem was not investigated before.

The key component for hardness and inapproximability results is the following theorem.

Theorem 1 ([2]) *Minimum Set Cover can be reduced to RAP, preserving the cost.*

This result implies that RAP is not only NP-hard, but also can not be approximated with a constant guarantee under standard complexity assumptions. Our reduction uses singleton scenario sets and for this setting we provide a randomized $O(\log n)$ polynomial approximation algorithm in case of general cost functions and a $O(1)$-approximation for uniform costs. Both algorithms match the approximability bounds known for Set Cover asymptotically. We complete the complexity landscape by investigating the simplest variant of RAP.

Theorem 2 ([2]) *RAP is NP-hard when restricted to instances with uniform costs and two vulnerable edges, i.e. with* $\mathcal{U} = \{f_1, f_2\}$.

The second part of the talk will discuss a version of RAP with uncertain machine node set W [3].

References

[1] D. Adjiashvili, S. Stiller, and R. Zenklusen (2015). *Bulk-Robust combinatorial optimization.* Mathematical Programming 149(1), pp. 361-390.

[2] D. Adjiashvili, V. Bindewald, and D. Michaels (2016). *Robust Assignments via Ear Decompositions and Randomized Rounding.* In Chatzigiannakis, I.; Mitzenmacher, M.; Rabani, Y.; Sangiorgi, D. (eds.): Proceedings of 43rd International Colloquium on Automata, Languages, and Programming (ICALP), pp. 71:1 -71:14. (Full version: https://arxiv.org/abs/1607.02437)

[3] D. Adjiashvili, V. Bindewald, and D. Michaels (2017). *Robust Assignments with Vulnerable Nodes.* arXiv preprint: https://arxiv.org/abs/1703.06074

On the Optima Localization in Two-Machine Routing Open Shops

Ilya Chernykh (Speaker) * Ekaterina Lgotina †

1 Introduction

The routing open shop model was introduced in [1, 2]. In this model the sets of n jobs $\mathcal{J}$ and m machines $\mathcal{M}$ are given, and machines have to perform operations of each job J_j (with given processing times p_{ji}) in an arbitrary order similar to the classic open shop scheduling problem [6]. The jobs are distributed among the nodes of some transportation network represented by edge-weighted graph G. The weight τ_{pq} of edge $[v_p, v_q]$ represents the travel distance between the nodes v_p and v_q. The set of jobs located at some node v_k is denoted as $\mathcal{J}_k$. All the machines are initially located at a predefined special node referred to as *the depot*. Machines have to travel with unit speed between the nodes of the transportation network to process their operations and to return to the depot. Machines are allowed to use the shortest paths between the nodes therefore we may assume that travel distances satisfy the triangle inequality.

For any schedule S value $R_{\max}(S)$ denotes the *makespan* of S which is the time moment of returning of the last machine to the depot after processing all the operations. The goal is to minimize the makespan. Following the traditional three-field notation we denote the routing open shop problem as $RO||R_{\max}$.

We assume that each node (with the possible exception of the depot) contains at least one job. This makes it necessary for each machine to visit each node at least once. Therefore the routing open shop with single machine is equivalent to the classic metric TSP which is well-known to be NP-hard in strong sense. On the other hand, a single-node routing open shop is just a plain open shop problem and is NP-hard for three and more machines while being polynomially solvable in the two-machine case [6]. Surprisingly, the combination of those classic problems remains NP-hard even in the simplest non-trivial case with two machines on a link ($RO2|G = K_2|R_{\max}$), as proved in [1].

Lets give a brief review of the routing open shop problem focusing on a case of two machines ($RO2||R_{\max}$). A first $\frac{7}{4}$-approximation algorithm is described in [1]. It was further improved in [4] where a $\frac{13}{8}$-approximation algorithm is described. This improvement is relatively significant due to the following remark. Note that the $RO2||R_{\max}$ problem includes metric TSP as a special case. Since the best known up to date approximation algorithm for the metric TSP is the $\frac{3}{2}$-approximation algorithm due to Christofides and

*idchern@math.nsc.ru. Sobolev Institute of Mathematics, 4 Koptyug ave., Novosibirsk, 630090, Russian Federation.

†kate.lgotina@gmail.com. Novosibirsk State University, 2 Pirogova Str, Novosibirsk, 630090, Russian Federation.

Serdyukov, we cannot hope to achieve better than $\frac{3}{2}$-approximation for $RO2||R_{\max}$ until a better approximation for the metric TSP will be found. On the other hand the *easy-TSP* version of the $RO2||R_{\max}$ (the case when an optimal solution for the underlying TSP is known or the time complexity of its search is not taken into account) the problem admits a $\frac{4}{3}$-approximation algorithm described in [4].

In [3] the following generalization of the routing open shop problem was introduced. In that model travel times τ_{pq}^i are specific for each machine M_i. The following hierarchy of the travel time models was considered in [3]:

- $RO||R_{\max}$: $\tau_{pq}^i = \tau_{pq}$ (*identical* travel times);
- $RO|Qtt|R_{\max}$: $\tau_{pq}^i = \dfrac{\tau_{pq}}{s_i}$ (*uniform* travel times, s_i represents the travel speed of machine M_i);
- $RO|Rtt|R_{\max}$: τ_{pq}^i are individual for each machine (*unrelated* travel times).

The following *standard lower bound* $\bar{R}$ for $RO||R_{\max}$ was introduced in [2]:

$$\bar{R} \doteq \max\left\{\ell_{\max} + T^*, \max_k(d_{\max}^k + 2\tau_{0k})\right\},$$

where $\ell_{\max} \doteq \max \ell_i = \max\limits_i \sum\limits_{j=1}^{n} p_{ji}$ is the maximum machine load, T^* stands for the optimum of TSP on graph G, $d_{\max}^k \doteq \max\limits_k \max\limits_{j\in\mathcal{J}_k} \sum\limits_{i=1}^{m} p_{ji}$ is the maximum job length from $\mathcal{J}_k$. Although (as shown in [3]) the standard lower bound cannot be directly generalized for *Qtt* and *Rtt* cases, this still can be done if we only have two machines:

$$\bar{R}_2 \doteq \max\left\{\max_i(\ell_i + T_i^*), \max_k(d_{\max}^k + \tau_{0k}^1 + \tau_{0k}^2)\right\},$$

where T_i^* stands for the optimal tour length for machine M_i.

2 New results and open questions

In this talk we consider the following *optima localization problem* (similar to [7]). Let $\mathcal{I}$ be some set of *non-trivial* instances of two-machine routing open shop problem (with $\bar{R}_2(I) > 0\ \forall I \in \mathcal{I}$). Define the following *critical bound*

$$\rho^*(\mathcal{I}) \doteq \sup_{I\in\mathcal{I}} \frac{R_{\max}^*(I)}{\bar{R}_2(I)}.$$

By that definition we have

$$\forall I \in \mathcal{I}\ R_{\max}^*(I) \in [\bar{R}_2(I), \rho^*(\mathcal{I})\bar{R}_2(I)]$$

and the bounds of that *optima localization* interval are tight. The problem of finding of the critical bound for wide sets of instances (i.e. optima localization problem) is of both theoretical and practical importance.

Let $\mathcal{I}_{\alpha 2}^G$, $\alpha \in \{\circ, Q, R\}$ stands for the set of all non-trivial instances of the two-machine routing open shop on graph G with α corresponding to identical, uniform and unrelated travel times. Then we have the following results:

Theorem 1 ([2, 5]) *For any $G \in \{K_2, K_3, star\}$ the critical bound $\rho^*\left(\mathcal{I}_2^G\right) = \dfrac{6}{5}$. For any $I \in \mathcal{I}_2^G$ a feasible schedule of makespan not exceeding $\frac{6}{5}\bar{R}_2(I)$ can be found in time $O(n)$.*

Theorem 2 *For any $G \in \{K_2, K_3, star\}$ the critical bounds $\rho^*\left(\mathcal{I}_{Q2}^G\right) = \rho^*\left(\mathcal{I}_{R2}^G\right) = \dfrac{5}{4}$. For any $I \in \mathcal{I}_{Q2}^G \cup \mathcal{I}_{R2}^G$ a feasible schedule of makespan not exceeding $\frac{5}{4}\bar{R}_2(I)$ can be found in time $O(n)$.*

Note that the general critical bound for $G = K_N$ is still unknown both for identical and individual travel times. It follows from [4] that

$$\rho^*\left(\mathcal{I}_2^{K_N}\right) \leqslant \frac{4}{3},$$

although there is yet no evidence known that the critical bound for $RO2||R_{\max}$ may exceed $\dfrac{6}{5}$. Therefore we propose the following

Conjecture 3 $\rho^*\left(\mathcal{I}_2^{K_N}\right) = \dfrac{6}{5}$.

References

[1] I. Averbakh and O. Berman and I. Chernykh (2006). *The routing open-shop problem on a network: complexity and approximation.* European Journal of Operational Research, Vol. 173(2), pp. 521–539.

[2] I. Averbakh and O. Berman and I. Chernykh (2005). *A 6/5-approximation algorithm for the two-machine routing open shop problem on a 2-node network.* European Journal of Operational Research, Vol. 166(1), pp. 3–24.

[3] I. Chernykh (2016). *Routing open shop with unrelated travel times.* In: 9th International Conference on Discrete Optimization and Operations Research, DOOR 2016; Lecture Notes in Computer Science, Vol. 9869, pp. 272–283.

[4] I. Chernykh and A. Kononov and S. Sevastyanov (2013). *Efficient approximation algorithms for the routing open shop problem.* Comput. Oper. Res., Vol 40(3), pp. 841–847.

[5] I. Chernykh and E. Lgotina (2016). *The 2-machine routing open shop on a triangular transportation network.* In: 9th International Conference on Discrete Optimization and Operations Research, DOOR 2016; Lecture Notes in Computer Science, Vol. 9869, pp. 284–297.

[6] T. Gonzalez and S. Sahni (1976). *Open shop scheduling to minimize finish time.* J. Assoc. Comput. Mach., Vol. 23, pp. 665–679.

[7] S. V. Sevastyanov and I. D. Tchernykh (1998). *Computer-aided way to prove theorems in scheduling.* In: Algorithms — ESA'98, Lecture Notes in Computer Science, Vol. 1461, pp. 502–513.

Exact Algorithms for the Equitable Traveling Salesman Problem

Joris Kinable [*] Bart Smeulders [†] Eline Delcour [‡]
Frits C.R. Spieksma (Speaker) [§]

1 Introduction

We consider the following variation of the Traveling Salesman Problem (TSP). Given is an edge-weighted graph $G = (V, E)$, with $|V|$ even, and with edge-costs d_e for each $e \in E$. The cost of a matching in G is defined as the sum of edge-costs of the edges in the matching. The problem is to find two perfect matchings in G such that (i) the two matchings form a Hamiltonian cycle in G, and (ii) the absolute difference of the costs of these two matchings is minimum. Notice that a feasible solution need not exist. We call this problem the Equitable Traveling Salesman Problem, or ETSP for short.

This name is motivated by the following, more frivolous, description of our problem: two friends, in possession of a single bike, have agreed to jointly visit all given cities, i.e., to construct a tour. In addition, they have agreed to use the bike as follows: one friend rides (pedals) the bike, while the other sits on the bike's back. Directly after having visited a city, the two friends interchange roles. The objective in this problem is to find a tour such that the difference between the distances pedalled by each of the two friends, is minimum.

We present the following results. First, the problem is shown to be NP-Hard, even when the graph G is complete. Second, we give two integer programming models formulating the ETSP (see Section 2), and we compare the strength of these formulations. Third, we perform computational experiments, solving one model through branch-and-cut, whereas the other model is solved through a branch-and-price framework. A simple local search heuristic is also implemented. We conduct these computational experiments on different types of instances, often derived from the TSPLib. It turns out that the behavior of the different approaches varies with the type of instances. We refer to Kinable et al. [2] for precise statements, their proofs, and a discussion of the computational experiments.

[*] jkinable@cs.cmu.edu. Robotics Institute & Tepper School of Business, Carnegie Mellon University, 5000 Forbes Ave Pittsburgh, USA.
[†] bart.smeulders@ulg.ac.be. HEC Management School, University of Liège, QuantOM, Belgium.
[‡] Barry Callebaut
[§] frits.spieksma@kuleuven.be. Faculty of Business and Economics, KU Leuven, Leuven, Belgium.

2 Formulations for the ETSP

We give two integer programming formulations of the ETSP. In fact, we treat a slightly more general problem where the edge-sets, and the edge-costs, need not be the same for the two matchings. We use E_B and E_R to denote the two edge-sets; E_B refers to the 'blue' edges that can be used for one matching, while E_R refers to the 'red' edges to be used for the other matching. More precisely, we are given a graph $G = (V, E_B \cup E_R)$, and, let $\mathcal{M}_B$ ($\mathcal{M}_R$) refer to the set of perfect matchings in (V, E_B) ((V, E_R)). Each edge e in E_B (E_R) has a cost d_e^b (d_e^r). The cost of a matching $M \in \mathcal{M}_B$ is defined as $c^b(M) = \sum_{e \in M} d_e^b$. Analogously, the cost of a matching $M \in \mathcal{M}_R$ is $c^r(M) = \sum_{e \in M} d_e^r$. Notice that this problem definition does not require that $E_B \cap E_R = \emptyset$. Furthermore, a single edge $e \in E_B \cap E_R$ may have different weights in the red or the blue matching (i.e., $d_e^b = d_e^r$ does not necessarily hold). Finally, we define $\delta(S)$, $S \subseteq V$ as the set of edges having exactly one endpoint in S.

2.1 Formulation FBB

The first formulation uses two binary variables for each edge $e \in E_B$:

$$x_e^b \ (x_e^r) := \begin{cases} 1 & \text{if edge } e \text{ is selected in the blue (red) matching,} \\ 0 & \text{otherwise,} \end{cases}$$

$$\begin{array}{lrll} FBB: & \min & |\sum_{e \in E_B} d_e^b x_e^b - \sum_{e \in E_R} d_e^r x_e^r| & \\ & \text{s.t.} & \sum_{e \in \delta(v) \cap E_B} x_e^b = 1 & \forall v \in V \\ & & \sum_{e \in \delta(v) \cap E_R} x_e^r = 1 & \forall v \in V \\ & & x_e^b + x_e^r \leq 1 & \forall e \in E_B \cap E_R \\ & & \sum_{e \in \delta(S) \cap E_B} x_e^b + \sum_{e \in \delta(S) \cap E_R} x_e^r \geq 2 & \forall S \subset V, |S| \geq 3 \\ & & x_e^b \in \{0,1\} & \forall e \in E_B \\ & & x_e^r \in \{0,1\} & \forall e \in E_R \end{array}$$

2.2 Formulation FBP

Our second formulation has a variable for each perfect matching in the graph. More precisely, we define a binary variable for each perfect matching $M \in \mathcal{M}_B$ in (V, E_B) ($M \in \mathcal{M}_R$ in (V, E_R)):

$$z_M^b \ (z_M^r) := \begin{cases} 1 & \text{if perfect matching } M \text{ is selected as the blue (red) matching,} \\ 0 & \text{otherwise,} \end{cases}$$

$$\begin{array}{lrll} FBP: & \min & |\sum_{M \in \mathcal{M}_B} c^b(M) z_M^b - \sum_{M \in \mathcal{M}_R} c^r(M) z_M^r| & \\ & \text{s.t.} & \sum_{M \in \mathcal{M}_B} z_M^b = 1 & \\ & & \sum_{M \in \mathcal{M}_R} z_M^r = 1 & \\ & & \sum_{M \in \mathcal{M}_B:\ e \in M} z_M^b + \sum_{M \in \mathcal{M}_R:\ e \in M} z_M^r \leq 1 & \forall e \in E_B \cap E_R \\ & & \sum_{e \in \delta(S)} (\sum_{M \in \mathcal{M}_B : e \in M} z_M^b + \sum_{M \in \mathcal{M}_R : e \in M} z_M^r) \geq 2 & \forall S \subset V, |S| \geq 3 \\ & & z_M^b \in \{0,1\} & \forall M \in \mathcal{M}_B \\ & & z_M^r \in \{0,1\} & \forall M \in \mathcal{M}_R \end{array}$$

2.3 Comparing formulations FBB and FBP

Strictly speaking, the formulations given above are not linear due to the absolute value present in the objective function; a standard trick exists to make these formulations linear. We are interested in comparing the linear relaxations of formulations FBB and FBP. Given an instance I of ETSP, let $v_{BB}(I)$ ($v_{BP}(I)$) denote the value of the linear relaxation of FBB (FBP) when applied to instance I. Following standard terminology (see Vielma [1] and references contained therein), we say that the linear relaxation of FBP is *stronger* than the relaxation of FBB when the two following conditions are fulfilled:

C1: for each instance I of ETSP, $v_{BP}(I) \geq v_{BB}(I)$,

C2: there exists an instance I of ETSP for which $v_{BP}(I) > v_{BB}(I)$.

Theorem 1 *The linear relaxation of FBP is stronger than the linear relaxation of FBB.*

References

[1] J. Vielma (2015). *Mixed integer linear programming formulation techniques*. SIAM Review 57, 3-57.

[2] J. Kinable and B. Smeulders and E. Delcour and F.C.R. Spieksma (2016). *Exact algorithms for the Equitable Traveling Salesman Problem*, Research Report KBI_1610, Faculty of Business and Economics, KU Leuven.

Polyhedral Results and Valid Inequalities for the Continuous Energy-Constrained Scheduling Problem

Margaux Nattaf (Speaker)* Tamás Kis† Christian Artigues*
Pierre Lopez*

1 Introduction

This works deals with event-based mixed integer linear programming formulations for resource-constrained scheduling problems. In this context, two strongly NP-hard problems are considered: the Resource-Constrained Project Scheduling Problem (RCPSP) and the Continuous Energy-Constrained Scheduling Problem (CECSP). Several mixed integer programs have been developed so far in order to solve them [1, 2, 4].

For the RCPSP, time-indexed formulations have a much better LP relaxation than the event-based ones. However, time-indexed formulations show their weakness for instances with large planning horizons and heterogeneous task durations, while event-based models proved to be more efficient to provide exact solutions for these types of instances [1]. For the CECSP, event-based formulations are the only ones that can provide optimal solutions. Indeed, an instance of the CECSP may have only solutions containing non-integer values and such a solution can not be reached by a time-indexed formulation [2]. In this work, we are interested in the strengthening of those event-based models. The results presented here are detailed in [3].

2 Problems description

In the CECSP, the goal is to schedule a set of tasks $\mathcal{A} = \{1, \ldots, n\}$ using a continuous resource of limited capacity B. At each time t during its processing, a task uses an amount of the resource $b_i(t)$, which has to lie between a minimal and a maximal requirement, b_i^{min} and b_i^{max}, respectively. In addition, a task has to be executed during its time window $[r_i, d_i]$. The particularity of the CECSP is that a task no longer has a fixed duration but instead an energy requirement W_i needs to be fulfilled before the task deadline. This energy is computed from the task resource usage, using an efficiency function f_i. We assume these functions to be continuous, non-decreasing and linear. The objective is to minimize the total resource consumption.

The RCPSP deals with a set of resources $\mathcal{R}$ and each task consumes a part of one or several of these resources. Each resource $k \in \mathcal{R}$ has a limited capacity B_k. Unlike the

*{nattaf,artigues,lopez}@laas.fr. LAAS-CNRS, Univ. de Toulouse, CNRS, INSA, Toulouse, France.

†kis.tamas@sztaki.mta.hu. Institute for Computer Science and Control, Kende str. 13-17, H-1111 Budapest, Hungary, supported by OTKA grant 112881.

CECSP, tasks have a fixed duration p_i and a fixed resource consumption b_{ik} (possibly zero). Furthermore, precedence relationships exist between tasks and a valid schedule has to satisfy those relationships while minimizing the project total duration.

3 Event-based models

Unlike time-indexed models, event-based models focus on the prevailing dates of the schedule, also called events. Then, an event can correspond for example to a task start or end time. These events are represented by a set of continuous variables t_e and $\mathcal{E}$ represents the index set of these events.

There exist two types of event-based models. The first one is called the start/end (SE) model and the second one is called the on/off (OO) model [1, 2]. In the SE model, a set of binary variables x_{ie} (resp. y_{ie}) is used to represent the fact that task i starts (resp. ends) at t_e. In the OO formulation, only one set of binary variables z_{ie} is used and is equal to 1 iff task i is in process between t_e and t_{e+1}.

Note that SE formulations have better relaxations than OO ones [4]. In the following, we present several sets of inequalities that can be added to the OO models in order to strengthen them. Note that these inequalities can also directly be rewritten for SE models as z_{ie} can be written as a linear function of x_{ie} and y_{ie}.

4 Valid inequalities for event-based models

Non-preemptive inequalities The first set of inequalities is called non-preemptive inequalities. This set can be used to provide a minimal description of the polytope of all feasible assignments to the on/off binary variables for a single activity.

$$\sum_{e_k \in S} (-1)^k z_{i,e_k} \leq 1 \tag{1}$$

where $S = \{e_0, e_1, \ldots, e_{2\ell}\}$ is a subset of $\mathcal{E} \setminus \{2n\}$ of odd cardinality such that $e_k < e_{k+1}$ for $k = 0, \ldots, 2\ell - 1$.

Note that in [4], inequalities (2.13) are a special case of our non-preemptive cuts, defined over three events only. The inequalities we propose can be partially derived from the polyhedral results in [5]. We present an alternative proof that these inequalities are facet defining tailored to our particular case and we also propose an original polynomial time separation algorithm.

Maximal distance between events The goal of this set of inequalities is to give an upper bound on the distance between two consecutive events. To define these inequalities, we consider the intervals of possible start/end times of tasks. Since an event must occur in each of these intervals, we can deduce bounds on the distance between consecutive intervals.

The same kind of reasoning can also be applied to find upper bounds on the date of a single event t_e.

Knapsack inequalities For the CECSP, since the minimum intensity of the activities can be positive, we can consider the following knapsack-type constraint for each $e \in \mathcal{E} \setminus \{2n\}$ from which one can easily derive valid inequalities:

$$\sum_{i \in \mathcal{A}} b_i^{min} z_{ie} \leq B \tag{2}$$

For the RCPSP, these inequalities correspond to the constraints limiting resources usage.

5 Experiments

The experiments are conducted on an Intel Core i7-4770 processor with 4 cores and 8 gigabytes of RAM under the 64-bit Ubuntu 12.04 operating system. We use IBM CPLEX 12.6 with one thread and a time limit of 1000 seconds for solving the MILP models. The benchmark instances we use to perform these experiments are those of [2] for the CECSP and of [1] for the RCPSP.

For the CECSP, the sets of inequalities allow us to solve more instances. They also help us to obtain solutions (first and final) of better quality and/or more quickly, especially when the number of tasks grows. However, these inequalities have little impact on the root gap.

For the RCPSP, the addition of the inequalities have less impact than for the CECSP. However, they still improve the time spent to solve the instances and to find first solutions of better quality but this improvement is not as significant as for the CECSP.

6 Conclusion and Further research

There are numerous directions for further research. In particular, a challenge should be to find symmetry breaking inequalities. Indeed, one weakness of event-based models is the considerable number of symmetries they contain. Besides, a minimal description of the polyhedra of all asymmetric feasible assignments of the on/off variables not for a single activity but for all of them can also be an interesting track.

References

[1] O. Koné, C. Artigues, P. Lopez and M. Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1): 3–13, 2011.

[2] M. Nattaf, C. Artigues, P. Lopez and D. Rivreau. Energetic reasoning and mixed-integer linear programming for scheduling with a continuous resource and linear efficiency functions. *OR Spectrum*, 38(2): 459–492, 2016.

[3] M. Nattaf, T. Kis, C. Artigues and P. Lopez (2016). Polyhedral results and valid inequalities for the Continuous Energy-Constrained Scheduling Problem. https://hal.archives-ouvertes.fr/hal-01391403.

[4] A. Tesch. *Compact MIP Models for the Resource-Constrained Project Scheduling Problem.* Master thesis, Technische Universität Berlin, 2015.

[5] H. Grflin, T.M. Liebling. Connecting and alternating vectors: polyhedra and algorithms. *Mathematical Programming*, 20, 233-244, 1981.

Efficient Frameworks for Utilization-Based Analysis for Fixed-Priority Scheduling in Real-Time Systems*

Jian-Jia Chen (Speaker) † Wen-Hung Huang ‡ Cong Liu §

1 Introduction

To analyze the worst-case response time or to ensure the timeliness of the system, for each of individual task models, researchers tend to develop dedicated techniques that result in schedulability tests with different computation complexity and accuracy of the analysis. Although many successful results have been developed, after many real-time systems researchers devoted themselves for many years, there does not exist a general framework that can provide efficient and effective analysis for different task models.

A very widely adopted case is the schedulability test of a (constrained-deadline) sporadic real-time task τ_k under fixed-priority scheduling in uniprocessor systems, in which the time-demand analysis (TDA) developed in [5] can be adopted. That is, if

$$\exists t \text{ with } 0 < t \leq D_k \;\; and \;\; C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t, \tag{1}$$

then task τ_k is schedulable under the fixed-priority scheduling algorithm, where $hp(\tau_k)$ is the set of tasks with higher priority than τ_k, D_k, C_k, and T_i represent τ_k's relative deadline, worst-case execution time, and period, respectively. TDA requires pseudo-polynomial-time complexity to check the time points that lie in $(0, D_k]$.

However, it is not always necessary to test all possible time points to derive a safe worst-case response time or to provide sufficient schedulability tests. The general and key concept to obtain sufficient schedulability tests in **k2U** in [2, 3] and **k2Q** in [1, 4] is to test only a subset of such points for verifying the schedulability. Traditional fixed-priority schedulability tests often have pseudo-polynomial-time (or even higher) complexity. The idea implemented in the **k2U** and **k2Q** frameworks is to provide a general k-point schedulability test, which only needs to test k points under *any* fixed-priority scheduling when checking schedulability of the task with the k^{th} highest priority in the system. Moreover, this concept is further extended in **k2Q** to provide a safe upper bound of the worst-case response time. The response time analysis and the schedulability analysis provided by the frameworks can be viewed as "*blackbox*" interfaces that can result in sufficient utilization-based analysis, in which the utilization of a task is its execution time divided by its period.

*This paper is supported by the DFG, as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/).

†jian-jia.chen@cs.uni-dortmund.de Department of Informatics, TU Dortmund, Germany

‡wen-hung.huang@cs.uni-dortmund.de Department of Informatics, TU Dortmund, Germany

§cong@utdallas.edu Department of Computer Science, University of Texas at Dallas, USA

In **k2U**, all the testings and formulations are based on the task utilizations. In **k2Q**, the testings are based not only on the task utilizations, but also on the task execution times. The different formulations of testings result in different types of solutions. The natural form of **k2U** is a hyperbolic form for testing the schedulability of a task, whereas the natural form of **k2Q** is a quadratic form.

2 k2U and k2Q Frameworks

We first present the definitions and properties of the **k2U** and **k2Q** frameworks for testing the schedulability of task τ_k in a given set of real-time tasks. We implicitly assume that there are $k-1$ higher-priority tasks when we consider task τ_k.

Definition 1. *A k-point effective schedulability test is a sufficient schedulability test of a fixed-priority scheduling policy, that verifies the existence of* $t_j \in \{t_1, t_2, \dots t_k\}$ *with* $0 < t_1 \leq t_2 \leq \cdots \leq t_k$ *such that*

$$C_k + \sum_{i=1}^{k-1} \alpha_i t_i U_i + \sum_{i=1}^{j-1} \beta_i t_i U_i \leq t_j, \tag{2}$$

where $C_k > 0$, $\alpha_i > 0$, $U_i > 0$, *and* $\beta_i > 0$ *are dependent upon the setting of the task models and task* τ_i. □

Definition 2 (Last Release Time Ordering)**.** *Let* π *be the last release time ordering assignment as a bijective function* $\pi : hp(\tau_k) \rightarrow \{1, 2, \dots, k-1\}$ *to define the last release time ordering of task* $\tau_j \in hp(\tau_k)$ *in the window of interest. Last release time orderings are numbered from* 1 *to* $k-1$, *i.e.,* $|hp(\tau_k)|$, *where 1 is the earliest and* $k-1$ *the latest.* □

Definition 3. *A k-point last-release schedulability test under a given ordering* π *of the* $k-1$ *higher priority tasks is a sufficient schedulability test of a fixed-priority scheduling policy, that verifies the existence of* $0 \leq t_1 \leq t_2 \leq \cdots \leq t_{k-1} \leq t_k$ *such that*

$$C_k + \sum_{i=1}^{k-1} \alpha_i t_i U_i + \sum_{i=1}^{j-1} \beta_i C_i \leq t_j, \tag{3}$$

where $C_k > 0$, *for* $i = 1, 2, \dots, k-1$, $\alpha_i > 0$, $U_i > 0$, $C_i \geq 0$, *and* $\beta_i > 0$ *are dependent upon the setting of the task models and task* τ_i. □

2.1 Key Properties of k2U

By using the property defined in Definition 1, we can have the following lemmas in the **k2U** framework [2, 3]. All the proofs of the following lemmas are in [2, 3].

Lemma 4. *For a given k-point effective schedulability test of a scheduling algorithm, defined in Definition 1, in which* $0 < t_k$ *and* $0 < \alpha_i \leq \alpha$, *and* $0 < \beta_i \leq \beta$ *for any* $i = 1, 2, \dots, k-1$, *task* τ_k *is schedulable by the scheduling algorithm if the following condition holds*

$$\frac{C_k}{t_k} \leq \frac{\frac{\alpha}{\beta} + 1}{\prod_{j=1}^{k-1}(\beta U_j + 1)} - \frac{\alpha}{\beta}. \tag{4}$$

Lemma 5. *For a given k-point effective schedulability test of a scheduling algorithm, defined in Definition 1, in which* $0 < t_k$ *and* $0 < \alpha_i \leq \alpha$ *and* $0 < \beta_i \leq \beta$ *for any* $i = 1, 2, \ldots, k-1$*, task* τ_k *is schedulable by the scheduling algorithm if*

$$\frac{C_k}{t_k} + \sum_{i=1}^{k-1} U_i \leq \frac{(k-1)((\alpha+\beta)^{\frac{1}{k}} - 1) + ((\alpha+\beta)^{\frac{1}{k}} - \alpha)}{\beta}. \tag{5}$$

2.2 Key Properties of k2Q

By using the property defined in Definition 3, we can have the following lemmas in the **k2Q** framework [1, 4]. All the proofs of the following lemmas are in [1, 4].

Lemma 6. *For a given k-point last-release schedulability test, defined in Definition 3, of a scheduling algorithm, in which* $0 < \alpha_i$*, and* $0 < \beta_i$ *for any* $i = 1, 2, \ldots, k-1$*,* $0 < t_k$*,* $\sum_{i=1}^{k-1} \alpha_i U_i \leq 1$*, and* $\sum_{i=1}^{k-1} \beta_i C_i \leq t_k$*, task* τ_k *is schedulable by the fixed-priority scheduling algorithm if the following condition holds*

$$\frac{C_k}{t_k} \leq 1 - \sum_{i=1}^{k-1} \alpha_i U_i - \frac{\sum_{i=1}^{k-1} (\beta_i C_i - \alpha_i U_i (\sum_{\ell=i}^{k-1} \beta_\ell C_\ell))}{t_k}. \tag{6}$$

It may seem at first glance that we need to test all the possible orderings. Fortunately, with the following lemma, we can safely consider only one specific ordering of the $k-1$ higher priority tasks.

Lemma 7. *The worst-case ordering* π *of the* $k-1$ *higher-priority tasks under the schedulability condition in Eq.* (6) *in Lemma 6 is to order the tasks in a non-increasing order of* $\frac{\beta_i C_i}{\alpha_i U_i}$*, in which* $0 < \alpha_i$ *and* $0 < \beta_i$ *for any* $i = 1, 2, \ldots, k-1$*, and* $0 < t_k$*.*

3 Applications

The **k2U** and **k2Q** frameworks can be used by a wide range of applications from uniprocessor systems to multiprocessor systems and from the simplest sporadic real-time task model to very expressive real-time task models. The frameworks can be applied as long as the users can properly specify the corresponding task properties C_i (in case of **k2Q**) and U_i and the constant coefficients α_i and β_i of every higher priority task τ_i. The choice of the parameters α_i and β_i affects the quality of the resulting schedulability bounds.

References

[1] J.-J. Chen, W.-H. Huang, and C. Liu. k^2Q: A quadratic-form response time and schedulability analysis framework for utilization-based analysis. *Computing Research Repository (CoRR)*, abs/1505.03883, 2015. extended report of the paper in RTSS 2016.

[2] J.-J. Chen, W.-H. Huang, and C. Liu. k^2U: A general framework from k-point effective schedulability analysis to utilization-based tests. *Computing Research Repository (CoRR)*, abs/1501.07084, 2015. extended report of the paper in RTSS 2015.

[3] J.-J. Chen, W.-H. Huang, and C. Liu. k2U: A general framework from k-point effective schedulability analysis to utilization-based tests. In *Real-Time Systems Symposium, RTSS*, pages 107–118, 2015.

[4] J.-J. Chen, W.-H. Huang, and C. Liu. k2Q: A quadratic-form response time and schedulability analysis framework for utilization-based analysis. In *Real-Time Systems Symposium, RTSS*, 2016.

[5] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

Make-to-Order Integrated Scheduling and Distribution*

Yossi Azar† Amir Epstein‡ Łukasz Jeż§(Speaker) Adi Vardi¶

Production and distribution are fundamental operational functions in supply chains. Some large retailers sell vast amount of different products while an increasing number of companies adopt make-to-order business models in which products are custom-made and quickly delivered to the customers directly from the factory. In either case, there is little to no product inventory. Scheduling productions and distribution jointly allows optimizing operational performance by leveraging the tradeoff between various costs, total revenue, and time between order placement and its delivery, thus reducing inventory levels and increasing responsiveness to customers. Most integrated production and distribution scheduling models consider the offline setting with full information about future orders but such assumption is non-realistic for the make-to-order business models.

Therefore, we study scheduling orders and their subsequent distribution to customers *online*, as they arrive over time. In our model, n customers are to be served by a manufacturer's m identical machines. Each customer i releases jobs (orders) over time, each with a specified processing time p_{ij}. The manufacturer must process all jobs, allowing preemption, and deliver finished jobs (products) to respective customers. A completed job has to be delivered to the customer but not necessarily immediately. Each customer i has a non-negative delivery cost D_i that does not depend on the number of jobs delivered in the same delivery (batch). Furthermore, deliveries to different customers cannot be combined. The goal is to minimize the total cost, which is the sum of the total *lead time*, i.e., time from a job's release till its delivery, and the total delivery cost. To this end, the manufacturer may postpone the delivery of completed jobs and aggregate them in batches. We let the minimum and maximum job processing time by $p_{\min} = \min_{i,j} p_{ij}$, $p_{\max} = \max_{i,j} p_{ij}$, and similarly, $D_{\min} = \min_i D_i$, $D_{\max} = \max_i D_i$. Finally, we let $\rho = \frac{p_{\max}}{p_{\min}}$ and $\Delta = \frac{D_{\max}}{D_{\min}}$.

Interestingly, this generalizes two fundamental problems in computer science that received much attention. When all delivery costs are zero, the problem reduces to minimizing the total flow time on parallel identical machines, see [3, 6]. On the other hand, when all jobs have zero processing time, it reduces to independent instances of the TCP Acknowledgment problem [4, 5], one per customer.

Our and Previous Results

We provide an $O(\log m + \log \Delta + \log \rho)$-competitive algorithm for the integrated production and distribution scheduling problem, which improves exponentially on (both)

*Previously appeared in Proceedings of SODA 2016.

†azar@tau.ac.il. School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel.

‡amire@il.ibm.com. IBM Research-Haifa, Haifa, Israel.

§lje@cs.uni.wroc.pl. Inst. Computer Science, U. Wrocław, 15 Joliot-Curie 50-383 Wrocław, Poland.

¶adi.vardi@gmail.com. School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel.

previous upper bound guarantees even for a single machine: of the two, one was $\rho+3$ [1] and the other $2\sum_i D_i/\min_i D_i$ [2], which is $\Omega(n+\Delta)$.

Moreover, we prove a lower bound of $\Omega(\frac{\log\Delta}{\log\log\Delta})$, which holds for any number of machines, thus refuting the conjecture of Averbakh et al. [1] that an $O(1)$-competitive algorithm for a single machine exists. As additional lower bounds of $\Omega(\log\rho)$ and $\Omega(\log m)$ follow from the problem of flow time minimization on identical machines [6], together these imply that our algorithm is essentially optimal.

Upper Bound Overview

Any algorithm for our model must have two components: job scheduling (i.e., which jobs to process at each point of time) and delivery scheduling (i.e., when to deliver jobs to the customer), which cannot be decoupled: On the one hand, we prefer to process jobs with small processing time in order to reduce the flow time (time between the job's release and completion time), on the other hand, we prefer to process jobs from the same client, especially if their delivery cost is relatively large, in order to deliver them together in a single batch. Hence, the challenge is to balance between the flow time, *lag time* (defined as the time between a job being completed and delivered) and delivery cost.

For a single machine and a single customer, the following elegant lgorithm, abbreviated as $SRPT_D$, is 2-competitive[2]: The jobs are scheduled according to SRPT (shortest remaining processing time); each time the total lag time of completed undelivered jobs equals the delivery cost, these jobs are delivered to the customer.

Using $SRPT_D$ for multiple customers (even for a single machine) seems to yield linear competitive ratio [2]. Hence, we develop a different algorithm, which nevertheless uses $SRPT_D$ to create batches of jobs for each customer. For job scheduling though, our algorithm uses the non-migratory scheduling algorithm presented in [3], which is known to achieve optimal competitive ratio for flow time minimization. This algorithm partitions the jobs into classes according to job size. It does not specify how to choose between jobs of the same class. As we aim to decrease the number of interleaved batches, our specific realization of the algorithm activates a single batch from each class on a machine, and processes only the jobs from active batches.

We note that when the processing schedule is given, the deliveries can be taken care of in a nearly optimal way. Specifically, one can easily observe that delivering all completed undelivered jobs of each customer once they accumulate a lag time equal to the delivery cost of that customer yields a 2-competitive delivery schedule (TCP Acknowledgment [4, 5]). However, it is difficult to analyze the combined algorithm (scheduling and delivery), since the job schedule produced by our algorithm may be far from optimal. Therefore, we analyze a "virtual" delivery procedure described below.

The main challenge of delivery scheduling is handling interleaved batches. Although in $SRPT_D$ simulation each batch accumulates a lag time equal to customer's delivery cost, the actual accumulated lag time of jobs in the batch as processed by our algorithm may be large due to interleaving with batches of other customers; note that a batch is not completed if preempted by other batches. Our algorithm handles this issue as follows: While the number of undelivered jobs in a batch is large, each time that the number of active jobs (released but not yet completed) is close to the number of completed undelivered jobs, the latter are delivered to the customer — this way their lag time be charged to the flow time of the former jobs. Moreover, the number of deliveries per batch is poly-logarithmic in m, ρ, and Δ.

When the number of active jobs in a batch drops below a certain threshold, the delivery pattern changes. Namely, we partition the batch into sub-batches consisting of jobs of the same size class. Then, there is only a single delivery per sub-batch, once all of its jobs are completed. We charge the lag time of these jobs to the flow time and delivery cost of other batches using an exponential charging scheme. We believe that this charging scheme can be used in related problems. Specifically, we charge each completed undelivered job to the flow time and delivery cost of all jobs from lower classes (since a job can be preempted only by a job from a lower class). A job from a certain class is charged with an appropriate weight, exponentially decreasing with the difference between the classes of the completed undelivered job and the preempting jobs.

Lower Bound Overview

Our lower bound is based on the following idea. There are only two customers, one with unit delivery cost and unit processing time jobs, and one with large delivery cost and jobs with large processing time. The lower bound consists of up to a logarithmic number of phases. The number of released unit processing time jobs increase exponentially with the phase, while the number of remaining large jobs decreases exponentially with the phase. If there ever is a moment such that, due to prioritizing the large jobs, the online algorithm has significantly more jobs than an algorithm that prioritizes small jobs, than the ratio between the number of their jobs can translated into a lower bound on the competitive ratio, as the former can be maintained by issuing a unit processing time job at every time unit for a substantial amount of time. Naturally, this results in the same ratio between the flow times of the algorithms, and in such case the overall cost is dominated by the flow time. Thus, a competitive online algorithm is forced to prioritize small jobs. As these arrive over time in successive phases, such algorithm starts processing the large jobs in each phase only to switch to the small jobs in the successive one. As the delivery cost for large jobs is large, the net result is that the algorithm accumulates large cost (lead time + delivery cost) for these jobs, which again dominates the overall cost.

References

[1] I. Averbakh and M. Baysan. Approximation algorithm for the on-line multi-customer two-level supply chain scheduling problem. *Oper. Res. Lett.*, 41(6):710–714, 2013.

[2] I. Averbakh and Z. Xue. On-line supply chain scheduling problems with preemption. *European Journal of Operational Research*, 181(1):500–504, 2007.

[3] B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing the flow time without migration. *SIAM Journal on Computing*, 31(5):1370–1382, 2002.

[4] D. R. Dooly, S. A. Goldman, and S. D. Scott. TCP dynamic acknowledgment delay: Theory and practice (extended abstract). In *Proc. of the 30th Annual ACM Symp. on the Theory of Computing (STOC)*, pages 389–398, 1998.

[5] D. R. Dooly, S. A. Goldman, and S. D. Scott. On-line analysis of the TCP acknowledgment delay problem. *J. ACM*, 48(2):243–273, 2001.

[6] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. *J. Comput. Syst. Sci.*, 73(6):875–891, 2007.

Competitive Greedy Algorithms for Stochastic Unrelated Machine Scheduling

Varun Gupta [*] Benjamin Moseley [†] Marc Uetz (Speaker) [‡]
Qiaomin Xie [§]

1 Introduction

A well studied class of problems is scheduling a set of n nonpreemptive jobs that arrive over time on m unrelated machines with the objective of minimizing the total weighted completion time. Here, unrelated machines refers to the fact that the matrix that describes the processing times of all jobs on all machines can have rank larger than 1. The offline version of that problem is denoted $\mathrm{R} \,|\, r_j \,|\, \sum w_j C_j$ in the three-field notation of Graham et al. [3], and it has always been a cornerstone problem for the development of new techniques in the design of (approximation) algorithms.

We here address the online version of that problem with stochastic jobs. Specifically, we are interested in online scheduling of non-preemptive, stochastic jobs in an unrelated machine environment to minimize the total weighted completion time. That means that this paper addresses the same problem as [5], however not for identical machines, but for the most general, *unrelated* machines model. In the stochastic unrelated machine setting, the scheduler is given a probability distribution of a job's processing time which is machine-dependent, and there need not be any correlation between the jobs' processing time distributions on different machines.

A priori, it is not clear that there should exist an algorithm with small competitive ratio. Prior work for the offline problem with stochastic jobs requires sophisticated linear [6] or convex [2] programming relaxations. Good candidates for online algorithms, however, should be simple and combinatorial. Even discovering an offline algorithm that is combinatorial remains a target.

We settle this question. For the case without nontrivial release dates we show:

Theorem 1 *There is a $(8 + 4\Delta)$-competitive greedy algorithm for online scheduling of stochastic jobs to minimize the expected total weighted completion times $\mathbb{E}[\sum_j w_j C_j]$.*

Here, Δ is an upper bound on the squared coefficients of variation of the procesing time distributions. For the case with nontrivial release dates r_j we show:

Theorem 2 *There is a $(144 + 72\Delta)$-competitive greedy algorithm for online scheduling of stochastic jobs to minimize the expected total weighted completion times $\mathbb{E}[\sum_j w_j C_j]$.*

[*] varun.gupta@chicagobooth.edu, University of Chicago
[†] bmoseley@wustl.edu, Washington University in St. Louis
[‡] m.uetz@utwente.nl, University of Twente
[§] qxie3@illinois.edu, University of Illinois at Urbana-Champaign

2 Greedy Algorithm and Sketch of Proof

The algorithm we analyze is the most natural greedy algorithm one might imagine; it is the same as in [5], only for unrelated machines.
GREEDY: Whenever a new job j is presented to the algorithm, compute for each of the machines i the *instantaneous expected increase* if the jobs already present on each machine where to be scheduled in non-increasing order of the ratios weight over expected processing time. Assign the job to one of the machines where this quantity is minimal. Once all jobs have arrived and are assigned, they will be sequenced in non-increasing order of ratios weight over expected processing time, which is known to be optimal conditioned on the given assignment. For the case with nontrivial release dates r_j, the algorithm is more complicated and inserts additional idle time; for details see the full version of this extended abstract [4]. The analysis, inspired by the "dual fitting" argument as suggested in [1], goes essentially in three steps:

1. set up an extended version of the LP-relaxation in [6, §8], with variables y_{ijt} denoting the probability for a job j to be in process on machine i at time t,
2. "de-stochastify" this LP-relaxation, this simplifies the LP, but yields a lost of $O(\Delta)$ in optimal LP solution value,
3. construct a solution to the LP dual of the simplified LP, that allows analyzing GREEDY, showing it's performance is within a factor $O(1)$ of the simplified LP.

Acknowledgements.

This work was done while all four authors were with the Simons Institute at UC Berkeley. The authors wish to thank the institute for the financial support and the organizers of the semester on "Algorithms & Uncertainty" for providing a very stimulating atmosphere.

References

[1] S. Anand, N. Garg, and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proc. 23rd SODA*, pages 1228–1241, 2012.

[2] S. Balseiro, D. Brown, and C. Chen. Static routing in stochastic scheduling: Performance guarantees and asymptotic optimality. Tech. Rep., 2016.

[3] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[4] V. Gupta, B. Moseley, M. Uetz, and Q. Xie. Stochastic Online Scheduling on Unrelated Machines. In *Proc. 19th IPCO*, to appear, 2017.

[5] N. Megow, M. Uetz, and T. Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.

[6] M. Skutella, M. Sviridenko, and M. Uetz. Stochastic scheduling on unrelated machines. *Mathematics of Operations Research*, 41(3):851–864, 2016.

How to Allocate Prices to Random Customers?

José Correa [*] Patricio Foncea [†] Ruben Hoeksma [‡]
Tim Oosterwijk (Speaker) [§] Tjark Vredeveld [¶]

1 Introduction

Posted price mechanisms constitute an attractive and widely applicable way of selling items to strategic consumers. In this context, a seller schedules take-it-or-leave-it offers to costumers, and therefore strategic behavior simply vanishes. This type of mechanism has been vastly studied, particularly in the marketing community. In recent years, there has been a significant effort to understand the expected revenue of the outcome generated by different posted price mechanisms when compared to that of the optimal auction [1, 2].

In its simplest form, the problem we consider can be described as follows. A monopolist is allocating a single item to one buyer from a set of known potential buyers. The seller places no value on the item, while the buyers have independent, not necessarily identical, random valuations for the item. The main question is to design a mechanism maximizing the revenue of the seller. This question was answered in a seminal paper by Myerson [5], and the solution is, in some situations, a remarkably simple mechanism. However, in many situations the mechanism is hard to implement, and the mechanism of choice turns out to be a simple posted price mechanism. A common example of this practice is so called *direct mail campaigns*, in which the seller contacts its potential buyers directly and offers each one a certain price for the item. The item is then sold to the first consumer who accepts the offer.

In this paper, we investigate the performance of posted price mechanisms to sell a single item to a given set of customers who arrive in a random unknown order. We consider two different models which share the property that each customer will be offered the item at most once. Upon receiving an offer, a customer immediately decides whether to buy the item at that price, in which case we allocate the item to her, or to pass and simply not buy, in which case she leaves the system. The *nonadaptive* model considers the situation in which all offers have to be scheduled beforehand, and customers respond

[*] correa@uchile.cl. Departamento de Ingenieria Industrial, Universidad de Chile, Republica 701, Santiago, Chile.

[†] patricio.foncea@ug.uchile.cl. Departamento de Ingenieria Industrial, Universidad de Chile, Republica 701, Santiago, Chile.

[‡] rhoeksma@dim.uchile.cl. Center for Mathematical Modeling, Universidad de Chile, Beauchef 851, Santiago, Chile.

[§] t.oosterwijk@maastrichtuniversity.nl. School of Business and Economics, Maastricht University, Tongersestraat 53, Maastricht, The Netherlands.

[¶] t.vredeveld@maastrichtuniversity.nl. School of Business and Economics, Maastricht University, Tongersestraat 53, Maastricht, The Netherlands.

in random order, akin to direct mail campaigns. The *adaptive* model considers a situation in which the seller may adapt the offer. Here, customers again arrive in random order. Whenever a customer arrives, she is offered the item at a price, which the seller may base on the customer he is offering to, as well as the customers who already rejected earlier offers. He allocates the item to her for the offered price if she accepts.

2 Problem description

A seller has a single item to allocate to one customer from a given set of customers $\mathcal{I}$. We assume that the seller has no value for keeping the item. Customers have independent random valuations for the item with customer $i \in \mathcal{I}$ valuing the item at v_i, drawn from distribution $F_i(\cdot)$. The customers arrive in (uniform) random order, and the goal of the seller is to maximize his expected revenue. We consider two scenarios.
Nonadaptive: The seller plans prices $p_i \geq 0$ for all $i \in \mathcal{I}$ beforehand, with the goal of maximizing his expected revenue, defined as

$$\sum_{i \in \mathcal{I}} p_i \mathbb{P}_{\sigma,v} \left[i = argmin_{j \in \mathcal{I}} \{\sigma(j) \,|\, v_j \geq p_j\} \right] ,$$

where the probability is taken over the arrival permutation σ and the customers' valuations v.
Adaptive: The seller offers each customer a price as she arrives. So, the seller sets functions $p_i : 2^{\mathcal{I}} \to \mathbb{R}$ for each customer i, such that, if S is the set of customers who already arrived and declined the offer, $p_i(S)$ is the price offered to customer i if she is next to arrive. For an arrival permutation σ, we denote $p_i(\sigma) = p_i(\{\sigma^{-1}(1), \ldots, \sigma^{-1}(\sigma(i) - 1)\})$, and therefore we can write the seller's expected revenue as

$$\mathbb{E}_\sigma \left[\sum_{i \in \mathcal{I}} p_i(\sigma) \mathbb{P}_v \left[i = argmin_{j \in \mathcal{I}} \{\sigma(j) \,|\, v_j \geq p_j(\sigma)\} \right] \right] ,$$

where the expectation is taken over the arrival permutation σ, and the probability is taken over the customers' valuations v.

3 Our results

As our main result, we prove the existence of a nonadaptive posted price mechanism that guarantees an expected revenue within a factor $1 - 1/e$ of that of Myerson's optimal auction. On the one hand, this bound matches the well known result of Chawla et al. [2], who designed a *sequential* posted price mechanism with the same approximation guarantee. Although their mechanism is also nonadaptive in the sense that the selected prices are planned a priori, it has the power to choose the arrival order of the customers. Thus, making it easier to extract revenue by offering to *good* customers first. On the other hand, this bound also matches the approximation guarantee obtained by Esfandiari et al. [4], who also consider the random arrival model, but in their mechanism the sequence of prices depends on the arrival order of customers, and it is therefore adaptive (according to the definition in this paper). Besides the natural application of our nonadaptive setting, it is therefore interesting to note that one can achieve this approximation factor in the random arrival model without using adaptivity. Also, as opposed to previous results, we prove that the bound of $1 - 1/e$ is best possible for our setting.

Theorem 1 *For any given set of potential customers $\mathcal{I}$, there exists a nonadaptive posted price mechanism that achieves an expected revenue of at least a $1 - 1/e$ fraction of that of Myerson's auction on $\mathcal{I}$.*

To prove the theorem, we prove a *basic lemma* about Bernoulli random variables. We consider a continuous relaxation of the maximization problem used to determine the approximation guarantee. Then we guess a solution in which we offer to each accepting customer with some instance-dependent probability. We proceed to look for the worst possible instance by applying the first order optimality conditions of a nonlinear problem. These conditions reveal some structural insight on what a worst case instance looks like. Using this, we are finally able to obtain the desired bound. Theorem 1 follows from the basic lemma with fairly little extra work. The basic tool for this is a fundamental result by Chawla et al. [2, Lemma 4] that upper bounds the revenue of the optimal auction. We also show that the lemma generalizes to arbitrary random variables. This implies a variant of the prophet inequality that was recently developed by Esfandiari et al. [4].

To complement our results, we provide instances that show that the bound in Theorem 1 is tight. In particular, we show that even with *independent identically distributed* (*i.i.d.*) customer valuations the bound of Theorem 1 cannot be beaten. Therefore adaptivity is necessary to go beyond $1 - 1/e$ even with i.i.d. distributions. For this setting we show the following result.

Theorem 2 *For any given set of potential customers $\mathcal{I}$ whose values are independent and identically distributed, there exists an adaptive posted price mechanism that achieves an expected revenue of at least a* 0.745 *fraction of that of Myerson's auction on $\mathcal{I}$.*

We do not think this last result is tight. The best upper bound known for the i.i.d. case is due to Blumrosen et al. [1], who prove that no algorithm can achieve a fraction of at least 0.79. We remark here that recent work of Duetting et al. [3] also studies the benefit of adaptivity in the i.i.d. case, but from a different perspective.

References

[1] L. Blumrosen and T. Holenstein. Posted prices vs. negotiations: An asymptotic analysis. In *Proceedings of the 9th ACM Conference on Electronic Commerce*, EC '08, pages 49–49. ACM, 2008.

[2] S. Chawla, J. D. Hartline, D. L. Malec, and B. Sivan. Multi-parameter mechanism design and sequential posted pricing. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 311–320. ACM, 2010.

[3] P. Duetting, F. A. Fischer, M. Klimm. Revenue Gaps for Discriminatory and Anonymous Sequential Posted Pricing. *Manuscript*, 2016.

[4] H. Esfandiari, M. Hajiaghayi, V. Liaghat, and M. Monemizadeh. Prophet secretary. In *Algorithms-ESA 2015*, pages 496–508. Springer, 2015.

[5] R. B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1): 58–73, 1981.

Competitive Packet Routing with Priority Lists

Tobias Harks [*] Britta Peis [†] Daniel Schmand [†]
Bjoern Tauer (Speaker) [†] Laura Vargas Koch [†]

1 Introduction

A fundamental combinatorial optimization problem that has received considerable attention in the past is *packet routing* in graphs (cf. [7, 9, 10, 11, 12]). We are given a set of packets, which may, for example, correspond to unit-sized messages/bits in a communication network. Originated at possibly different start nodes, the goal is to transfer all packets as fast as possible to their respective destination nodes. It is assumed that each edge is equipped with a capacity (or bandwidth) and a travel time. A prominent variant is *discrete store-and-forward packet routing*, where every node can store arbitrarily many packets, but only a limited number can enter an edge simultaneously at each discrete time step, see [9]. Applications can be found in routing models used in synchronized systems with a given clock-rate. Imagine, for example, a chip with components and wires corresponding to the nodes and edges, respectively, of the associated graph, and with a centralized clock rate for the chip given by a crystal oscillator.

In this work, we focus on *selfish* or *competitive* packet routing using the discrete store-and-forward packet routing model. We are given a multi- or single-commodity network, where the commodities are specified by a source and a sink node and represent the players that route rational and selfishly one packet from their source to their sink through the network. Each edge of the network is endowed with an integral travel time and an integral capacity. The capacity of the edge defines the number of players that may enter the edge simultaneously. For each edge, we are given a priority list (i.e., an ordered list of the players) to resolve conflicts whenever more than capacity many players seek to enter that edge at the same point in time. All players are ready to start right from the beginning (i.e., there are no release dates) and aim to minimize their respective arrival time at the sink. Since the outcome of this competitive situation intrinsically depends on the priority lists employed on the edges, the problem of finding *good* priority lists renders into a coordination mechanism design problem. See [3] for the first landmark paper and several follow ups [1, 4, 6].

[*] tobias.harks@math.uni-augsburg.de. Department of Mathematics, University of Augsburg, 86135 Augsburg, Germany.

[†] (britta.peis, daniel.schmand, bjoern.tauer, laura.vargas)@oms.rwth-aachen.de. School of Business and Economics, RWTH Aachen University, 52072 Aachen, Germany.

2 Our Contribution

We explore properties of selfish discrete store-and-forward packet routing with *priority based* scheduling policies. We consider local priority lists and global priority lists (that may or may not depend on the edge). Furthermore we distinguish between a multi-commodity and a single-commodity network, where the same source and sink pair is dedicated to every player. We obtain the following results.

Price of Anarchy/Stability. For global priority lists and multi-commodity instances we show that the price of stability (PoS for short) is upper bounded by $\frac{n+1}{2}$ and the price of anarchy (PoA) is upper bounded by $\frac{5}{6} + \frac{n^2}{6}$, where n denotes the number of players. For global priority lists and symmetric games (that is, all packets travel from a common source to a common sink) the PoS is one, while the PoA for these games is exactly $\frac{n+1}{2}$ and this bound holds even for multiple sources and a single sink.

For general local priority lists (that is, the predefined order may be different among edges) and multi-commodity games, we derive that the PoA is in between $\frac{T}{4}$ and $6T^2$, where T is a kind of *dilation* of the graph, i.e., the maximal length of a path, where edges with travel time 0 contribute 1 to the path. This result is obtained via adapting the primal-dual technique introduced by Kulkarni and Mirrokni [8].

Computational Complexity. We turn to the question of computing *optimal* priority lists, that is, priority lists that induce best possible social optima or Nash equilibria. It turns out that computing an optimal local priority list with respect to minimizing a social optimum in symmetric instances can be done in polynomial time. The question of computing an optimal local priority list with respect to minimizing the inefficiency of Nash equilibria for symmetric instances remains open. For multi-commodity graphs, we show that both problems are APX-hard. Note that this is the first hardness result for the underlying coordination mechanism design problem and complements several approximability results for the tree case recently derived by Bhattacharya et al. [2]. Technically, we adapt a construction of Peis et al. [11], where it is shown that the problem to compute a schedule minimizing the makespan (the latest arrival of any packet) is APX-hard. Our result implies that the problem of defining global as well as local priority lists for minimizing the cost of any Nash equilibrium or of any social optimum is APX-hard.

We finally derive several further hardness results for our model: In multi-commodity games with local priority lists it is NP-hard to compute a pure Nash equilibrium. Moreover, it is NP-hard to compute a best response in symmetric games with local priority lists. These results are obtained by adapting the reduction described in Hoefer et al. [5] which is used to prove hardness in a more general setting.

An open question is the complexity of computing a pure Nash equilibrium in a symmetric game with local priority lists. Attention should be paid to the fact, that there exist competitive packet routing games with local priority lists which improve the PoA in contrast to any global priority list. For example there exist a local priority list such that the PoA of the n-th Braess graph is equal to 1, whereas it is $\frac{n+1}{2}$ for every global priority list. For global priority lists, we get an efficient Dijkstra-type algorithm for computing a best response and, thus, a pure Nash equilibrium.

References

[1] Yossi Azar, Lisa Fleischer, Kamal Jain, Vahab S. Mirrokni, and Zoya Svitkina. Optimal coordination mechanisms for unrelated machine scheduling. *Operations Research*, 63(3):489–500, 2015.

[2] Sayan Bhattacharya, Janardhan Kulkarni, and Vahab S. Mirrokni. Coordination mechanisms for selfish routing over time on a tree. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 186–197, 2014.

[3] George Christodoulou, Elias Koutsoupias, and Akash Nanavati. Coordination mechanisms. *Theoretical Computer Science*, 410(36):3327–3336, 2009.

[4] Richard Cole, José R. Correa, Vasilis Gkatzelis, Vahab S. Mirrokni, and Neil Olver. Inner product spaces for minsum coordination mechanisms. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 539–548, 2011.

[5] Martin Hoefer, Vahab S. Mirrokni, Heiko Röglin, and Shang-Hua Teng. Competitive routing over time. *Theoretical Computer Science*, 412(39):5420–5432, 2011.

[6] Nicole Immorlica, Li Erran Li, Vahab S Mirrokni, and Andreas S Schulz. Coordination mechanisms for selfish scheduling. *Theoretical Computer Science*, 410(17):1589–1598, 2009.

[7] Ronald Koch, Britta Peis, Martin Skutella, and Andreas Wiese. Real-time message routing and scheduling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, pages 217–230, 2009.

[8] Janardhan Kulkarni and Vahab S. Mirrokni. Robust price of anarchy bounds via LP and fenchel duality. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1030–1049, 2015.

[9] Frank T. Leighton, Bruce Maggs, and Satish Rao. Packet routing and job-shop scheduling in $\mathcal{O}$(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[10] Frank T. Leighton, Bruce Maggs, and Andrea W Richa. Fast algorithms for finding $\mathcal{O}$(congestion + dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999.

[11] Britta Peis, Martin Skutella, and Andreas Wiese. Packet routing: Complexity and algorithms. In *Approximation and Online Algorithms, 7th International Workshop, WAOA 2009, Copenhagen, Denmark, September 10-11, 2009. Revised Papers*, pages 217–228, 2009.

[12] Aravind Srinivasan and Chung-Piaw Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing*, 30(6):2051–2068, 2001.

Scheduling Maintenance Jobs in Networks

Fidaa Abed [*] Lin Chen [†] Yann Disser [‡] Martin Groß[§]
Nicole Megow [¶] Julie Meißner [‖] Alexander T. Richter [**]
Roman Rischke (Speaker) [††]

1 Introduction

Transportation and telecommunication networks are important backbones of modern infrastructure and have been a major focus of research in combinatorial optimization and other areas. Research on such networks usually concentrates on optimizing their usage, for example by maximizing throughput or minimizing costs. In the majority of the studied optimization models, it is assumed that the network is permanently available, and our choices only consist in deciding which parts of the network to use at each point in time. Practical transportation and telecommunication networks, however, can generally not be used non-stop. Be it due to wear-and-tear, repairs, or modernizations of the network, there are times when parts of the network are unavailable. We study how to coordinate such maintenance in different parts of the network to ensure connectivity.

While network problems and scheduling problems individually are fairly well understood, the combination of both areas that results from scheduling network maintenance has only recently received some attention [2, 3, 7, 1, 5] and is theoretically hardly understood.

2 Problem Definition

We study connectivity problems which are fundamental in this context. In these problems, we aim to schedule the maintenance of edges in a network in such a way as to preserve connectivity between two designated vertices. Given a network and maintenance jobs with processing times and feasible time windows, we need to decide on the temporal allocation of the maintenance jobs. While a maintenance on an edge is performed, the edge is not available. We distinguish between MinConnect, the problem in which we minimize the total time in which the network is disconnected, and MaxConnect, the problem in which we maximize the total time in which it is connected.

[*] fabed@uj.edu.sa. University of Jeddah, Jeddah, Saudi Arabia.
[†] chenlin198662@gmail.com. University of Houston, Texas, USA.
[‡] disser@mathematik.tu-darmstadt.de. TU Darmstadt, Darmstadt, Germany.
[§] gross@math.tu-berlin.de. TU Berlin, Berlin, Germany.
[¶] nicole.megow@uni-bremen.de. University of Bremen, Bremen, Germany.
[‖] jmeiss@math.tu-berlin.de. TU Berlin, Berlin, Germany.
[**] a.richter@tu-bs.de. TU Braunschweig, Braunschweig, Germany.
[††] rischke@ma.tum.de. TU München, München, Germany.

In both of these problems, we are given an undirected graph $G = (V, E)$ with two distinguished vertices $s^+, s^- \in V$. We assume w.l.o.g. that G is simple. Every edge $e \in E$ needs to undergo $p_e \in Z_{\geq 0}$ time units of maintenance within the time window $[r_e, d_e]$ with $r_e, d_e \in Z_{\geq 0}$, where r_e is called the release date and d_e is called the deadline of the maintenance job for edge e. An edge $e = \{u, v\} \in E$ that is maintained at time t, is not available at t in the graph G. We consider preemptive and non-preemptive maintenance jobs. If a job must be scheduled non-preemptively then, once it is started, it must run until completion without any interruption. If a job is allowed to be preempted, then its processing can be interrupted at any time and may resume at any later time.

A *schedule* S for G assigns the maintenance job of every edge $e \in E$ to a single time interval (if non-preemptive) or a set of disjoint time intervals (if preemptive) $S(e) := \{[a_1, b_1], \ldots, [a_k, b_k]\}$ with $r_e \leq a_i \leq b_i \leq d_e$, for $i \in [k]$ and $\sum_{[a,b] \in S(e)} (b - a) = p_e$. We define $T := \max_{e \in E} d_e$ as our time horizon. We do not limit the number of simultaneously maintained edges.

For a given maintenance schedule, we say that the network G is *disconnected at time t* if there is no path from s^+ to s^- in G at time t, otherwise we call the network G *connected at time t*. The goal is to find a maintenance schedule for the network G so that the total time where G is disconnected is minimized (MinConnect). We also study the maximization variant of the problem, in which we want to find a schedule that maximizes the total time where G is connected (MaxConnect).

3 Our Contribution

Preemptive Scheduling. For *preemptive* maintenance jobs, we show the following.

Theorem 1 *Both* MaxConnect *and* MinConnect *with preemptive jobs can be solved optimally in polynomial time on arbitrary graphs.*

This result crucially requires that we are free to preempt jobs at arbitrary points in time. Under the restriction that we can *preempt* jobs only at *integral points in time*, the situation changes drastically.

Theorem 2 MaxConnect *with preemption only at integral time points is NP-hard and does not admit a $(2 - \epsilon)$-approximation algorithm for any $\epsilon > 0$, unless $P = NP$.* MinConnect *with preemption only at integral time points is inapproximable.*

We remark that this is true even for unit-size jobs. This complexity result is interesting and may be surprising, as it is in contrast to results for standard scheduling problems, without an underlying network. Here, the restriction to integral preemption typically does not increase the problem complexity when all other input parameters are integral.

Non-Preemptive Scheduling. For *non-preemptive* instances, our main result is as follows.

Theorem 3 *Unless $P = NP$, there is no $(c\sqrt[3]{|E|})$-approximation algorithm for non-preemptive* MaxConnect*, for some constant $c > 0$.* MinConnect *is inapproximable even on disjoint paths between the two nodes s^+ and s^-, unless $P = NP$.*

On the positive side, we give an $(\ell+1)$-approximation algorithm for MaxConnect on general graphs, where $\ell := |\{d_e - p_e : e \in E\}|$ is the number of distinct latest start times.

Power of Preemption. We use the notion *power of preemption* to capture the benefit of allowing arbitrary job preemption. The power of preemption is a commonly used measure for the impact of preemption in scheduling. It is defined as the maximum ratio of the objective values of an optimal non-preemptive and an optimal preemptive solution.

Theorem 4 *The power of preemption is* $\Theta(\log |E|)$ *for* MinConnect *on a path and unbounded for* MaxConnect *on a path.*

This is in contrast to other scheduling problems, where the power of preemption is constant, e.g. [4, 8].

Mixed Scheduling. For *mixed* instances, which have both preemptive and non-preemptive jobs, we obtain the following.

Theorem 5 MaxConnect *and* MinConnect *with preemptive and non-preemptive maintenance jobs are weakly NP-hard, even on a path.*

This hardness result is of particular interest, as both purely non-preemptive and purely preemptive instances can be solved efficiently on a path (see Theorem 1 and [6]). Furthermore, we give a simple 2-approximation algorithm for mixed instances of MinConnect.

References

[1] Bley, A., Karch, D., D'Andreagiovanni, F.: WDM fiber replacement scheduling. Electronic Notes in Discrete Mathematics, 41:189–196, 2013.

[2] Boland, N., Kalinowski, T., Kaur, S.: Scheduling arc shut downs in a network to maximize flow over time with a bounded number of jobs per time period. Journal of Combinatorial Optimization, 1–21, 2015.

[3] Boland, N., Kalinowski, T., Waterer, H., Zheng, L.: Scheduling arc maintenance jobs in a network to maximize total flow over time. Discrete Applied Mathematics, 163:34–52, 2014.

[4] Correa, J.R., Skutella, M., Verschae, J.: The power of preemption on unrelated machines and applications to scheduling orders. Mathematics of Operations Research, 37(2):379–398, 2012.

[5] Flammini, M., Monaco, G., Moscardelli, L., Shachnai, H., Shalom, M., Tamir, T., Zaks, S.: Minimizing total busy time in parallel scheduling with application to optical networks. Theoretical Computer Science, 411(40–42):3553–3562, 2010.

[6] Khandekar, R., Schieber, B., Shachnai, H., Tamir, T.: Real-time scheduling to minimize machine busy times. Journal of Scheduling, 18(6):561–573, 2015.

[7] Nurre, S.G., Cavdaroglu, B., Mitchell, J.E., Sharkey, T.C., Wallace, W.A.: Restoring infrastructure systems: An integrated network design and scheduling (INDS) problem. European Journal of Operational Research, 223(3):794–806, 2012.

[8] Schulz, A.S., Skutella, M.: Scheduling unrelated machines by randomized rounding. SIAM Journal on Discrete Mathematics, 15(4):450–469, 2002.

New Structural Results for Bin Packing with a Constant Number of Item Types*

Klaus Jansen † Kim-Manuel Klein (Speaker) ‡

1 Introduction

We consider the classical bin packing problem with d different item sizes $s_1, \ldots, s_d$ and build upon the results by Goemans and Rothvoß [1] to obtain a new polynomial time algorithm for the bin packing problem when d is constant [3]. Therefore, we present new techniques on how solutions of an instance can be modified and we give a new structure theorem that relies on the set of vertices of the underlying integer polytope. As a result of our new structure theorem, we obtain an algorithm for the bin packing problem with running time $|V|^{2^{O(d)}} \cdot enc(I)^{O(1)}$, where V is the set of vertices of the underlying integer knapsack polytope and $enc(I)$ is the encoding length of the bin packing instance. The algorithm is fixed parameter tractable, parameterized by the number of vertices of the integer knapsack polytope $|V|$. This shows that the bin packing problem can be solved efficiently when the underlying integer knapsack polytope has an easy structure, i.e. has a small number of vertices. This is for example the case when all item sizes $s_1, \ldots, s_d$ are of the form $s_i = \frac{1}{a_i}$ for some $a_i \in \mathbb{Z}_{\geq 1}$ (in that case we obtain a running time of $2^{2^{O(d)}} \cdot enc(I)^{O(1)}$ and therefore fpt in d).

Furthermore, we show that the presented bounds of the structure theorem are asymptotically tight. We give a construction of bin packing instances using new structural insights and classical number theoretical theorems which yield the desired lower bound.

2 The Structure Theorem

Given the polytope $\mathcal{P} = \{x \in \mathbb{R}^d \mid Ax \leq c\}$ for some matrix $A \in \mathbb{Z}^{m \times d}$ and a vector $c \in \mathbb{Z}^d$. We consider the integer cone

$$int.cone(\mathcal{P} \cap \mathbb{Z}^d) = \{ \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p \mid \lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d} \}$$

of integral points inside the polytope $\mathcal{P}$. When we choose $\mathcal{P}$ to be the knapsack polytope, i.e. $\mathcal{P} = \{x \in \mathbb{Z}^d_{\geq 0} \mid s^T x \leq 1\}$, then each integral point of the polytope represents one possibility of packing a single bin with items from $s_1, \ldots, s_d$. Hence a vector $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$

*This work was partially supported by DFG Project, Entwicklung und Analyse von effizienten polynomiellen Approximationsschemata für Scheduling- und verwandte Optimierungsprobleme, Ja 612/14-2

†kj@informatik.uni-kiel.de. Department of Computer Science, University of Kiel.

‡kmk@informatik.uni-kiel.de. Department of Computer Science, University of Kiel.

of $int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ represents a packing of the bin packing problem. A long standing open question was, if the bin packing problem can be solved in polynomial time when the number of different item sizes d is constant. This problem was recently solved by Goemans and Rothvoß [1] using structural properties of the integer cone. Essentially, they proved the existence of a distinguished set $X \subset \mathcal{P}$ of bounded size such that for every vector $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$ there exists an integral vector $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$ where most of the weight lies in X.

In our paper we show that a similar structure theorem holds for a rather natural choice of the distinguished set X. Therefore, we consider the so called integer polytope $\mathcal{P}_I$. It is defined by the convex hull of all integer points inside $\mathcal{P}$ (see figure 1) i.e. $\mathcal{P}_I = Conv(\mathcal{P} \cap \mathbb{Z}^d)$. Let V_I be the vertices of the integer polytope $\mathcal{P}_I$ i.e. $\mathcal{P}_I = Conv(V_I)$. Based on the set V_I, we showed the following structure theorem for solutions λ of $int.cone(\mathcal{P} \cap \mathbb{Z}^d)$:

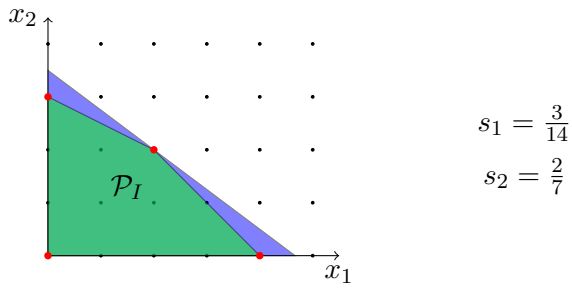


Figure 1: The integer knapsack polyope

Theorem 1 *Let $\mathcal{P} = \{x \in \mathbb{R}^d \mid Ax \leq c\}$ be a polytope with $A \in \mathbb{Z}^{m \times d}, c \in \mathbb{Z}^d_{\geq 0}$ and let $supp(\lambda)$ be the set of non-zero components of λ. Then for any vector $b \in int.cone(\mathcal{P} \cap \mathbb{Z}^d)$, there exists an integral vector $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{P} \cap \mathbb{Z}^d}$ such that $b = \sum_{p \in \mathcal{P} \cap \mathbb{Z}^d} \lambda_p p$ and*

1. $\lambda_p \leq 2^{2^{O(d)}} \quad \forall p \in (\mathcal{P} \cap \mathbb{Z}^d) \setminus V_I$
2. $|supp(\lambda) \cap V_I| \leq d \cdot 2^d$
3. $|supp(\lambda) \setminus V_I| \leq 2^{2d}$

As a consequence of our structure theorem, we obtain an algorithm for the bin packing problem with a running time of $|V_I|^{2^{O(d)}} \cdot \log(\Delta)^{O(1)}$, where Δ is the maximum over all multiplicities b and denominators in s. Since $|V_I| \geq d + 1$ this is an fpt-algorithm parameterized by the number of vertices of the integer knapsack polytope V_I.

Theorem 2 *The bin packing problem can be solved in time $|V_I|^{2^{O(d)}} \cdot (\log \Delta)^{O(1)}$ and hence in* fpt*-time, parameterized by the number of vertices V_I.*

This algorithmic result shows that the bin packing problem can be solved efficiently when the underlying knapsack polytope has an easy structure i.e. has not too many vertices. However, since the total number of vertices is bounded by $O(\log \Delta)^d$ (see [2]) the algorithm has a worst case running time of $(\log \Delta)^{2^{O(d)}}$, which is identical to the running time of the algorithm by Goemans and Rothvoß [1].

Furthermore, we were able complement this result by giving a matching lower bound. We prove that the double exponential bound of the structure theorem is actually tight, even in the mentioned special case of bin packing, when all items sizes $s_1, \ldots, s_d$ are of the form $s_i = \frac{1}{a_i}$ for some $a_i \in \mathbb{Z}_{\geq 1}$

References

[1] M. X. Goemans and T. Rothvoss *Polynomiality for Bin Packing with a Constant Number of Item Types, SODA, 2014, 830–839.*

[2] A.C. Hayes and D.G. Larman. *The vertices of the knapsack polytope, Discrete Applied Mathematics 6.2, 1983, pp. 135–138.*

[3] Klaus Jansen and Kim-Manuel Klein *About the Structure of the Integer Cone and its Application to Bin Packing, SODA, 2017, 1571–1581.*

A Note on Online Machine Minimization

Yossi Azar * Sarel Cohen (Speaker)*

The machine minimization problem is to schedule a set of jobs with specified time intervals on the smallest possible number of machines. Each job j arrives at its release time r_j, has to be processed for p_j time units, and must be completed by its deadline d_j. The goal is to minimize the number of machines the algorithm uses for processing all jobs by their deadlines. In the preemptive model each processed job may be stopped and resumed later, possibly on a different machine. In the online setting, the jobs arrive online and the attributes (r_j, p_j, d_j) of the job are known to the algorithm at the release time r_j.

There are various well-known algorithms for scheduling jobs on machines. For example, EDF (earliest deadline first) schedules the jobs currently in the system which have the earliest deadlines. Another algorithm is LLF (least laxity first) which schedules the jobs currently in the system in increasing order of their laxities (the jobs with the least laxities are scheduled first). At time t (for $r_j \leq t < d_j$), a job j has laxity $\ell_j(t) = (d_j - t - p_j(t))$ where $p_j(t)$ is the remaining processing time of job j at time t.

Let m be the minimum number of machines required in order to process all the jobs by their deadlines in the offline setting, when all the jobs and their attributes (r_j, p_j, d_j) are known in advance. In the uniprocessor case ($m = 1$), if there exists any feasible schedule, then both EDF ([4]) and LLF ([3]) find feasible solutions on a single machine. In the multiprocessor case ($m \geq 2$), any online algorithm must use more than m machines on some inputs ([3]).

Let $P_{\max}$ be the maximum processing time of a job and $P_{\min}$ be the minimum processing time of a job. Phillips, Stein, Torng, and Wein proved in [5] that LLF is $O(\log \frac{P_{\max}}{P_{\min}})$-competitive. They also showed a lower bound of 5/4 on the competitive ratio for any deterministic algorithm, leaving a huge gap of $O(\log \frac{P_{\max}}{P_{\min}})$ on the competitive ratio of LLF. They observed that EDF does not improve the competitive ratio by giving an $\Omega(\log \frac{P_{\max}}{P_{\min}})$ lower bound on EDF.

Nearly two decades later, Chen, Megow and Schewior (SODA 2016) [1] were the first to significantly improve the competitive ratio. They presented an algorithm which is $O(\log m)$-competitive. In particular, for fixed m this yields a constant competitive algorithm. They also showed that a variant of their algorithm is constant competitive when all jobs have processing time windows that are either laminar or agreeable. Chen, Megow and Schewior prove in [2] that job migration (the ability to choose a different machine when resuming a job) significantly affects machine minimization. The number of machines required in a non-migrative solution is unbounded as a function of m (the number of machines required in a migrative solution). Hence, job migration is necessary to achieve good competitive ratio.

*School of Computer Science, Tel-Aviv University. E-mails: azar@tau.ac.il, sarelcoh@post.tau.ac.il.

It is not known if the online machine minimization problem admits an $O(1)$-competitive algorithm or not, which is an important open problem in this field [1, 5, 6].

Our Contribution: As described above, it is a wide open question to narrow the gap between the $O(\log m)$ upper bound and the constant lower bound for the machine minimization problem. We go one step further in narrowing this gap by providing an $O(\frac{\log m}{\log \log m})$-competitive algorithm.

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

1 The Improved Algorithm

We assume m, the optimum number of machines, is known in advance for the online algorithm, otherwise we can use the doubling technique and lose a factor of 4 in the competitive ratio, as proved in Theorem 2.2 in [1].

Our most significant observation is an improvement of Lemma 3.3 in [1] which implies an improved lower bound on the the minimum number of machines used by the optimum algorithm. We need the following definitions. We say that a job is α-loose if $p_j < \alpha(d_j - r_j)$, and otherwise the job is α-tight. For a job j we define its interval as $I(j) = [r_j, d_j)$. For a set S of jobs we define $I(S) := \bigcup_{j \in S} I(j)$, the interval $I(S)$ of the set S is the union of the job intervals of all the jobs in S. For $I = \cup_{i=1}^{k} [a_i, b_i)$ where $[a_1, b_1), \ldots, [a_k, b_k)$ are pairwise disjoint, we define the length of I to be $|I| = \sum_{i=1}^{k} (b_i - a_i)$. Our improved Lemma is:

Lemma 2. *[Proof is in the full paper] Let $S_1, \ldots, S_{\lceil 3m/\alpha \rceil}$ be pairwise disjoint sets of α-tight jobs such that $I(S_1) \subseteq \ldots \subseteq I(S_{\lceil 3m/\alpha \rceil})$. Then $|I(S_{\lceil 3m/\alpha \rceil})| \geq \frac{1}{1-\alpha} |I(S_1)|$.*

The original Lemma in [1] claimed that $|I(S_{\lceil 2m/\alpha \rceil})| \geq 2|I(S_1)|$, so here we get a bound which is more tight by a factor of $\Theta(\frac{1}{1-\alpha})$, which is significant when $1 - \alpha$ approaches 0 as a function of m.

This allows us to improve the lower bound of Theorem 3.1 of [1]. To state the improved lower bound we first need the following definition of a (μ, β)-critical pair.

Definition 3. *Let G be a set of α-tight jobs and let T be a non-empty finite union of time intervals. For some $\mu \in N$ and $\beta \in (0, 1)$, a pair (G, T) is called (μ, β)-critical if: (1) each $t \in T$ belongs to the intervals of at least μ jobs in G (i.e., $|\{j \in G | t \in I(j)\}| \geq \mu$), (2) $|T \cap I(j)| \geq \beta \ell_j$ for any $j \in G$.*

While [1] proved that the optimum algorithm uses at least $m = \Omega(\frac{\mu}{\log \frac{1}{\beta}})$ machines, we prove a stronger lower bound:

Theorem 4. *[Proof is in the full paper] If there exists a (μ, β)-critical pair, then $m = \Omega(\frac{\mu}{\log \frac{1}{\beta}} \cdot \log \frac{1}{1-\alpha})$.*

Corollary 5. *If there exists a (μ, β)-critical pair, then $\mu = O(\frac{m \log \frac{1}{\beta}}{\log \frac{1}{1-\alpha}})$.*

The proof of Theorem 4 is similar to the proof in [1], but we further exploit our stronger Lemma 2 and consider $\alpha = \alpha(m)$ as a function of m rather than a constant. This allows us to choose a better value for α to obtain a better competitive ratio guarantee.

We find a better competitive ratio for scheduling α-tight jobs. We use the algorithm of [1] which we refer to as the "LaxityPartition" algorithm. Their algorithm considers laxities as budgets which are partitioned into $\mu = m'+1$ sub-budgets. In order to process m jobs which are α-tight, their LaxityPartition algorithm opens $O(m \log m)$ machines. We show that less machines are sufficient, more precisely, $O(\frac{m \log m}{\log(1/(1-\alpha))})$ machines. This follows from our Corollary 5, thus obtaining the following Lemma.

Lemma 6. *The LaxityPartition algorithm for α-tight jobs with $0 < \alpha < 1$ is $O(\frac{\log m}{\log(1/(1-\alpha))})$-competitive (by choosing $\mu = \Theta(\frac{m \log m}{\log(1/(1-\alpha))})$).*

Here is the description of the $O(\frac{\log m}{\log \log m})$-competitive algorithm which we call the "CombinedAlgorithm":

- Schedule all α-loose jobs using EDF.
- Schedule all α-tight jobs using the LaxityPartition algorithm.

We show that CombinedAlgorithm is $O(\frac{\log m}{\log \log m})$-competitive. According to Lemma 6, scheduling all α-tight jobs using LaxityPartition is $O(\frac{\log m}{\log(1/(1-\alpha))})$-competitive. According to Theorem 2.3 in [1], scheduling all α-loose jobs using EDF is $O(1/(1-\alpha)^2)$-competitive. Thus, the competitive ratio of CombinedAlgorithm is $O(\max(1/(1-\alpha)^2, \frac{\log m}{\log(1/(1-\alpha))}))$, which is minimized for $1 - \alpha = \sqrt{\frac{\log \log m}{\log m}}$ obtaining theorem 1.

References

[1] Lin Chen, Nicole Megow, and Kevin Schewior. An o(log m)-competitive algorithm for online machine minimization. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 155–163, 2016.

[2] Lin Chen, Nicole Megow, and Kevin Schewior. The power of migration in online machine minimization. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, pages 175–184, New York, NY, USA, 2016. ACM.

[3] M. L. Dertouzos and A. K. Mok. Multiprocessor online scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, Dec 1989.

[4] Michael L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

[5] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 140–149, 1997.

[6] K. Pruhs. 10071 open problems – scheduling. In Susanne Albers, Sanjoy K. Baruah, Rolf H. Möhring, and Kirk Pruhs, editors, *Scheduling*, number 10071 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2010. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Maximizing the Minimum Gap

Marin Bougeret * Guillerme Duvillié (Speaker) † Rodolphe Giroudeau ‡

1 Introduction

In this paper, we consider some scheduling problems using the length of the minimum gap in the schedule as a performance measure. Even though the notion of gap has been introduced with the study of minimum energy scheduling, the study of other natural gap scheduling problems is relatively new.

Throughout this paper, we consider two variants of a problem introduced and left open by Chrobak et al. in [2]. In this problem, a set of jobs with release dates and deadlines have to be scheduled on a single machine while maximizing the length of the minimum idle period between two adjacent jobs. Depending on whether the gaps on length 0 (*i.e.* gaps delimited by two consecutive jobs) are taken into consideration, two variants of this problem can be defined.

2 Problem Formalization

In the following, we are given a single machine and a set of unit length jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. Each job $J \in \mathcal{J}$ can be seen as a couple (r_J, d_J) where r_J represents the release date and d_J the deadline.

We define scheduling as follows.

Definition 1 (Scheduling)**.** *A scheduling is a function $\sigma : \mathcal{J} \to \mathbb{Z}$ that assigns to each $J \in \mathcal{J}$ a slot $\sigma(J)$.*

A scheduling is said proper if and only if no two jobs are scheduled at the same time and every job is scheduled after its release date but before its deadline. More formally:

1. *for each couple of jobs $(J, J') \in \mathcal{J}^2$, $\sigma(J) \neq \sigma(J')$,*
2. *for each job $J \in \mathcal{J}$, $r_J \leq \sigma(J) \leq d_J$*

We now present the definitions of gaps we use in the rest of the paper.

Definition 2 (Strict Gap)**.** *Given a scheduling σ, two consecutive jobs $J, J' \in \mathcal{J}$ define a strict gap of length $|\sigma(J') - \sigma(J) - 1|$.*

*bougeret@lirmm.fr. Department of Computer Science, LIRMM, 161 rue Ada, 34095 Montpellier Cedex 5, France.

†guillerme.duvillie@ulb.ac.be. Department of Computer Science, Universit libre de Bruxelles, Boulevard du Triomphe CP 210/01, B-1050 Bruxelles, Belgium.

‡rgirou@lirmm.fr. Department of Computer Science, LIRMM, 161 rue Ada, 34095 Montpellier Cedex 5, France.

Definition 3 (Gap)**.** *Given a scheduling σ, two consecutive but not adjacent jobs $J, J' \in \mathcal{J}$ define a gap of length $|\sigma(J') - \sigma(J) - 1|$.*

We can now define the following problem:

Optimization Problem 1. MaxMinGap

Input	A single machine, a set of unit jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and two functions $r : \mathcal{J} \to \mathbb{Z}$, $d : \mathcal{J} \to \mathbb{Z}$ defining respectively the release date and the deadline of each job.
Output	A proper scheduling
Objective	Maximize the minimum gap.

We will also consider a variant of this problem called MaxMinStrictGap where the objective is to maximize the minimum strict gap.

3 Results

3.1 MaxMinStrictGap

In a first time we show that the decision version of this problem reduces to the feasibility version of $1|r_j, p_j = p|\sum U_j$. This problem is characterized by a single machine, a set of jobs $\mathcal{J} = \{1, 2, \ldots, n\}$. For each job a release date r_j and a deadline d_j are given, and every job has length p. The objective is to know whether all jobs can be properly scheduled, *i.e.* such that $\forall j \in \mathcal{J}, \sigma(j) \geq s_j$, $\sigma(j) + p \leq d_j$ and no two consecutive jobs overlap.

Theorem 4. *There exists a reduction from* Decision MaxMinStrictGap *to feasibility version of* $1|r_j, p_j = p|\sum U_j$.

Since the feasibility version of $1|r_j, p_j = p|\sum U_j$ can be solved in $\mathcal{O}(n \log n)$ using algorithm of Garey et al. [1], we get the following result.

Corollary 5. MaxMinStrictGap *can be solved in polynomial-time with an algorithm running in $\mathcal{O}(n \log n \log d_{max})$, with d_{max} being the largest release date among the jobs release dates.*

3.2 MaxMinGap

Then, we present a polynomial dynamic programming for the decision version of MaxMinGap. We follow a reasoning scheme similar as the one used by Chrobak et al. [2] to highlight polynomial-time algorithm for other gap scheduling problem. In a first time, we show that we can narrow the research to schedules verifying *Earliest Deadline Property* (EDP).

Proposition 6. *We can restrict our attention to schedules σ that verify EDP, i.e. such that at any time t, either machine is idle, either the job with the smallest deadline among the released jobs is scheduled. If at least two jobs can be scheduled at time t, then the order is chosen arbitrarily.*

Then we show that possible schedules dates of jobs can be chosen in a set of at most $2nn^3$ without losing solutions. Based on these properties, we shows that a dynamic programming can solve the decision version of the problem in time $\mathcal{O}(n^{10})$.

The dynamic programming algorithm is based on the fact that once the job with the largest deadline has been scheduled, remaining jobs can be partitioned into two sets from which we can define two subinstances. We can prove that if both subinstances are positive, so does the original one. It follows that an instance I is positive if and only if there exists a schedule of the job with the largest deadline that splits I into two positive subinstances.

This strategy is thus to try all possible schedule of the job with the largest deadline, split the instance into two smaller instances and apply recursively the strategy in order to get trivially positive or negative instances.

References

[1] Jacques Carlier. Problèmes d'ordonnancement à durées égales. *Questiió: Quaderns d'Estadística, Sistemes, Informatica i Investigació Operativa*, 5(4):219–228, 1981.

[2] Marek Chrobak, Mordecai J. Golin, Tak Wah Lam, and Dorian Nogneng. Scheduling with gaps: New models and algorithms. In *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, pages 114–126, 2015.

Problems of Scheduling with Imprecise Computation Revisited

Akiyoshi Shioura * Natalia V. Shakhlevich †
Vitaly A. Strusevich (Speaker) ‡

1 Introduction

The research area of *Scheduling with Controllable Processing Times (SCPT)* has been active for more than 35 years. What is unusual is that during all these years there has been a parallel stream of research, termed *Scheduling with Imprecise Computation* (*SIC*). In the range of models studied within the SIC research the processing machines are seen as processors, the jobs are computational tasks, and these tasks are allowed to be processed partially, thereby generating errors of computation. No close examination is needed to observe that the SIC models are versions or, more precisely, particular meaningful interpretations of the SCPT models. Both the SCPT and the SIC studies address essentially the same range of problems, and often apply the same methods.

The purpose of this paper is to clarify and improve the running times needed to solve the problems traditionally studied in the SIC research, so that in many occasions the best possible results can be achieved.

2 The Models

The jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on parallel machines $M_1, M_2, \ldots, M_m$, where $m \geq 2$. In the SCPT setting, the actual processing time $p(j)$ of job $j \in N$ is not given in advance but has to be chosen by a decision-maker from a given interval $[l(j), u(j)]$. Such a decision results in *compression* of the longest processing time $u(j)$ down to $p(j)$, and the value $x(j) = u(j) - p(j)$ is called the *compression amount* of job j. Compression may decrease the completion time of each job j but incurs additional cost.

Given m parallel machines, we distinguish between the *identical* machines and the *uniform* machines. In the latter case, it is assumed that machine M_i has speed s_i, $1 \leq i \leq m$. If for job j the value $u(j)$ is compressed to $p(j)$ and this job is assigned to machine M_i alone then the duration of such processing is $p(j)/s_i$.

*shioura.a.aa@m.titech.ac.jp Department of Industrial Engineering and Economics, Tokyo Institute of Technology, Tokyo 152-8550, 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.

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. In the processing of any job, *preemption* is allowed.

Let $C(j)$ denote the completion time of job $j \in N$, provided that its processing time is equal to $p(j)$. A schedule is called *feasible* if no job j is processed outside the time interval $[r(j), d(j)]$. To solve a problem with fixed processing times 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. We denote a generic feasibility problem with fixed processing times by $\alpha|r(j), C(j) \leq d(j), pmtn|-$. Here, in the first field either $\alpha = P$ or $\alpha = Q$, for identical parallel machines and for uniform machines, respectively.

Solving a typical problem from the SCPT range requires two decisions: (1) finding the compression amounts $x(j)$ for all jobs, and (2) determining a deadline-feasible preemptive schedule with actual processing times $p(j) = u(j) - x(j)$. The objective is to minimize a certain penalty function Φ that depends on compression amounts $x(j)$. For the range of problems traditionally considered in the SCPT literature, the most studied objective function represents the *total compression cost* and we denote it by $\Phi_\Sigma = \sum_{j \in N} w_T(j)x(j)$, where $w_T(j)$ is the unit cost, i.e., the cost of compressing job $j \in N$ by one unit of time, and given by a non-negative real number. Problems of minimizing the *maximum compression cost* are mainly studied within the SIC body of research; we denote such an objective function by $\Phi_{\max} = \max\{x(j)/w_M(j) | j \in N\}$, where for a given positive weight $w_M(j)$ the fraction $1/w_M(j)$ represents the unit cost. To refer to an SCPT problem, we use the generic notation $\alpha|r(j), p(j) = u(j) - x(j), C(j) \leq d(j), pmtn|\Phi$.

The SCPT problems can be interpreted in terms of SIC as follows. The jobs are seen as computational tasks to be processed with preemption in a computing system that consists of several parallel processors (machines). In computing systems that support imprecise computation, some computations (image processing programs, implementations of heuristic algorithms) can be run partially, producing less precise results. A task with processing requirement $u(j)$ can be split into a mandatory part which takes $l(j)$ time, and an optional part that may take up to $u(j) - l(j)$ additional time units. If instead of an ideal computation time $u(j)$ a task is executed for $p(j) = u(j) - x(j)$ time units, then computation is imprecise and $x(j)$ corresponds to the error of computation. In this settings, the objectives Φ_Σ and $\Phi_{\max}$ are understood as the total weighted error and the maximum weighted error, respectively. A popular research direction in SIC is related to the lexicographical optimization of the two criteria; see [2]. If the maximum weighted error $\Phi_{\max}$ should be minimized first and then further optimization is performed in the obtained class of solutions to minimize the total weighted error Φ_Σ, then the relevant problem is generically denoted by $\alpha|r(j), p(j) = u(j) - x(j), C(j) \leq d(j), pmtn|Lex(\Phi_{\max}, \Phi_\Sigma)$. In the counterpart with $Lex(\Phi_\Sigma, \Phi_{\max})$, the goal is to find a schedule that minimizes maximum weighted error among all schedules with the smallest total weighted error.

3 Main Results

Problems $P|r(j), C(j) \leq d(j), pmtn|-$ and $Q|r(j), C(j) \leq d(j), pmtn|-$ are known to be solvable in $O(n^3)$ time and in $O(mn^3)$ time, respectively by a reduction to finding the maximum flow in special bipartite networks. In the SIC literature, the following claim

is widely accepted: problem $P|r(j), C(j) \leq d(j), pmtn|-$ can be solved in $O\left(n^2 \log^2 n\right)$ time by using a modified network; see [2, 3]. We show that although such an approach will work in the case of a single machine, its extension to parallel machines is not possible. Thus, the estimates $O\left(n^3\right)$ and $O\left(mn^3\right)$ produce the lower bounds on the running times of possible algorithms for solving problems $P|r(j), p(j) = u(j) - x(j), C(j) \leq d(j), pmtn|\Phi$ and $Q|r(j), p(j) = u(j) - x(j), C(j) \leq d(j), pmtn|\Phi$, respectively.

Most of the results known in the SIC literature on minimizing the total error rely on traditional techniques of finding a max-flow or a min-cost max-flow, which gives the running times of $O\left(n^4 \log n\right)$ and $O\left(mn^4\right)$ for problems $P|r(j), p(j) = u(j) - x(j), C(j) \leq d(j), pmtn|\Phi_\Sigma$ and $Q|r(j), p(j) = u(j) - x(j), C(j) \leq d(j), pmtn|\Phi_\Sigma$; see [3]. We show that for these problems the lower bounds of $O\left(n^3\right)$ and $O\left(mn^3\right)$ can be achieved by the use of McCormick's method [4] for finding the max-flow in a network in which some arcs have capacities that linearly depend on an arc-related parameter.

In the case of the problems of minimizing the maximum error, we reduce the relevant problems to finding a lexicographically sharing max-flow in a network in which capacities of some arcs depend on a common parameter. Applying the techniques developed by Gallo et al. [1], we deduce that the best possible running times of $O\left(n^3\right)$ and $O\left(mn^3\right)$ can be achieved for problems $P|r(j), p(j) = u(j) - x(j), C(j) \leq d(j), pmtn|\Phi_{\max}$ and $Q|r(j), p(j) = u(j) - x(j), C(j) \leq d(j), pmtn|\Phi_{\max}$, respectively. Moreover, the same running times hold for the problems with the objectives $Lex(\Phi_{\max}, \Phi_\Sigma)$ and $Lex(\Phi_\Sigma, \Phi_{\max})$. This considerably improves the previously known running times, e.g., from $O\left(mn^5\right)$ to $O\left(mn^3\right)$ for problem $Q|r(j), p(j) = u(j) - x(j), C(j) \leq d(j), pmtn|Lex(\Phi_\Sigma, \Phi_{\max})$; see [2].

Furthermore, it is also possible to design algorithms with the same running times for the problems on identical and uniform parallel machines, respectively, to minimize a quadratic objective function $\Phi_{\text{quad}} = \sum w'(j) x(j)^2$, as well as for minimizing various lexicographic combinations of the functions Φ_Σ, $\Phi_{\max}$ and Φ_{quad}.

References

[1] G. Gallo, M.D. Grigoriadis, R.E. Tarjan A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.* 18 (1989), pp. 30–55.

[2] K.I.-J. Ho. Dual criteria optimization problems for imprecise computation tasks. In: J.Y.-T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models and Performance Analysis* , Chapman & Hall/CRC, 2004, pp. 35-1 – 35-26.

[3] J.Y.-T. Leung. Minimizing total weighted error for imprecise computation tasks. In: Leung, J.Y.-T. (ed.), *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Chapman & Hall/CRC, 2004, pp. 34-1 – 34-16.

[4] S.T. McCormick. Fast algorithms for parametric scheduling come from extensions to parametric maximum flow. *Oper. Res.* 47 (1999), pp. 744–756.

[5] G. Wan, J.Y.-T. Leung, M.L. Pinedo. Scheduling imprecise computation tasks on uniform processors. Inform. Process. Lett. 104 (2007), pp. 45–52.

A New Mixed Time Framework for the Periodically Aggregated Resource-Constrained Project Scheduling Problem

Pierre-Antoine Morin (Speaker) [1,2] Christian Artigues [2] Alain Haït [1,2]

1 Introduction

We consider a project scheduling problem in a context such that temporal aspects (start and completion of activities, precedence between activities) are handled precisely, while resource consumption is evaluated more roughly over aggregated periods. Hence, the Periodically Aggregated Resource-Constrained Project Scheduling Problem (PARCPSP) is well-suited for an intermediate planning/scheduling decision level, in particular with human resources or energetic resources. This strongly NP-hard problem, introduced in [1] and in-depth studied in [2], has been modelled by means of a Mixed Integer Linear Programming (MILP) formulation, based on a mixed time framework. In this abstract, we present an alternative way to manage both continuous and discrete temporal representations simultaneously, in order to derive a new MILP formulation for a purpose of theoretical and computational comparison.

2 Periodically Aggregated Resource-Constrained Project Scheduling Problem

The problem consists in finding a non-preemptive schedule that minimises the execution duration of a project (1) under precedence constraints (2) and periodically aggregated resource constraints (3). The project is defined by a set $\mathcal{A}$ of n activities and a set $\mathcal{R}$ of m renewable resources. At each instant of its execution, activity i (processing time p_i) requires a fix amount ($r_{i,k}$) on resource k (capacity b_k). Precedence relations are represented by an activity pair list (E). The time horizon is subdivided into periods of parameterized length Δ (positive real). An abstract formulation for the PARCPSP is:

$$\text{Minimise} \quad S_{n+1} - S_0 \tag{1}$$

$$\text{s.t.} \quad S_{i_2} - S_{i_1} \geq p_{i_1} \qquad \forall (i_1, i_2) \in E \tag{2}$$

$$\sum_{i \in \mathcal{A}} r_{i,k} \frac{d_{i,\ell}(S)}{\Delta} \leq b_k \qquad \forall k \in \mathcal{R} \quad \forall \ell \in \mathbb{Z} \tag{3}$$

pierre-antoine.morin@isae.fr, artigues@laas.fr, alain.hait@isae.fr.

[1]ISAE-SUPAERO, Université de Toulouse, Toulouse, France.

[2]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France.

Where: S_i is the start date of activity i, S_0 and S_{n+1} denote respectively the least start date and the greatest completion date, and $d_{i,\ell}(S)$ is the execution duration of activity i in period ℓ, i.e. the length of the intersection of two time intervals: the ℓ^{th} period (i.e. $[(\ell-1)\Delta, \ell\Delta]$) and the execution interval of activity i (i.e. $[S_i, S_i + p_i]$). Hence, the term $r_{i,k}\frac{d_{i,\ell}(S)}{\Delta}$ is the average demand of activity i on resource k in period ℓ.

$$d_{i,\ell}(S) = \max(0, \min(S_i + p_i, \ell\Delta) - \max(S_i, (\ell-1)\Delta))$$

3 A new mixed time framework

Two temporal representations coexist. On the one hand, start dates S_i permit to handle events and precedence exactly (continuous time representation). On the other hand, lengths $d_{i,\ell}(S)$ have to be computed in every period ℓ to evaluate the resource usage (discrete time representation). In terms of modelling, how to ensure a cohesion between these two temporal representations?

The initial MILP introduced in [1], adapted from [3], uses binary variables with an increasing step behaviour, and translate 6 out of 13 relations from Allen's algebra [4] (abstracts all possible relative positioning of two time intervals) into linear constraints. In this first model, constraints with a big-M term are required.

We design another approach to compute $d_{i,\ell}(S)$ values. This second model involves more (continuous) variables but less constraints, none of which have a big-M term.

Suppose the planning horizon contains L consecutive periods $\{1, \cdots, L\}$, so that the time interval covered is $[0, L\Delta]$. For every activity i, in every period ℓ (time interval $[(\ell-1)\Delta, \ell\Delta]$), two lengths are considered: the length $\lambda_{i,\ell}(S)$ of the intersection of the period and $[0, S_i]$, and the length $\mu_{i,\ell}(S)$ of the intersection of the period and $[S_i + p_i, L\Delta]$.

$$\lambda_{i,\ell}(S) = \max(0, \min(\Delta, S_i - (\ell-1)\Delta)) \qquad \mu_{i,\ell}(S) = \max(0, \min(\Delta, \ell\Delta - S_i - p_i))$$

The new mixed time framework relies on the fact that the three intervals whose length is measured by $\lambda_{i,\ell}(S)$, $d_{i,\ell}(S)$ and $\mu_{i,\ell}(S)$ form a partition of period ℓ .

$$\lambda_{i,\ell}(S) + d_{i,\ell}(S) + \mu_{i,\ell}(S) = \Delta$$

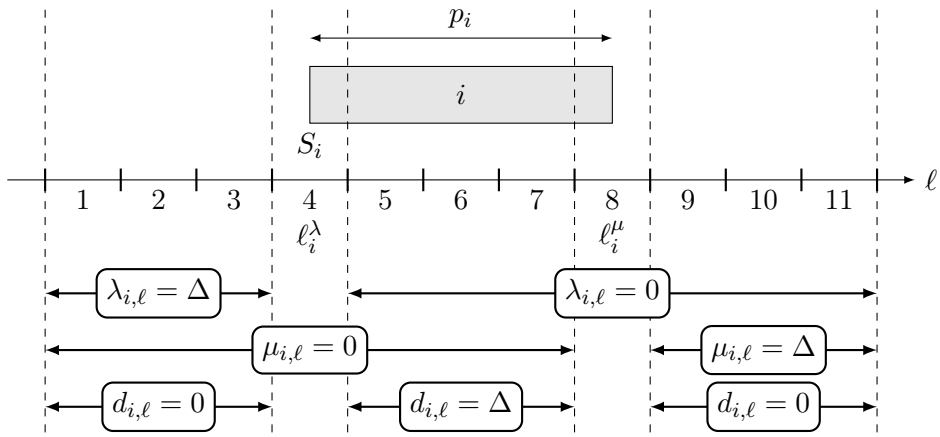


Figure 1: Principle of the new mixed time framework

Let $\ell_i^\lambda(S)$ (respectively $\ell_i^\mu(S)$) the index of the period that contains S_i (respectively $S_i + p_i$). Note that $\lambda_{i,\ell}(S)$ and $\mu_{i,\ell}(S)$ are equal to either 0 or Δ in almost every period ℓ, with a decreasing (respectively increasing) step behaviour (see Fig. 1).

$$\lambda_{i,\ell}(S) \begin{cases} = \Delta & \text{if } \ell < \ell_i^\lambda(S) \\ \in [0, \Delta] & \text{if } \ell = \ell_i^\lambda(S) \\ = 0 & \text{if } \ell > \ell_i^\lambda(S) \end{cases} \qquad \mu_{i,\ell}(S) \begin{cases} = 0 & \text{if } \ell < \ell_i^\mu(S) \\ \in [0, \Delta] & \text{if } \ell = \ell_i^\mu(S) \\ = \Delta & \text{if } \ell > \ell_i^\mu(S) \end{cases}$$

The values of $\ell_i^\lambda(S)$ and $\ell_i^\mu(S)$ are encoded by binary variables $z_{i,\ell}^\lambda(S)$ and $z_{i,\ell}^\mu(S)$ that follow the same step behaviour.

$$z_{i,\ell}^\lambda(S) \begin{cases} = 1 & \text{if } \ell \leq \ell_i^\lambda(S) \\ = 0 & \text{otherwise} \end{cases} \qquad z_{i,\ell}^\mu(S) \begin{cases} = 1 & \text{if } \ell \geq \ell_i^\mu(S) \\ = 0 & \text{otherwise} \end{cases}$$

Thus, linear lower and upper bounds on $\lambda_{i,\ell}(S)$ and $\mu_{i,\ell}(S)$ can be derived (note that a step behaviour of the binary variables is enforced by transitivity).

$$z_{i,\ell+1}^\lambda(S) \leq \frac{\lambda_{i,\ell}(S)}{\Delta} \leq z_{i,\ell}^\lambda(S) \qquad z_{i,\ell-1}^\mu(S) \leq \frac{\mu_{i,\ell}(S)}{\Delta} \leq z_{i,\ell}^\mu(S)$$

It remains to ensure that each activity i is processed during exactly p_i time units; by doing so, the values of $\lambda_{i,\ell_i^\lambda(S)}(S)$ and $\mu_{i,\ell_i^\mu(S)}(S)$ are implicitly balanced.

$$\sum_{\ell=1}^{L} d_{i,\ell}(S) = \sum_{\ell=1}^{L} (\Delta - \lambda_{i,\ell}(S) - \mu_{i,\ell}(S)) = p_i$$

Finally, S_i can indeed be inferred directly from either $\lambda_{i,\ell}(S)$ or $\mu_{i,\ell}(S)$.

$$S_i = \sum_{\ell=1}^{L} \lambda_{i,\ell}(S) \quad \Leftrightarrow \quad S_i + p_i = \sum_{\ell=1}^{L} \lambda_{i,\ell}(S) + \sum_{\ell=1}^{L} d_{i,\ell}(S) = L\Delta - \sum_{\ell=1}^{L} \mu_{i,\ell}(S)$$

References

[1] P.A. Morin, C. Artigues and A. Haït (2016). "A new relaxation of the Resource-Constrained Project Scheduling Problem", in *Proceedings of the 15th International Conference on Project Management and Scheduling (PMS2016)*, Valencia, Spain, 2016.

[2] P.A. Morin, C. Artigues, A. Haït, T. Kis and F. Spieksma (2017). "Structural Properties and Complexity of the Periodically Aggregated Resource-Constrained Project Scheduling Problem", Technical Report, LAAS-CNRS, 2017.

[3] A. Haït and G. Baydoun (2012). "A new event-based MILP model for the resource-constrained project scheduling problem with variable intensity activities (RCPSVP)", in *Proc. IEEE International Conference on Industrial Engineering and Engineering Management*, Hong Kong, 2012.

[4] J.F. Allen (1981). "An Interval-Based Representation of Temporal Knowledge", in *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI81)*, Vancouver, Canada, 1981.

A Variant of the Periodic Event Scheduling Problem and its Cycle Periodicity Formulation

Tobias Hofmann *

1 Introduction

The employment of industrial robot systems in the automotive industry noticeably changed the view of production plants and led to a tremendous increase in productivity. Nonetheless, rising technological complexity, the parallelization of production processes, as well as the crucial need for respecting specific safety issues pose new challenges for man and machine. Furthermore, the progress shall proceed – production cannot be too fast, too safe or too cheap.

This is the topic that me and my colleagues from Chemnitz University of Technology tackle within the ERDF research project viRAL (Validierte Inbetriebnahme von Roboteranlagen mit automatischer Logik- und Lageprüfung) joint with Voith Engineering Services GmbH and Fraunhofer IWU in Chemnitz.

Our goal is to develop algorithms, guidelines and tools that make the commissioning of industrial robot systems more dependable by verifying the programs of robots and logical controllers. This in particular includes optimizing the schedule of the robot systems in order to ensure desired period times as well as conflict free timetables already in the planning stage.

2 The Periodic Event Scheduling Problem and its Industrial Application

The talk will be about a periodic scheduling problem as it typically appears in the context of generating train timetables (see e. g. Liebchen and Möhring [2]). Because in both scenarios – train timetable design as well as robot system scheduling – one is confronted with temporal precedences, collision constraints and often also periodic processes. Therefore the mathematical models used for train timetabling scenarios are well applicable to the scheduling of robot systems.

The basic modelling framework considered in this talk will be the periodic event scheduling problem proposed by Serafini and Ukovich in 1989 [5], which can be adapted to the industrial environment, yielding a number of integer programming formulations. One of them is the following.

*tobias.hofmann@mathematik.tu-chemnitz.de. Professorship of Algorithmic and Discrete Mathematics, Chemnitz University of Technology, Reichenhainer Straße 39, 09126 Chemnitz, Germany.

Definition 1 *(Periodic Event Scheduling Problem).*
Given a directed graph $D = (V, A)$ *with* l, $u \in \mathbb{Q}^A$, *find an* $x \in \mathbb{Q}^V$ *as well as a* $p \in \mathbb{Z}^A$ *such that*

$$\forall\, a = (i, j) \in A: \quad l_a \leq x_j - x_i + T\, p_a \leq u_a.$$

Furthermore, the considered robot scheduling scenario gives rise to additional frequency constraints as well as the objective of minimizing the period time of the given robot systems. This leads to a variant of the periodic event scheduling problem involving more structural properties than the standard formulation.

3 The Cycle Periodicity Formulation for the Periodic Event Scheduling Problem

Besides its comprehensive modelling capabilities, one of the typical characteristics of the periodic event scheduling problem is its large number of different integer programming formulations. In our case its cycle periodicity formulation seems to be a good choice in order to reduce the number of integer offset variables $p \in \mathbb{Q}^A$, which are the hard part of solving periodic event scheduling problems (see e. g. Liebchen [1]). Whereby for a fixed integer offset vector $p \in \mathbb{Z}^A$, the periodic event scheduling problem can be solved in $\mathcal{O}(|V||A|)$ time, the periodic event scheduling problem in general is proved to be NP-complete (see Odijk [4]).

Furthermore, it is known that it suffices to require the cycle periodicity equations only for $\nu = |A| - |V| - 1$ fundamental cycles of a connected periodic event scheduling problem instance (see Nachtigall [3]). In order to get a well solvable cycle periodicity formulation, it plays a crucial role to choose a suitable cycle basis of the underlying precedence graph D.

Our actual research focuses on the latter aspect. We identified appropriate cycle basis as well as admissible bounds for the remaining integer offset variables.

References

[1] C. Liebchen (2006). *Periodic Timetable Optimization in Public Transport.* dissertation.de.

[2] C. Liebchen, R.H. Möhring (2007). *The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables and Beyond.* Algorithmic Methods for Railway Optimization, pages 3–40.

[3] (1994) K. Nachtigall. *A Branch-And-Cut Approach for Periodic Network Programming.* Hildesheimer Informatik Berichte 29.

[4] (1994) M.A. Odijk. *Construction of Periodic Timetables, Part 1: A Cutting Plane Algorithm.* Technical Report 94-61, TU Delft.

[5] (1989) P. Serafini, W. Ukovich. *A Mathematical Model for Periodic Scheduling Problems.* SIAM Journal on Discrete Mathematics, volume 2, number 4, pages 550–581.