



Proceedings of the 15th Workshop on Models and Algorithms  
for Planning and Scheduling Problems, MAPSP 2022

June 12-17, 2022, The Oropa Sanctuary, Biella, Italy.

### **Program committee**

Leah Epstein	University of Haifa, Israel (PC chair)
János Balogh	University of Szeged, Hungary
József Békési	University of Szeged, Hungary
Janka Chlebíková	University of Portsmouth, UK
Federico Della Croce	Politecnico di Torino, Italy
Franziska Eberle	London School of Economics, UK
Alexander Grigoriev	Maastricht University, The Netherlands
Péter Györgyi	SZTAKI, Hungary
Sven Jäger	Fraunhofer Institute for Industrial Mathematics, Germany
Lukasz Jeż	University of Wrocław, Poland
Shahin Kamali	University of Manitoba, Canada
Dusan Knop	Czech Technical University in Prague, Czech Republic
Marten Maack	Paderborn University, Germany
Chris Schwiegelshohn	Aarhus University, Denmark
Dvir Shabtay	Ben-Gurion University of the Negev, Israel
Tami Tamir	Reichman University, Israel

### **Organizing committee**

Federico Della Croce	Politecnico di Torino, Italy (chair)
Roberto Baretto	Politecnico di Torino, Italy
Gabriele Dragotto	Polytechnique Montréal, Canada
Alessandro Druetto	Università di Torino, Italy
Marco Ghirardi	Politecnico di Torino, Italy
Andrea Grosso	Università di Torino, Italy
Giuseppe Lancia	Università di Udine, Italy
Elena Renner	Politecnico di Torino, Italy
Fabio Salassa	Politecnico di Torino, Italy
Rosario Scatamacchia	Politecnico di Torino, Italy

## Keynote lectures

Scheduling DAGs: recent results and challenges <i>Alberto Marchetti Spaccamela</i>	9
Scheduling machines subject to unrecoverable failures and other related stochastic sequencing problems <i>Alessandro Agnetis</i>	11
Explorable Uncertainty and Untrusted Predictions <i>Thomas Erlebach</i>	12
Paging and Packing with Possibly Precise Predictions <i>Lene M. Favrholdt</i>	13

## Regular contributions

---

### Monday morning

---

Breaking the Barrier of 2 for the Competitiveness of Longest Queue Drop <i>Antonios Antoniadis, Matthias Englert, Nicolaos Matsakis, and Pavel Veselý (speaker)</i>	14
Randomized Item Collecting from Weight-Monotone Queues <i>Marek Chrobak, Lukasz Jeż (speaker), and Jiří Sgall</i>	18
Buffer Minimization with Conflicts on a Line <i>Felix Höhne and Rob van Stee (speaker)</i>	22
Barely Random Algorithms for Online Scheduling with Equal Job Sizes <i>Neda Abbasi Shahkooh (speaker), Martin Böhm, and Lukasz Jeż</i>	25
A parametric flow-based algorithm for crew diagramming: a pragmatic large-scale approach at the Swiss railways <i>Raphael Haemmerli, Alexander Souza (speaker), Denys Trieskunov, and Hsuan-Pin Wu</i>	29
Independent Sampling for the Secretary Problem <i>José Correa, Andrés Cristi, Laurent Feuilloley, Tim Oosterwijk (speaker), and Alexandros Tsigonias-Dimitriadis</i>	33
Collective schedules: how to find a consensus schedule? <i>Martin Durand (speaker) and Fanny Pascual</i>	37
Tight Competitive Analyses of Online Car-sharing Problems <i>Ya-Chun Liang (speaker), Kuan-Yun Lai, Ho-Lin Chen, Kazuo Iwama and Chung-Shou Liao</i>	41

---

**Monday afternoon**

---

SPT optimality via linear programming <i>Woo-Hyung Cho (speaker), David Shmoys, and Shane Henderson</i>	45
Polynomial-Size ILP formulations for the Total Completion Time Problem on a Parallel Batching Machine <i>Alessandro Druetto (speaker) and Andrea Grosso</i>	49
A Competitive Algorithm for Throughput Maximization on Identical Machines <i>Benjamin Moseley, Kirk Pruhs (speaker), Clifford Stein and Rudy Zhou</i>	53
Scheduling with Machine Conflicts <i>Moritz Buchem (speaker), Linda Kleist, and Daniel Schmidt genannt Waldschmidt</i>	57
Load Balancing: The Long Road from Theory to Practice <i>Sebastian Berndt, Max A. Deppert (speaker), Klaus Jansen, and Lars Rohwedder</i>	61
Rescheduling with new orders under a maximum completion time disruption constraint <i>Stefan Lendl, Ulrich Pferschy, Elena Renner (speaker), Fabio Salassa, and Vincent T'kindt</i>	65
An Exact Algorithm for the Linear Tape Scheduling Problem <i>Valentin Honoré, Bertrand Simon (speaker), and Frédéric Suter</i>	69
Additive Approximation Schemes for Load Balancing Problems <i>Moritz Buchem, Lars Rohwedder, Tjark Vredeveld (speaker), and Andreas Wiese</i>	73
A Primal-Dual and Primal-Greedy Approximation Framework for Weighted Covering Problems <i>Britta Peis, Niklas Rieken, José Verschae (speaker), Andreas Wierz</i>	77
On Flow Shop Scheduling with Job-Dependent Storage Requirements <i>Alexander Kononov (speaker) and Marina Pakulich</i>	81
Flow Time Scheduling and Prefix Beck-Fiala <i>Nikhil Bansal, Lars Rohwedder (speaker), and Ola Svensson</i>	85
An Inclusion-Exclusion based algorithm for permutation flowshop scheduling with job precedences <i>Olivier Ploton (speaker) and Vincent T'kindt</i>	88

---

**Tuesday morning**

---

Scheduling with Speed Predictions <i>Eric Balkanski, Tingting Ou, Clifford Stein, and Hao-Ting Wei (speaker)</i>	91
Non-Clairvoyant Scheduling with Predictions Revisited <i>Alexander Lindermayr (speaker) and Nicole Megow</i>	95

Contract Scheduling with Predictions <i>Spyros Angelopoulos (speaker) and Shahin Kamali</i>	99
The rectangle covering bound on the extension complexity of small cut polytopes <i>Wouter Fokkema (speaker) and Matthias Walter</i>	103
Randomized Cup Game Algorithms Against Strong Adversaries (SODA '21) <i>Michael A. Bender and William Kuszmaul (speaker)</i>	107
Multi-Purpose Machine Scheduling: An Application to Smart Cosmetic Manufacturing <i>Michele Barbato (speaker), Antonio Belotti, Alberto Ceselli, and Giovanni Righini</i>	111
Operating rooms scheduling with a shared resource: a red-blue knapsack modeling approach <i>Federico Della Croce, Andrea Grosso (speaker), and Vincent T'kindt</i>	115
Using Benders' cuts to backtrack in sports timetabling <i>David Van Bulck (speaker) and Dries Goossens</i>	119
Minimizing Tardiness in a Scheduling Environment with Jobs' Dominance Hierarchy <i>Michal Sinai (speaker) and Tami Tamir</i>	123
Replacement and Repair of Common Components in Systems Subject to Operations Planning <i>Gabrijela Obradović (speaker), Ann-Brith Strömberg, and Kristian Lundberg</i>	127
Parameterized Complexity of Single-machine Scheduling with Precedence, Release Dates and Deadlines <i>Claire Hanen, Maher Mallem (speaker), and Alix Munier Kordon</i>	131
Joint replenishment meets scheduling <i>Tamás Kis (speaker), Péter Györgyi, Tímea Tamási, József Békési</i>	135
The High-Dimensional Cow-Path Problem <i>Antonios Antoniadis, Ruben Hoeksma, Sándor Kisfaludi-Bak, and Kevin Schewior (speaker)</i>	139
Probabilistic Real-Time Scheduling <i>Georg von der Brüggen (speaker), Sergey Bozhko, Kuan-Hsun Chen, Jian-Jia Chen, and Björn B. Brandenburg</i>	143
Restricted Adaptivity in Stochastic Scheduling <i>Guillaume Sagnol and Daniel Schmidt genannt Waldschmidt (speaker)</i>	146
Data transfer scheduling for deep space exploration <i>Emmanuel Hebrard (speaker), Christian Artigues, Pierre Lopez, Arnaud Lusson, Steve Chien, Adrien Maillard, and Gregg Rabideau</i>	149

---

**Tuesday afternoon**

---

Better Algorithms for Online Bin Stretching via Computer Search <i>Matej Lieskovský</i>	153
A Multi-Phase Algorithm for Bin Stretching with Stretching Factor below 1.5 <i>Martin Böhm, Matej Lieskovský, Sören Schmitt (speaker), Jiří Sgall, and Rob van Stee</i>	156
Resource Optimization for Program Committee Members: A Subreview Article <i>Michael A. Bender, Samuel McCauley, Bertrand Simon, Shikha Singh (speaker), and Frédéric Vivien</i>	160
Orthogonal schedules in Round Robin competitions <i>Roel Lambers (speaker), Jop Briët, Viresh Patel, Frits Spijksma, and Mehmet Akif Yıldız</i>	164
Faster Matchings via Learned Duals <i>Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley (speaker), and Sergei Vassilvitskii</i>	167
The Power of Amortized Recourse on Online Graph Problems <i>Hsiang-Hsuan Liu and Jonathan Toole-Charignon (speaker)</i>	171

---

**Wednesday morning**

---

On Hop-Constrained Steiner Trees in Tree-Like Metrics <i>Martin Böhm, Ruben Hoeksma (speaker), Nicole Megow, Lukas Nölke, and Bertrand Simon</i>	175
On the Extended TSP Problem <i>Julián Mestre (speaker), Sergey Pupyrev, and Seun William Umboh</i>	178
Arc-retraversing Solutions to the Traveling Salesman Problem with Drones <i>Nicola Morandi (speaker), Roel Leus, Hande Yaman</i>	181
Solving a real-life assembly problem with lobster precedence and workforce constraint <i>Mauro Dell’Amico and Dario Bezzi (speaker)</i>	184
A 12/7-approximation algorithm for the discrete Bamboo Garden Trimming problem <i>Martijn van Ee</i>	188
On the bamboo garden trimming problem <i>Felix Höhne (speaker) and Rob van Stee</i>	192
Bamboo Trimming Revisited: Simple Algorithms Can Do Well Too <i>John Kuszmaul</i>	195

An approximation algorithm for sequencing unreliable jobs on parallel machines with job duplication <i>Ben Hermans (speaker), Alessandro Agnetis, Mario Benini, Paolo Detti, and Marco Pranzo</i>	199
Complexity of partitioned scheduling for periodic tasks <i>Pontus Ekberg (speaker) and Sanjoy Baruah</i>	202
Periodic scheduling with shared resources <i>Vít Koštejn (speaker) and Jiří Sgall</i>	206
Exact Polynomial Time Algorithm for the Response Time Analysis of Harmonic Tasks <i>Thi Huyen Chau Nguyen, Werner Grass, and Klaus Jansen (speaker)</i>	210
The Periodic Lock Scheduling Problem <i>Julian Golak (speaker), Alexander Grigoriev, Freija van Lent, and Tom van der Zanden</i>	214
Makespan minimization in tree data gathering networks with memory limits <i>Joanna Berlińska</i>	218
Static schedules for fault-tolerant transmission <i>Kunal Agrawal (speaker), Sanjoy Baruah, Alan Burns, Jeremy Fineman, and Zhe Wang</i>	222
Scheduling Messages towards Detecting Monochromatic Pattern on a Bounded Degree Network <i>Bala Kalyanasundaram (speaker) and Mahe Velauthapillai</i>	225
Data-driven scheduling in serverless computing to reduce response time <i>Bartłomiej Przybylski, Paweł Żuk (speaker), and Krzysztof Rządca</i>	229
<hr/> <b>Thursday morning</b> <hr/>	
Automatic HBM Management: Models and Algorithms <i>Daniel DeLayo (speaker), Kenny Zhang, Kunal Agrawal, Michael A. Bender, Jonathan W. Berry, Rathish Das, Ben Moseley, and Cynthia A. Phillips</i>	233
Paging and the Address-Translation Problem <i>Michael A. Bender, Abhishek Bhattarcharjee, Alex Conway, Martín Farach-Colton (speaker), Rob Johnson, Sudarsun Kannan, William Kuszmaul, Nirjhar Mukherjee, Don Porter, Guido Tagliavini, Janet Vorobyeva, and Evan West</i>	237
Tight Bounds for Parallel Paging and Green Paging <i>Kunal Agrawal, Michael A. Bender (speaker), Rathish Das, William Kuszmaul, Enoch Peserico, and Michele Squizzato</i>	241
Relations between Periodic Scheduling with Harmonic Periods and 2D Packing <i>Claire Hanen (speaker) and Zdeněk Hanzálek</i>	245

Filling a Theater During the COVID-19 Pandemic <i>Danny Blom, Rudi Pendavingh, and Frits Spieksma (speaker)</i>	249
Queuing safely for elevator systems amidst a pandemic <i>Sai Mali Ananthanarayanan, Charles Branas, Adam N. Elmachtoub, Clifford Stein (speaker), and Yeqing Zhou</i>	252
Learning-Augmented Dynamic Power Management with Multiple States via New Ski Rental Bounds <i>Antonios Antoniadis (speaker), Christian Coester, Marek Eliáš, Adam Polak, and Bertrand Simon</i>	256
Speed-Robust Scheduling: Sand, Bricks, and Rocks <i>Franziska Eberle (speaker), Ruben Hoeksma, Nicole Megow, Lukas Nölke, Kevin Schewior, and Bertrand Simon</i>	259
Balancing Flow Time and Energy Consumption <i>Sami Davies (speaker), Samir Khuller, and Shirley Zhang</i>	263
Speed Scaling with Explorable Uncertainty <i>Evrpidis Bampis, Konstantinos Dogeas, Alexander Kononov, Giorgio Lucarelli (speaker), and Fanny Pascual</i>	266
Allocation and pricing of planetarium seats <i>Freija van Lent (speaker), Julian Golak, and Alexander Grigoriev</i>	270
Towards hardness of approximate pure equilibria <i>Vipin Ravindran Vijayalakshmi and Alexander Skopalik (speaker)</i>	274
Scheduling Games With Rank-Based Utilities <i>Shaul Rosner (speaker) and Tami Tamir</i>	278
Scheduling Games with Machine-Dependent Priority Lists <i>Vipin Ravindran Vijayalakshmi, Marc Schroder (speaker), and Tami Tamir</i>	282
<hr/> <b>Thursday afternoon</b> <hr/>	
Scheduling Parallel Jobs in Two-processor Systems with Energy Constraint <i>Yulia Zakharova (speaker) and Alexander Kononov</i>	285
The parallel AGV Scheduling Problem with battery constraints: exact and heuristic approaches <i>Adriano Masone (speaker), Maurizio Boccia, Andrea Mancuso, Claudio Sterle</i>	289
A hybrid local search algorithm for the Continuous Energy-Constrained Scheduling Problem <i>Roel Brouwer (speaker), Marjan van den Akker, and Han Hoogeveen</i>	293
Graph burning and non-uniform $k$ -centers for small treewidth <i>Matej Lieskovský and Jiří Sgall (speaker)</i>	297

Methods for Planning the Search of Infected Nodes in Uncertain Graphs  
*Benjamín Rubio Orellana (speaker), José Baboun, Isabelle Beaudry, Mauricio Castro, Alejandro Jara, José Verschae* 301

Robustification of Online Graph Exploration Methods  
*Franziska Eberle, Alexander Lindermayr, Nicole Megow, Lukas Nölke, and Jens Schlöter (speaker)* 305

# Scheduling DAGs: recent results and challenges

Alberto Marchetti Spaccamela\* (Keynote Lecture, Monday Morning)

---

Multi-core architectures are nowadays widely used for their increased performance over single-core processors. To take full advantage of these architectures we must exploit intra-task parallelism.

The Directed Acyclic Graph (DAGs) is a popular representation to describe the structure of parallel applications and to model the execution of multi-threaded programs that is widely used in cloud computing and in real-time systems.

In this talk I will present recent results on DAG scheduling considering different models and focusing on complexity, approximation and integer linear program representations.

---

\*[alberto.marchetti-spaccamela@uniroma1.it](mailto:alberto.marchetti-spaccamela@uniroma1.it). Dipartimento di Ingegneria informatica automatica e gestionale, Università di Roma La Sapienza, Italy.

# Scheduling machines subject to unrecoverable failures and other related stochastic sequencing problems

Alessandro Agnetis\* (Keynote Lecture, Tuesday afternoon)

---

Consider the following basic setting. A set of  $n$  jobs has to be performed on a set of parallel, identical machines. Unlike most scheduling situations, machines may actually fail, i.e., while performing a job  $i$ , they can become unavailable (e.g. break down) with probability  $\pi_i$ . Such failures are unrecoverable, in the sense that from then onwards the machine is lost and so are the jobs not yet processed on that machine. If a job  $i$  is successfully completed, a reward  $r_i$  is attained. The problem is how to assign the jobs to the machines and sequence them so that the expected reward is maximized. In this talk we review the main results, discuss relationships with other sequencing problems and point out some open problems.

While the single-machine case is easy, for two or more machines the problem is hard and various approaches have been proposed to address it. For general  $m$ , list scheduling yields a 0.8531-approximate solution. The argument is similar to the one used by Schwegelshohn to prove Kawaguchi and Kyan's bound for the minimization of total weighted completion time. A variant of this problem is attained if, in order to hedge against failures, we can use job replication. In this case, copies of the same job can be scheduled on different machines, and the reward  $r_i$  is attained if at least one copy is successfully completed. In this case, the sequence of the  $n$  copies of the jobs on each machine has to be decided. Although also this problem is hard for  $m \geq 2$ , relatively simple algorithms provide solutions which are provably close to optimality.

A related sequencing problem is the following. A system consists of  $n$  components, each of which can be either functioning or not. Only if all components are functioning, the system is "up". Component  $i$  is functioning with probability  $\pi_i$ , and testing it costs  $c_i$ . As soon as a component that is not functioning is detected, the testing stops (concluding that the system is "down"). The problem is to decide in which order should the components be tested, in order to minimize the expected costs. While the single-tester problem is solved by a simple priority rule, various problem variants can be considered. In particular, if several testers operate in parallel, under time constraints, the problem gets more complicated. While it is NP-hard for three or more testers, its complexity with two testers is still open.

---

\*agnetis@diism.unisi.it. Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, Italy.

# Explorable Uncertainty and Untrusted Predictions

Thomas Erlebach\* (Keynote Lecture, Thursday afternoon)

---

The research area of explorable uncertainty is concerned with problems where some of the input data is uncertain, but queries can be executed to reveal the true values of uncertain input elements. Uncertain values are typically given in the form of intervals. The goal is to minimise the number of queries that are needed until a provably correct solution can be output. In addition to the intervals, we may have access to predictions of the true values, possibly obtained via machine learning. Our aim is then to devise query strategies that benefit from accurate predictions but do not get misled too much by incorrect predictions. In this talk, we will discuss some recent progress in this combined setting, including query strategies for minimum spanning trees with edge uncertainty that benefit from good predictions while maintaining the best possible worst-case competitive ratio even if the predictions are arbitrarily wrong.

(The talk is mainly based on joint work with Murilo Santos de Lima, Nicole Megow and Jens Schlöter.)

---

\*[thomas.erlebach@durham.ac.uk](mailto:thomas.erlebach@durham.ac.uk). Department of Computer Science, University of Durham, United Kingdom.

# Paging and Packing with Possibly Precise Predictions

Lene Monrad Favrholt\* (Keynote Lecture, Friday morning)

---

The talk will survey techniques for designing and analyzing online algorithms with predictions, advice as well as possibly imprecise predictions.

Online problems with predictions relate to semi-online problems in that some information is available which would not be available to a purely online algorithm. In the advice as well as the semi-online setting, this additional information is usually exact. In the semi-online setting, some emphasis is put on the information being realistically obtainable, whereas in the advice setting, the focus is on minimizing the amount of information communicated to the algorithm. In online problems with predictions, the concern is how well the algorithm adapts to errors in the information and the information should preferably be learnable.

The running example will be the paging problem, but we will also consider packing problems such as knapsack and bin packing.

---

\*[lenem@imada.sdu.dk](mailto:lenem@imada.sdu.dk). Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark.

# Breaking the Barrier of 2 for the Competitiveness of Longest Queue Drop

Antonios Antoniadis <sup>\*</sup>    Matthias Englert <sup>†</sup>    Nicolaos Matsakis <sup>‡</sup>  
Pavel Veselý (Speaker) <sup>§</sup>

---

We study a fundamental model of buffer management in shared-memory network switches. A shared-memory switch consists of a buffer of size  $M \in \mathbb{N}$ , an input port, and  $N \in \mathbb{N}$  output ports. We consider a slotted time model. In each time step, an arbitrary number of unit-valued packets arrive to the input port. Each packet comes with a label specifying the output port that it has to be forwarded to. A buffer management algorithm has to make a decision for each packet: either irrevocably reject it, or accept it while ensuring that the buffer capacity  $M$  is respected, which may mean that a previously accepted packet has to be evicted. At the end of the time step, each output port with at least one packet in the buffer destined to it transmits a packet. The goal of the buffer management algorithm is to accept/reject incoming packets or evict already accepted packets, so as to maximize the throughput, i.e., the total number of transmitted packets, while ensuring that at most  $M$  packets in total are stored for all output ports at any time. (We remark that eviction is also referred to as preemption, and is necessary to achieve a constant competitive ratio [5].)

Given the inherently online nature of buffer management problems, a standard approach is to design online algorithms for them and evaluate the algorithm's performance using its competitive ratio. More specifically, an online algorithm  $\text{ALG}$  is  $c$ -competitive (where  $c \geq 1$ ) if the number of packets transmitted by an optimal offline algorithm  $\text{OPT}$  (that has full knowledge of the incoming packet sequence a priori) is at most  $c$  times the number of packets transmitted by  $\text{ALG}$ . There exists an extensive body of research dedicated to designing competitive online algorithms with the aim of improving the performance of networking devices that incorporate buffers (see e.g. [3, 8]).

Since packets have unit value, we can assume without loss of generality that the packets destined to a specific output port are transmitted in an earliest-arrival (FIFO) fashion and thus, it is helpful to associate each output port with a queue. Intuitively speaking, to maximize throughput, one would like to maintain a flow

---

<sup>\*</sup>a.antoniadis@utwente.nl. University of Twente, The Netherlands.

<sup>†</sup>M.Englert@warwick.ac.uk. University of Warwick, UK.

<sup>‡</sup>nickmatsakis@gmail.com. Athens, Greece.

<sup>§</sup>vesely@iuuk.mff.cuni.cz. Computer Science Institute of Charles University, Czech Republic.

of packet transmissions for as many queues in parallel as possible. It is therefore desirable to prioritize accepting packets for queues that do not have many incoming packets in the near future. Unfortunately, an online algorithm does not know which queues these are, and in order to be insured against an adversarial input it seems reasonable to try to keep the queue lengths as balanced as possible in every step. This is exactly the idea behind the online algorithm *Longest Queue Drop (LQD)*, introduced in 1991 by Wei, Coyle, and Hsiao [9]: The incoming packet is always accepted and if this causes the buffer to exceed its capacity then one packet from the longest queue, breaking ties arbitrarily, is evicted (this could be the incoming packet).

**Previous results.** Hahne, Kesselman, and Mansour [4] provided the first formal analysis of LQD, showing that it is 2-competitive (see also Aiello, Kesselman, and Mansour [1]). The proof follows from a simple procedure that charges the extra profit of OPT to the profit of LQD. Furthermore, they demonstrated that LQD is at least  $\sqrt{2}$ -competitive, and also showed a general lower bound of  $4/3$  for the competitive ratio of any deterministic online algorithm.

The analysis of LQD in [1, 4] was then refined by Kobayashi, Miyazaki, and Okabe [6] who showed that the LQD competitive ratio is at most  $2 - \min_{k=1, \dots, N} (\lfloor M/k \rfloor + k - 1)/M$ . However, for  $N > \sqrt{M}$ , this bound becomes  $2 - O(1/\sqrt{M})$  and therefore does not establish a  $2 - \varepsilon$  upper bound for a constant  $\varepsilon > 0$  in general. Additionally, for the case of  $N = 2$  output ports, Kobayashi et al. [6] proved that LQD is  $4/3$ -competitive, and for  $N = 3$ , Matsakis shows that LQD is 1.5-competitive [7].

More recently, Bochkov, Davydov, Gaevoy, and Nikolenko [2] improved the lower bound on the competitiveness of LQD from  $\sqrt{2}$  to approximately 1.44 (using a direct simulation of LQD and also independently, by solving a linear program). Moreover, they show that any deterministic online algorithm is at least  $\sqrt{2}$ -competitive, using a construction inspired by the LQD specific lower bound from [1, 4]. To the best of our knowledge, so far, no randomized algorithms for this problem have been studied.

**Our contribution.** Although LQD is a very natural online algorithm to derive (and the best known for buffer management in shared-memory switches), determining its true competitiveness remains an elusive problem and has been described as a significant open problem in buffer management [3, 8]. After the initial analysis which showed that LQD is 2-competitive and not better than  $\sqrt{2}$ -competitive [1, 4] progress on the upper bound has been limited to special cases. In this paper, we make the first major progress in almost twenty years on upper bounding the competitive ratio of LQD. Namely, we prove the first  $(2 - \varepsilon)$  upper bound for a constant  $\varepsilon > 0$  without restrictions on the number of ports or the size of the buffer:

**Theorem 1.** *LQD is 1.707-competitive.*

This result appears in the proceedings of ICALP 2021 and in arXiv:2012.03906.

We remark that Theorem 1 applies to LQD with any tie-breaking rule, even if tie-breaking is under control of the adversary, and that our upper bound is strictly smaller than  $1 + 1/\sqrt{2}$ .

**Our techniques.** The proof of 2-competitiveness of LQD in [1, 4] uses the following general approach. If an optimal offline algorithm OPT currently stores more packets for a queue than LQD does, these excess packets present potential extra profit for OPT. Each such potential extra packet  $p$  in OPT is then matched to a packet that is transmitted by LQD at some point before packet  $p$  can be transmitted by OPT.

Our approach is different in that we (for the most part) do not match specific packets to one another. Instead, the idea is to take the total profit of LQD in each step and distribute it evenly among all potential extra packets that exist at the time. As such, the scheme is less discrete than the previous one. We then carefully calculate that, for each queue, *on average* each potential extra packet in that queue receives a profit strictly larger than one, and this implies the  $(2 - \varepsilon)$ -competitiveness of LQD.

As described here, this approach does not quite work yet. Two additional types of charging concepts have to be combined with this first idea: One involves not splitting the LQD profit completely evenly and instead slightly favoring queues with relatively few potential extra packets, and the other involves matching some of the potential extra packets of OPT to extra packets that LQD transmits. Another difficulty is that the lengths of two queues from which packets are rejected or evicted in the same time step may differ by one packet. This makes our proof more intricate. To deal with this, we introduce a potential function that will amortize the LQD profit in a suitable way. Then, the main challenge is to obtain useful lower bounds on the profit assigned to each queue, for which we introduce a novel scheme that relates the buffers of LQD and of OPT.

We remark that in a step when OPT transmits extra packets from some queues, the current LQD profit may be very small compared to that of OPT and thus, it is in general not possible to show that in each step LQD transmits a constant fraction of packets sent by OPT in the same step.

## References

- [1] W. Aiello, A. Kesselman, and Y. Mansour. Competitive buffer management for shared-memory switches. *ACM Transactions on Algorithms*, 5(1):3:1–3:16, 2008.
- [2] I. Bochkov, A. Davydow, N. Gaevoy, and S. I. Nikolenko. New competitiveness bounds for the shared memory switch. *CoRR*, abs/1907.04399, 2019.
- [3] M. H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010.

- [4] E. L. Hahne, A. Kesselman, and Y. Mansour. Competitive buffer management for shared-memory switches. In *Proceedings of the 13th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 53–58, 2001.
- [5] A. Kesselman and Y. Mansour. Harmonic buffer management policy for shared memory switches. *Theoretical Computer Science*, 324(2-3):161–182, 2004.
- [6] K. M. Kobayashi, S. Miyazaki, and Y. Okabe. A tight bound on online buffer management for two-port shared-memory switches. In *Proceedings of the 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 358–364, 2007.
- [7] N. Matsakis. *Approximation Algorithms for Packing and Buffering problems*. PhD thesis, University of Warwick, UK, 2015.
- [8] S. I. Nikolenko and K. Kogan. Single and multiple buffer processing. In *Encyclopedia of Algorithms*, pages 1988–1994. Springer, 2016.
- [9] S. X. Wei, E. J. Coyle, and M. T. Hsiao. An optimal buffer management policy for high-performance packet switching. In *Proceedings of the Global Communication Conference (GLOBECOM)*, pages 924–928, 1991.

# Randomized Item Collecting from Weight-Monotone Queues

Marek Chrobak\*

Łukasz Jeż (Speaker)<sup>†</sup>

Jiří Sgall<sup>‡</sup>

---

## 1 Introduction

We consider a scheduling problem, where a collection of *items* (*packets*) is stored in a queue, item  $p$  having weight  $w_p \geq 0$ . The queue is subject to updates: insertions and expirations. At any step, first, some prefix of the queue can expire, i.e., items in it are removed from the queue. Then new items can be inserted into the queue in arbitrary locations. Our goal is to collect a maximum-weight subset of items from the queue.

We focus on randomized online algorithms for this problem, whose (random) decision whether or not to collect an item do not depend on future queue updates. We express the performance of an online algorithm  $\mathcal{A}$  by its *competitive ratio*  $R = \max_I \mathcal{A}(I)/\text{opt}(I)$ , where  $\mathcal{A}(I)$  and  $\text{opt}(I)$  denote the expected profit of  $\mathcal{A}$  and the optimum (offline) profit on an instance  $I$ . (We assume the *oblivious adversary* model, where the queue updates do not depend on the outcomes of earlier random choices of  $\mathcal{A}$ .)

This problem was studied [1, 2]) as a generalization of the standard “packet scheduling” problem (see [5] and references therein), in which an algorithm knows the packet’s expiration time as soon as it arrives; by contrast, in our problem it only knows the relative order of packet expirations, i.e., the queue order, which we denote  $\triangleleft$ .

Formally, it is better to assume that items remain in the queue after arrival but those expired or collected cannot be collected. Then  $\triangleleft$  is a linear order on all items, and an arriving item  $a$  must satisfy  $e \triangleleft a$  for every expired item  $e$ , lest  $a$  is expired at arrival.

We consider a restricted class of *weight-monotone* instances, in which  $a \triangleleft b$  implies  $w_a \leq w_b$ . Our main result is a 1.5-competitive randomized algorithm, improving upon the bound of  $e/(e-1) \approx 1.58$  [3] that holds for general instances. We also give a lower bound of  $\approx 1.255$  for randomized algorithms, establishing the best bound without further assumptions and improving upon the 1.25 bound for packet scheduling [4]; the next section explains its significance.

---

\*Department of Computer Science, University of California at Riverside, CA, U.S.A.

<sup>†</sup>Institute of Computer Science, University of Wrocław, Poland

<sup>‡</sup>Computer Science Institute of Charles University, Prague, Czech Republic

## 2 Intuitions and lower bounds

We illustrate the problem by recovering lower bounds for packet scheduling. In our constructions there are only a few items, all arriving at once, but expiring at unknown times. Note that with expirations known, an optimum solution to an instance with no further arrivals can be easily found at the beginning; indeed, the original proofs [4] involve potentially long sequences of choices, with packets arriving over time.

We identify items with their weights. Consider two items,  $\alpha < 1$ , and two scenarios: after one step either both items expire, or only item  $\alpha$ , yielding optimum gains 1 and  $1 + \alpha$  respectively. If an algorithm deterministically chooses to collect  $\alpha$ , the first scenario may happen, and when it chooses 1, the other. Thus, the competitive ratio is at least  $\min\{1/\alpha, 1 + \alpha\}$ , which attains the maximum of  $\phi = 1 + \frac{1}{\phi} \approx 1.618$  for  $\alpha = \phi - 1 = 1/\phi$ . Now consider the same for a randomized algorithm, which chooses  $\alpha$  with probability  $p$ . It may collect both items if first it collects  $\alpha$  and only this item expires. Specifically, its expected gain when both items expire after one step is  $p\alpha + (1 - p)$ , and when only  $\alpha$  expires it is  $1 + p\alpha$ . The optimum gains are unchanged, and the lower bound is  $\max_{\alpha} \min_p \max\{1/(p\alpha + (1 - p)), (1 + \alpha)/(1 + p\alpha)\}$ , since the algorithm can choose  $p$  given  $\alpha$ . For best choice of  $p$ ,  $\alpha = 0.5$  yields the maximum ratio of 1.25.

The new lower bound is of similar nature: The queue contains three items  $\alpha < \beta < 1$ , and we consider three expiration scenarios: After one step either all three items expire, or the first two, or just the first one with the second one expiring after another step. Similarly as before, any algorithm can be captured by its probability distribution for the item it chooses in the first step, and another one for the second step if there is a choice to make (i.e., the algorithm collected  $\alpha$  and no other item expired). Letting  $\alpha = 0.4$  and  $\beta = 0.5$ , the calculations yield a lower bound of  $\approx 1.255$ . What makes it interesting is the fact that for deterministic algorithms no such improvement is possible, as there is a  $\phi$ -competitive algorithm for weight-monotone instances [2].

## 3 Barely Random Algorithm

Our 1.5-competitive algorithm simulates a deterministic algorithm chosen upfront: Greedy with probability 1/3 and MarkAndPick<sub>0.5</sub> (or MaP<sub>0.5</sub> for short) with probability 2/3. Before explaining MaP, we note that our proof essentially consists of a charging scheme in which each item from the optimum solution gives rise to either a single charge of its weight or two charges, each of exactly half its weight. Each charge is assigned to an item collected by either Greedy or MaP no lighter than the charge weight in such a way that each item collected by either algorithm receives at most one charge.

MaP <sub>$\lambda$</sub>  algorithm (with implicit  $\lambda = \phi - 1$ ; we will use  $\lambda = \frac{1}{2}$  later), introduced as the optimal algorithm for weight-monotone instances [2], works as follows: While

there is a pending item, it marks the heaviest unmarked item  $m$  (which may already be expired or collected) and collects the earliest pending item  $\ell$  such that  $\ell \geq \lambda m$ .

That  $\text{MaP}_\lambda$  is  $\max\{1 + \lambda, 1/\lambda\}$ -competitive follows from several structural observations, expressed as *dominance* relations for various sets of items [2]:  $V$  *dominates*  $U$ , denoted  $V \succeq U$ , if and only if there is an injection  $f : U \mapsto V$  such that  $f(u) \geq u$  for all  $u \in U$ ; we also allow multipliers, e.g.,  $a \cdot V \succeq b \cdot U$ . Those relations, together with some new ones that we introduce, provide the actual rules for the charging scheme. The novelty of our result thus lies in observing and proving that  $\text{MaP}$  and  $\text{Greedy}$  complement each other: when one of them performs poorly, the other performs significantly better. The technical contribution lies in appropriately extending said structural properties.

### 3.1 Known Properties of MarkAndPick

Basic dominance relations and the fact that  $\text{MaP}$  is well defined, i.e., that items  $m$  and  $\ell$  do exist are known. Specifically, let  $M_t$ ,  $L_t$  and  $Z_t$  denote, respectively, the sets of items marked by  $\text{MaP}$ , collected by  $\text{MaP}$ , and collected by optimum solution up to and including step  $t$ . Then  $M_t \succeq L_t$ ,  $L_t \succeq \lambda \cdot M_t$ , and  $|M_t| = |L_t|$ . It is also known that to analyze  $\text{MaP}$  (or our algorithm), it suffices to consider a “single phase” in which  $\text{MaP}$  (though not  $\text{Greedy}$ ) collects items in each step up to a point when it no longer has pending items. At that point there are no more updates until the phase ends, which happens when the optimum solution, which also collected some item in each step thus far, collects all the items that are not yet expired. Suppose that the optimum solution collects  $T$  items in the phase but  $\text{MaP}$  collects only  $T - k$  items for some  $k \geq 0$ . Denote these  $k$  last items collected by the optimum by  $K$ . Clearly,  $K$  is the set of  $k$  heaviest items overall, hence marked and collected by  $\text{MaP}$ . Moreover,  $Z_T$  can be partitioned into  $Z_T^+$  of size  $k$  and  $Z_T^-$  such that  $\lambda \cdot K \succeq Z_T^+$  and  $M_{T-k} \succeq Z_T^-$ . These summarize final proof, which can be seen as a charging scheme:  $w(Z_T) = w(Z_T^+) + w(Z_T^-) \leq \lambda \cdot w(K) + w(M_{T-k}) = \lambda \cdot w(K) + w(K) + w(M_{T-k} \setminus K) \leq (1 + \lambda)w(K) + \frac{1}{\lambda}w(L_{T-k} \setminus K) \leq \max\left\{1 + \lambda, \frac{1}{\lambda}\right\} \cdot w(L_{T-k})$ .

### 3.2 New Properties of MarkAndPick and the Charging Scheme

The basic idea of our proof (for  $\lambda = 0.5$ ) is to split the charge from  $w(Z_T^-)$  using  $w(Z_T^-) \leq w(M_{T-k}) \leq \frac{1}{2}w(M_{T-k}) + w(L_{T-k})$ , i.e., charge those items to both those collected by  $\text{Greedy}$  and those collected by  $\text{MaP}$ . The issue with that is that not all marked items are actually collected by  $\text{Greedy}$ . The second idea is to leverage the fact that when an expired item  $m$  is marked, the item  $\ell$  collected by  $\text{MaP}$  satisfies a stronger relation  $\ell \geq m$ . Hence, if  $M^I$  and  $L^I$  denote the items marked and collected when  $\text{Greedy}$  idles, we have  $M^I \succeq L^I$ , which could be used for different charging through  $M^I$ , using all capacity of  $L^I$ . The final obstacle is that the heaviest  $k$  items in  $L_{T-k}$ ,  $K$ , already receive a different charge, so if  $L^I$  and  $K$  share some elements, those will be overcharged. This may indeed happen, but

we demonstrate a construction of a different set  $L^I$  which is disjoint with  $K$  yet satisfies the same desired relations.

## References

- [1] M. Bienkowski, M. Chrobak, C. Dürr, M. Hurand, A. Jež, Ł. Jež, and G. Stachowiak. Collecting weighted items from a dynamic queue. *Algorithmica*, 65(1):60–94, 2013.
- [2] M. Bienkowski, M. Chrobak, C. Dürr, M. Hurand, A. Jež, Ł. Jež, and G. Stachowiak. A  $\phi$ -competitive algorithm for collecting items with increasing weights from a dynamic queue. *Theoretical Computer Science*, 475:92 – 102, 2013.
- [3] F.Y.L. Chin, M. Chrobak, S.P.Y. Fung, Wojciech Jawor, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *J. of Discrete Algorithms*, 4(2):255–276, 2006.
- [4] F.Y.L. Chin and S.P.Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.
- [5] P. Veselý, M. Chrobak, Ł. Jež, and J. Sgall. A  $\phi$ -competitive algorithm for scheduling packets with deadlines. In *Proc. Symp. Discrete Algorithms (SODA)*, 123–142, 2019.

# Buffer Minimization with Conflicts on a Line\*

Felix Höhne<sup>†</sup>

Rob van Stee (Speaker)<sup>‡</sup>

---

## 1 Introduction

Scheduling is a fundamental problem in computer science. It played an important part in the development of approximation and online algorithms. One of the earliest online algorithms was designed for makespan scheduling in the 1960s [1]. Many objective functions for online scheduling have been studied over the years. In the well-known area of throughput scheduling, the goal is to maximize the weighted or unweighted number of completed jobs (the throughput). Each job has a deadline and we only count the jobs that complete by their deadlines [2].

In the current paper, we consider a variation of this problem in which jobs do not have deadlines. Instead, jobs that arrive are stored in a buffer of some size until they can be processed. The buffer minimization in multiprocessor systems with conflicts or simply *buffer minimization* problem was first introduced by Chrobak et al. [3]. A sequence of tasks needs to be scheduled in a multi-processor system with conflicts. A conflict occurs if two processors share a common resource that they cannot both access at the same time. The multi-processor system is modelled as an undirected graph where the processors are the nodes of the graph. They are in conflict if they are connected by an edge. Without loss of generality, the conflict graph is connected.

In this paper we consider the special case where this graph is a path. At any time tasks may arrive on a processor adding to this processor's workload. Each processor stores its workload in a separate input buffer. The goal is to find a scheduling strategy that minimizes the maximum used buffer size of all processors.

An online algorithm processes the workload without knowledge of future tasks. We measure the performance of such algorithms using the competitive ratio. For the present problem, that is the ratio between the maximum used buffer size of an online algorithm and the buffer size of an optimal offline algorithm in the worst case. Chrobak et al. [3] provide results for several kinds of graphs, including  $K_n$  and trees as well as graphs with up to four vertices. In particular, they show that GREEDY is  $\frac{5}{2}$ -competitive on the graph that has four vertices on a path and they give a lower bound of 2 for the path with three vertices. In each time instant,

---

\*This paper previously appeared in 14th Workshop on Frontiers in Algorithmics (FAW'20).

<sup>†</sup>[felix.hoehne@uni-siegen.de](mailto:felix.hoehne@uni-siegen.de). University of Siegen, Walter-Flex-Strae 3, 57072 Siegen.

<sup>‡</sup>[rob.vanstee@uni-siegen.de](mailto:rob.vanstee@uni-siegen.de). University of Siegen, Walter-Flex-Strae 3, 57072 Siegen.

GREEDY iteratively picks the highest available machine and runs that machine (making its neighbors unavailable). In the case of two adjacent machines having the same load GREEDY rapidly switches between them. This can also be seen as both machines running at the same time at half speed. One of our main results is providing the exact competitive ratio for the path with four machines, which turns out to be  $\frac{9}{4}$ .

We also introduce the flow model. In the original model, blocks of load can arrive at any time, with size up to the offline buffer size. In contrast, in the flow model load only arrives incrementally at a rate of at most 1. Clearly, any algorithm for the original model can also be used in the flow model. However in many cases it is possible to do better. We provide modified algorithms which are optimal for the path with up to five machines.

In this paper, we assume that the size of the offline buffer is known and we scale it to 1. Chrobak et al. [3] also consider the case where the offline buffer is not known and show that the competitive ratio in that case is at most a factor of 4 higher.

We give an example using GREEDY in the flow model. Assume there are three machines 1, 2 and 3 which are empty at time  $t = 0$ . For one unit of time, load arrives at a rate of 1 on machines 1 and 2. GREEDY runs both machines at half speed. At time  $t = 1$ , GREEDY has load  $\frac{1}{2}$  on both machines 1 and 2, and no load on machine 3. Assume the optimal offline algorithm runs machine 2 at full speed, that is at a processing rate of 1, for that unit of time. Then at time  $t = 1$  the optimal offline algorithm has one unit of load on machine 1, and its other machines are empty.

For the next unit of time, load arrives at a rate of 1 on machine 3. Since the highest available machines are 1 and 2 and they have the same load, GREEDY initially runs all machines at speed  $\frac{1}{2}$ . This means the loads on machines 1 and 2 decrease at a rate of  $\frac{1}{2}$  while the load on machine 3 *increases* at a rate of  $\frac{1}{2}$ . After  $\frac{1}{2}$  unit of time, all machines of GREEDY have load  $\frac{1}{4}$ . Since machine 3 is still receiving load, GREEDY runs machines 1 and 3 at full speed for the remainder of this timestep. At time  $t = 2$ , GREEDY has no load on machine 1 and  $\frac{1}{4}$  load on machine 2 and 3. The optimal offline algorithm can run machines 1 and 3 at full speed for the entire unit of time, therefore its machines are empty at time  $t = 2$ .

## 2 Overview of results

We provide lower and upper bounds on the competitive ratios on the paths with four and five machines for both the original model as well as the flow model. For the path with four machines the ratio  $\frac{9}{4}$  is optimal in the original model, while in the flow model we find optimal ratios for both the path with four and five machines and these are  $\frac{4}{3}$  and  $\frac{3}{2}$ .

We also show that GREEDY is 1-competitive on two machines and give a 1-competitive algorithm on three machines. We also give a  $(\frac{1}{\varepsilon} + 2)$ -competitive algorithm, whose machines run at speed  $1 + 2\varepsilon$ . Finally, for large  $m$ , we show that

no algorithm can be better than  $\frac{12}{5}$ -competitive on  $m$  machines in the original model. The following table sums up our and previous bounds on the competitive ratio for different path lengths for both versions of the problem.

Number of machines	2	3	4	5	$m > 5$
Lower bound	$3/2$ [3]	$2$ [3]	$9/4$	$16/7$	$12/5$
Upper bound	$3/2$ [3]	$2$ [3]	$9/4$	$5/2$	$m/2 + 1/4$
Lower bound flow	1	1	$4/3$	$3/2$	$3/2$
Upper bound flow	1	1	$4/3$	$3/2$	$m/2 - 2/3$

Before this paper, the best known lower bound on general connected graphs (of any size) was only 2, whereas the best known upper bound is linear in the diameter of the graph (the length of the path, in our case). This upper bound (with a minor technical change) also applies to the flow model.

## References

- [1] RONALD L. GRAHAM, Bounds on Multiprocessing Timing Anomalies, *SIAM Journal of Applied Mathematics* 1969
- [2] JIŘÍ SGALL, Open Problems in Throughput Scheduling, *ESA* 2012
- [3] MAREK CHROBAK, JÁNOS CSIRIK, CSANÁD IMREH, JOHN NOGA, JIŘÍ SGALL AND GERHARD J. WOEGINGER, The Buffer Minimization Problem for Multiprocessor Scheduling with Conflicts, *ICALP* 2001

# Barely Random Algorithms for Online Scheduling with Equal Job Sizes

Neda Abbasi Shahkooch (Speaker) \*    Martin Böhm †    Lukasz Jeż ‡

---

## 1 Introduction

We consider the problem of non-preemptive throughput maximization where jobs of equal size (larger than 1) arrive over time, each with a release time and deadline. An online algorithm learns of the job and all its parameters upon its release. Since the job an algorithm start is wlog the one with minimum deadline job among those available, the only decision is when to start such job. Even so, every deterministic algorithm that runs any job when there are some available is 2-competitive [2], which is optimal [2]. Therefore, only randomized algorithms are of interest, for which the lower bound is  $\frac{4}{3}$  [3]. The  $\frac{5}{3}$ -competitive barely-random algorithm RANDLOCK of Chrobak et al. [1], which is “two-threaded”, i.e., chooses one of two deterministic algorithms (uniformly) at random, remains the only one known to have ratio below 2. The same article provides a  $\frac{3}{2}$  lower bound for such two-threaded algorithms, which is further improved to  $\frac{8}{5}$  when the two threads have equal probability.

## 2 Our algorithm

We give another  $\frac{5}{3}$ -competitive two-threaded algorithm that we call PROCRASTINATOR to stress how it differs from RANDLOCK. Both work as follows: Initially, the algorithm generates a bit  $r \in \{0, 1\}$  uniformly at random, simulates two threads (deterministic algorithms),  $T_0$  and  $T_1$ , on the whole input it receives, making the same decisions as  $T_r$ . The two threads are identical but they interact through a shared lock. Specifically, the set  $J_i$  of jobs pending at time  $t$  for  $T_i$  is *flexible* if all jobs in  $J_i$  can be completed if the earliest one is started at time  $t + p$ , and otherwise it is *urgent*. If at time  $t$  the set  $J_i$  is empty, then  $T_i$  idles until a new job is released, otherwise if  $J_i$  is urgent,  $T_i$  starts a job, and finally if  $J_i$  is flexible, the lock mechanism is employed. Here is where the two algorithms differ: In RANDLOCK, in such case  $T_i$  tries to acquire the lock; if it succeeds, it starts a

---

\*neda.abbasi.shahkooch@cs.uni.wroc.pl. Dept. of Computer Science, University of Wrocław

†boehm@cs.uni.wroc.pl. Dept. of Computer Science, University of Wrocław, Poland.

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

job, releasing the lock upon its completion, and otherwise it waits until the lock becomes available or  $J_i$  becomes urgent. In our algorithm, in such case  $T_i$  does the same except it only tries to acquire the lock only if at the moment *neither* is  $T_{1-i}$  running a job nor is  $J_{1-i}$  urgent (i.e., we assume that a thread whose set of jobs is urgent and that is not in the middle of processing a job, starts a job before the other thread tries to acquire the lock). In other words, when a thread has pending jobs that are not urgent, it waits (or procrastinates, hence the name) when the other thread is urgent. In both algorithm, when both threads try to acquire the lock, an arbitrarily chosen thread gets it. Especially when these choices consistently favor one thread over the other, the schedules produced by RANDLOCK and PROCRASTINATOR may differ substantially. Nevertheless, we prove the following.

**Theorem 1** *The competitive ratio of PROCRASTINATOR is exactly  $\frac{5}{3}$ .*

The instance which shows the ratio is at least  $\frac{5}{3}$  is the same as for RANDLOCK. The upper bound is proved by a charging scheme, which is substantially different from the one employed for RANDLOCK.

### 3 Further results

We investigate barely random algorithms further. For an algorithm that has  $k$  threads, we denote those by  $T_0, T_1, \dots, T_{k-1}$  and let  $\mathcal{T}$  denote the set of all those threads. In our lower bounds, we use the notion of a *subinstance* that may be *played against*  $(T, \mathcal{T}')$  at time  $t_0$ , where  $\mathcal{T}' \subseteq \mathcal{T}$  is the set of threads that are busy at time  $t_0$  and will remain busy until at least time  $t_0 + 2$  and  $T \in \mathcal{T} \setminus \mathcal{T}'$  is a thread that is not busy at time  $t_0$  before the subinstance is played. First, a job  $j_1$  with deadline  $t_0 + 2p + 1$  is released at time  $t_0$ . If  $T$  starts *any* job (including  $j_1$ ) at time  $t_0$ , then a tight job is released at time  $t_0 + 1$ , otherwise a tight job is released at time  $t_0 + p$ . We note the following:

1.  $T$  may complete at most one job from the subinstance, because by design the tight job cannot be scheduled together with  $j_1$ ,
2. Each thread in  $T' \in \mathcal{T}'$  can complete at most one job from the subinstance, because it is busy until (at least) time  $t_0 + 2$  and every job in the subinstance has deadline at most  $t_0 + 2p + 1$ ,
3. It is possible to schedule both jobs from the subinstance in  $[t_0, t_0 + 2p + 1]$ .

This roughly corresponds to known lower bounds [2, 3] when  $\mathcal{T}' = \emptyset$  and the subinstance is the whole instance: The optimum gain is 2, if we consider a one-threaded (i.e., deterministic) algorithm such that  $T$  is its only thread, then its gain is 1, and if we consider a two-threaded algorithm such that  $T$  is a thread with probability  $q$ , then that algorithm's expected gain is at most  $1 + (1 - q)$ ; choosing  $T$  as the thread with larger probability, that expected gain is at most  $\frac{3}{2}$ , yielding lower bound of  $\frac{4}{3}$ .

**Theorem 2** *Any three-threaded algorithm has competitive ratio at least 1.5 regardless of the probability distribution over the threads.*

**Proof:** Suppose that there is a  $(1.5 - \epsilon)$ -competitive three-threaded algorithm. Fix  $(\xi, \delta)$  such that  $1 \leq \xi \leq p - 3$  and  $\xi + 2 \leq \delta \leq p - 1$ , and consider the following instance. First, a job  $j_1$  with deadline  $3p + \delta$  arrives at time 0. We will number the threads  $T_0, T_1, T_2$  in a non-decreasing order of the times  $\tau_0, \tau_1, \tau_2$  they start  $j_1$  if no other job is released, breaking ties arbitrarily. We note that the first starting time  $\tau_0$  must be equal to 0, because the rest of the instance may consist of tight jobs released at times  $p$  and  $2p$ . Consider the following cases regarding  $\tau_1$ :

- If  $\tau_1 \leq \xi$ , then at time  $\tau_1 + 1$  the subinstance is played against  $(T_2, \{T_0, T_1\})$ . By the subinstances property, each thread completes at most one job from it and possibly also  $j_1$ . OPT schedules  $j_1$  after both jobs from the subinstance.
- If  $\tau_1 > \xi$ , then at time  $p$  the subinstance is played against  $(T_0, \emptyset)$ . By the subinstances property,  $T_0$  completes at most one job from it and also  $j_1$ . We claim that  $T_1$  and  $T_2$  also complete at most two jobs. Suppose either completes all three jobs, i.e., both jobs from the subinstance in  $[p, 3p + 1]$  and also  $j_1$ . Then such thread has to complete  $j_1$  either no later than at time  $p + 1$ , which is impossible, because  $\tau_2 \geq \tau_1 \geq \xi + 1 \geq 2$ , or it has to start  $j_1$  no earlier than at time  $3p$ , which is again impossible because its deadline is  $3p + \delta \leq 4p - 1$ . Hence, each thread completes at most two jobs, whereas OPT schedules  $j_1$  in  $[0, p]$ , followed by both jobs from the subinstance.

Thus, every thread completes at most two jobs in either case while OPT completes three jobs.  $\blacksquare$

To the best of our knowledge, a four-threaded algorithm may be able to attain ratio  $\frac{4}{3}$ .

We also note that if there is a two-threaded 1.5-competitive algorithm, then its probability distribution over threads is uniquely determined.

**Lemma 3** *If ALG is a two-threaded 1.5-competitive algorithm, then its threads have probabilities  $\frac{1}{3}$  and  $\frac{2}{3}$ .*

**Proof:** Consider ALG on the instance which proves the lower bound  $\frac{8}{5}$  for two-threaded algorithms with uniform probability [1]: First, a job  $j_1$  with deadline at least  $6p$  is released at time 0. Let the threads  $T_0$  and  $T_1$  start that job at times  $\tau_0$  and  $\tau_1$  respectively if no other job is released, where  $\tau_0 \leq \tau_1$ . For the second starting time, we have  $\tau_1 \geq \tau_0 + p - 1$ , because otherwise neither thread could complete a tight job released at time  $\tau_1 + 1$ , yielding ratio at least 2. At time  $\tau_0 + 1$ , a tight job  $j_2$  is released. As  $T_0$  cannot complete it,  $T_1$  must do so, because otherwise the ratio would again be at least 2. Next, let  $\tau'_1 \geq \tau_0 + p + 1$  be the time when  $T_1$  starts  $j_1$  after completing  $j_1$ . We play the subinstance against  $(T_0, \{T_1\})$  at time  $\tau'_1 + 1$ . Both threads complete  $j_1$  and at most one job from the subinstance by the subinstance property, and, crucially, only  $T_1$  completes  $j_2$ .

So if  $q_0$  and  $q_1$  denote the probabilities of the threads, then the expected gain of ALG is at most  $2 + q_1$ . The optimum solution completes all four jobs ( $j_1$ 's deadline of  $6p$  suffices for this, but making it arbitrarily large makes this property trivial). Hence, since ALG is 1.5-competitive, we get  $q_1 \geq \frac{2}{3}$ .

For the upper bound on  $q_1$ , let us consider a new instance, which consists solely of the subinstance played against  $(T_1, \emptyset)$ . As the optimum gain on such an instance is 2, the expected gain of ALG is  $2 - q_1$ , and it is 1.5-competitive, we get that  $q_2 \leq \frac{2}{3}$ .

Hence,  $q_1 = \frac{2}{3}$  and  $q_0 = 1 - q_1 = \frac{1}{3}$ . ■

Furthermore, our analysis shows that to break the  $\frac{5}{3}$  barrier, a two-threaded algorithm when presented with the lower bound instance of [1] must start the long job on the second thread before the sub-instance is complete.

We can make this property much stronger, as the following theorem explains.

**Theorem 4** *Let  $A$  be a two-equal-thread algorithm which is strictly better than  $5/3$ -competitive. Let  $0 \leq \Delta \leq p - 4$  be a parameter. Assume that  $A$  starts the first job  $L$  at time  $t$ , and assume  $d_L \geq t + 6p + \Delta + 6$ .*

*If, at time  $t + 1$ , a job  $M$  with  $d_M = t + 3p + \Delta + 1$  is released, then it must hold that the starting time of  $M$  on one of the threads, denoted by  $S_M$ , must lie in  $[t + \Delta - 1, t + \Delta + 3]$ .*

*Furthermore, if a new tight job is released at time  $S_M + 1$ , then both threads must complete all occurrences of the jobs  $M$  and  $L$  by time  $S_M + 2p + 2 \leq t + 2p + \Delta + 5$ .*

## References

- [1] MAREK CHROBAK, WOJCIECH JAWOR, JIŘÍ SGALL, AND TOMÁŠ TICHÝ *Online Scheduling of Equal-Length Jobs: Randomization and Restarts Help*. SIAM J. Comput. 36(6): 1709–1728 (2007).
- [2] SANJOY K. BARUAH, JAYANT R. HARITSA, AND NITIN SHARMA *On-line scheduling to maximize task completions*. Proceedings of the IEEE Real-Time Systems Symposium (RTSS): 228–236 (1994).
- [3] SALLY A. GOLDMAN, JYOTI PARWATIKAR, AND SUBHASH SURI *Online scheduling with hard deadlines* J. Algorithms 34(2): 370-389 (2000).

# A parametric flow-based algorithm for crew diagramming: a pragmatic large-scale approach at the Swiss railways

Raphael Haemmerli \*      Alexander Souza (Speaker) \*  
Denys Trieskunov \*      Hsuan-Pin Wu \*

---

## 1 Introduction

One of the central planning challenges in the realm of mass transportation is to determine shifts for crews (located at a depot), e.g., drivers, pilots, or conductors. A shift, also called a *crew diagram*, describes the activities, called *crew requirements*, the crew has to execute during a day of operations, e.g., setup-, driving-, shutdown-, passride-, and break-tasks. Labor rules must be respected to ensure the safety of the service.

Mathematically, the CREW DIAGRAMMING problem asks to partition a set of crew requirements into a set of crew diagrams, while respecting constraints, such as chronological consistency, local consistency (bridging gaps by passrides), maximum duration of uninterrupted work (ensured by breaks), maximum diagram duration, and start- and end-location at the depot of the crew. The objective is to minimize a weighted sum of the number of crew diagrams and further problem data. It is related to SET COVER, respectively SET PARTITION, see [1], and is NP-complete (in its decision version).

CREW DIAGRAMMING has attracted substantial research from the operations research community, see, e.g., [3, 8, 10, 11, 5, 6, 4, 9, 7]. The dominant approach is column-generation [12], with SET COVER as the master problem [3, 10, 5, 4, 9] and RESOURCE CONSTRAINED SHORTEST PATH (RCSP) as the subproblem [14, 5, 13, 9].

An operational optimizer, which is based on column-generation, was rejected by the planning department at the Swiss railways due to the following reasons: unsatisfactory runtimes, especially for medium and large depots; deteriorated quality of the solution as a result of postprocessing (needed for converting multiply covered crew requirements into passrides); and the variability of the solutions due to built-in randomization.

---

\* { raphael | alexander | denys | hsuan-pin }@algomia.com      Algomia      GmbH,  
Schachenallee 29, CH-5000 Aarau, Switzerland.

## 2 A parametric flow-based algorithm

This research originates from our work at the Swiss railways, where we designed and implemented a new optimizer for CREW DIAGRAMMING. Our goal was to address the reasons leading to the rejection of the existing optimizer. In our approach, we introduce a variety of parameters for our algorithm, allowing the planners to balance between runtime, feasibility, and optimality. Hence we leave it up to the users, which aspect of the optimization they want to emphasize. In our experience, speed is by far the most important feature, followed by feasibility and finally optimality. The main reason is that planners always adjust the results of the optimizer, e.g., to incorporate soft-knowledge or preferences that are not reflected therein.

Even during the development, we tested our prototypes on real-world data. This allowed us to reject approaches that performed poorly already early on. Furthermore, our solutions have been constantly validated by super-users, leading to incorporation of soft-knowledge and a very good acceptance by the customer.

The basic idea of the algorithm is to combine a state-expanded MINIMUM COST FLOW formulation with additional SET PARTITION constraints. A source vertex in the flow network represents the start of the planning day at the depot, while a target represents the end there. Any other vertex represents the execution of a crew requirement in a crew diagram and contains additional values, called resources. These represent (the relevant part of) the history of the vertex, e.g., the start of its crew diagram, the end of the last break in the diagram, and so on. If there exists an edge  $uv$ , between vertices  $u$  and  $v$ , then the crew requirement of  $v$  can be executed after  $u$ , and the resource-values are propagated accordingly. The edges carry supporting information, like, breaks or passrides that are planned between crew requirements. By construction, each path from the source to the target represents a valid crew diagram. Capacity constraints ensure that at most one unit of flow passes through each vertex. Hence the entire flow decomposes into disjoint paths. SET PARTITION constraints ensure that each crew requirement occurs in exactly one vertex with positive flow.

The algorithm has several parameters that are aimed to balance between optimality, feasibility, and speed. Clearly, the state-expansion has to be done with care, because it can lead to combinatorial explosion of the network, otherwise. We do so by introducing discretization parameters that lead to “rounding” of the resource-values, thus significantly reducing the state-space. Of course, we only generate states that can be reached from the source. Furthermore, we find that the LP-relaxation of our formulation is very strong empirically: usually, the LP- and ILP-optima are only a few percent away from each other. To accelerate the algorithm even further, we shrink the ILP after the LP-relaxation with vertex-elimination heuristics. In case of multi-depot/multi-group planning, the algorithm allows the user to compute a pre-assignment of the crew requirements, which leads to a further significant speed-up (at the potential expense of optimality). It is furthermore possible to split large instances by solving a relaxed version

of the problem. All these parametrization options lead to attractive runtimes in practice with good solution qualities.

The following results have been obtained with a cloud-instance having 128 GB of main memory and 28 CPU cores.

Depot (Groups)	CRs	Manual CDs	Optimizer CDs	$\Delta$	Runtime [s]
02 (1)	111	11	11	0	12
04 (1,2)	699	47	44	-3	330
07 (1,2)	372	30	30	0	845
10 (1)	208	18	18	0	10
18 (1,2,3)	579	40	39	-1	135
19 (1,2,3,4)	688	56	57	+1	968
21 (1,2)	817	58	55	-3	2412
36 (1,2,3,4)	1016	80	80	0	5264

In conclusion, our algorithm is capable of solving all 36 depots at the Swiss railways sequentially in about 3 hours. Also simultaneous multi-depot optimization is possible. All of the criticism against the existing optimizer has been addressed: our algorithm has good runtimes, does not need postprocessing, and is deterministic. The most important feature is that it works well in practice.

## References

- [1] M.R. GAREY AND D.S. JOHNSON (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.
- [2] R. KWAN (2011). Case studies of successful train crew scheduling optimisation. *J. Scheduling*, 14(5). 423-434. pp. 49-54, 2011.
- [3] B.M. SMITH AND A. WREN (1988). *A bus crew scheduling system using a set covering formulation*. In: *Transportation Research*, vol. 22A, pp. 97-108, 1988.
- [4] S. FORES, L. PROLL AND A. WREN (2002). *TRACS: a hybrid IP/heuristic driver scheduling system for public transport*. *Journal of the Operational Research Society*, 53:1093-1100, 2002
- [5] A.S.K. KWAN (1999). *Train Driver Scheduling*, PhD Thesis, University of Leeds
- [6] G.P. SILVA, N.D.F. GUALDA AND R. KWAN (2000). *Bus Scheduling Based on an Arc Generation – Network Flow Approach*, 8th International Conference on Computer-Aided Scheduling of Public Transport - CASPT 2000
- [7] A. KWAN, M. PARKER, R. KWAN, S. FORES AND A. WREN (2022). *Recent advances in TRACS*, Technical Report, University of Leeds, 2022.

- [8] J.E. BEASLEY AND B. CAO (1996). *A tree search algorithm for the crew scheduling problem*, J. Operational Research 94, pp. 517 – 526, 1996.
- [9] S. JÜTTE (2012). *Large-Scale Crew Scheduling: Models, Methods, and Applications in the Railway Industry*, Springer Verlag, 2012.
- [10] R. BORNDÖRFER, M. GRÖTSCHEL, AND A. LÖBEL (1998). *Optimization of Transportation Systems*. Technical Report ZIB-Report 98-09, Zuse-Institut Berlin, 1998
- [11] R. BORNDÖRFER, A. LÖBEL, U. STRUBBE, AND M. VÖLKER (1999). *Zielorientierte Dienstplanoptimierung*. In: Heureka '99: Optimierung in Verkehr und Transport, pp. 171194, Köln, 1999. ZIB Report 98-41.
- [12] G.B. DANTZIG AND P. WOLFE (1960). *Decomposition Principle for Linear Programs*. Operations Research, 8(1):101111, 1960.
- [13] S. IRNICH AND G. DESAULNIERS (2005). *Shortest Path Problems with Resource Constraints*. In: Desaulniers G., Desrosiers J., Solomon M.M. (eds) Column Generation. Springer, Boston, MA, 2005.
- [14] M. DESROCHERS (1986). *La Fabrication d'horaires de travail pour les conducteurs d'autobus par une méthode de génération de colonnes*. Ph.D Thesis, Centre de recherche sur les Transports, Publication 470, Université de Montréal, Canada, 1986.

# Independent Sampling for the Secretary Problem

José Correa <sup>\*</sup>      Andrés Cristi <sup>†</sup>      Laurent Feuilloley <sup>‡</sup>

Tim Oosterwijk (Speaker) <sup>§</sup>      Alexandros Tsigonias-Dimitriadis <sup>¶</sup>

---

## 1 Introduction

The secretary problem is probably the most well-studied optimal stopping problem with many applications in economics and management. In the secretary problem, a decision-maker faces an unknown sequence of values, revealed one after the other, and has to make irrevocable take-it-or-leave-it decisions. Her goal is to select the maximum value in the sequence. While in the classic secretary problem, the values of upcoming elements are entirely unknown, in many realistic situations, the decision-maker still has access to some information, for example, in the form of past data. In this paper, we take a sampling approach to the problem and assume that before starting the sequence, each element is independently sampled with probability  $p$ . This allows us to quantify the relationship between the availability of information and the success guarantee (i.e., the worst case probability of selecting the maximum value). Our results interpolate between the setting with unknown values ( $p = 0$ ) and the full-information setting ( $p = 1$ ) in which the values are assumed to originate from a distribution known to the algorithm.

For the classical problem where the unknown values are revealed in a random order, the best possible success guarantee has long been known to be  $1/e$  [2, 5]. Gilbert and Mosteller [4] considered the full information secretary problem and were able to conclude, numerically, that the best possible success probability is  $\gamma \approx 0.5801$ . In subsequent work, Samuels [7] finds an explicit expression for this quantity.

For the variant of the secretary problem where the values are revealed in adversarial order, the results are worse. Without any information, no success guarantee

---

<sup>\*</sup>correa@uchile.cl. Department of Industrial Engineering, Universidad de Chile, Beauchef 851, Santiago, Chile.

<sup>†</sup>andres.cristi@ing.uchile.cl. Department of Industrial Engineering, Universidad de Chile, Beauchef 851, Santiago, Chile.

<sup>‡</sup>lfeuilloley@liris.cnrs.fr. Univ. Lyon, Université Lyon 1, LIRIS UMR CNRS 5205, F-69621, Lyon, France.

<sup>§</sup>t.oosterwijk@vu.nl. Department of Operations Analytics, Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV, Amsterdam, the Netherlands.

<sup>¶</sup>alexandros.tsigonias@tum.de. Operations Research, TU München, Arcisstr. 21, 8033,3 München, Germany.

larger than 0 can be achieved. Allaart and Islas [1], and independently Esfandiari et al. [3], considered the adversarial order secretary problem in which an adversary chooses  $n$  distributions  $F_1, \dots, F_n$  from which independent values are drawn and sequentially uncovered. They prove that the best possible success guarantee equals  $1/e$ .

## 2 Problem description

We are given  $n$  elements with values  $\alpha_1, \dots, \alpha_n$ , which are unknown to us, and an order  $\sigma : [n] \rightarrow [n]$ . Each element is sampled independently with probability  $p$ . Let  $S$  be the (random) set of sampled elements and  $V$  be the remaining elements, also referred to as the online set or the set of online elements. The elements in  $V$  are then presented to us in the order dictated by  $\sigma$ . Once an element is revealed we either pick it and stop the sequence or drop it forever and continue. The goal is to maximize the probability of picking the maximum valued element in  $V$ . In the *adversarial order secretary problem with  $p$ -sampling* (AOS $p$ ) the order  $\sigma$  is chosen by an adversary that knows all values  $\alpha_1, \dots, \alpha_n$  and the random sets  $S$  and  $V$ .<sup>1</sup> In the *random order secretary problem with  $p$ -sampling* (ROS $p$ ) the order  $\sigma$  is just a uniform random permutation.

Given  $n$  and an algorithm we define its *success probability* as the infimum over all values  $\alpha_1, \dots, \alpha_n$  of the probability that the algorithm stops with the maximum  $\alpha_i \in V$ . Moreover, the *success guarantee* of an algorithm is the infimum over all values of  $n$  of its success probability.

## 3 Results

### 3.1 AOS $p$

For AOS $p$  we define the  *$k$ -max algorithm* as follows: the  $k$ -th largest value of the sampled elements is set as a threshold, and the algorithm accepts the first element in the set  $V$  of online values whose value surpasses this threshold. If there are less than  $k$  sampled elements, then the algorithm accepts the first online element.

**Theorem 1** *Let  $k = \lfloor \frac{1}{1-p} \rfloor$ . Then the  $k$ -max algorithm achieves a guarantee of  $kp^k(1-p)$  for AOS $p$ . Furthermore, no algorithm can achieve a better success guarantee.*

The proof of the guarantee of the algorithm is easy: Assume that the values are sequenced in non-decreasing order. An instance in which the algorithm is successful is exactly a sequence ending in  $k$  sampled elements plus one online element that is somewhere in the last  $k$  entries of the sequence. The probability that this happens equals  $kp^k(1-p)$ . The proof that this is the best possible consists

---

<sup>1</sup>Our results, and in particular the upper bounds on the success probability, remain true if the adversary knows all values  $\alpha_1, \dots, \alpha_n$  but not the result of the sampling process.

of several steps. We start by considering the special case where the algorithm does not know  $n$ , meaning, it needs to make the same decision at the same point in instances that look the same up to that point but have a different total number of elements. Assuming non-decreasing values again, we can reduce an instance to a binary string that states for each element whether it is sampled (1) or not (0), and the algorithm needs to stop at the last 0. This allows us to create an infinite conflict graph with a vertex for every possible instance (binary string) and an edge between two instances if and only if they are in conflict: The algorithm can only win in one of the two instances because it needs to make the same decision. The result follows from arguments on this conflict graph.

### 3.2 ROS $p$

For ROS $p$  it is useful to have the following equivalent point of view. We assign a uniformly random arrival time  $\tau_i$  to each of the  $n$  elements in the interval  $[0, 1]$ . If  $\tau_i < p$  we add  $i$  to  $S$  and otherwise we add it to  $V$ . Then the elements in  $V$  are revealed in the order of the  $\tau_i$ 's.

Consider the following family of algorithms. We fix a sequence  $t = (t_i)_{i \in \mathbb{N}}$  such that  $0 \leq t_1 < t_2 < \dots < 1$ . Between times  $t_k$  and  $t_{k+1}$  the algorithm  $ALG_t$  sets as a threshold the  $k$ -th largest sampled value. More precisely, suppose the value  $\alpha_i$  is revealed and assume  $t_k \leq \tau_i < t_{k+1}$ .  $ALG_t$  accepts  $\alpha_i$  if it is the largest among the values from  $V$  seen so far, and is greater than the  $k$ -th largest value from  $S$ . For simplicity, if  $|S| < k$  we define the  $k$ -th largest value of  $S$  as  $-\infty$ .

**Theorem 2** *The sequence  $t^*$  defined as the unique solution of the equations*

$$\ln\left(\frac{1}{t_i^*}\right) + \sum_{j=1}^{i-1} \frac{(1/t_i^*)^j - 1}{j} = 1, \quad \text{for all } i \in \mathbb{N},$$

*obtains the best possible success guarantee for ROS $p$ .  $t^*$  is independent of  $p$  and  $n$ .*

We prove this theorem in two main steps. First, we find the sequence  $t^*$  that maximizes the success guarantee of  $ALG_t$ . Then, we find an expression for the optimal success probability when  $p$  and  $n$  are given, and prove that for fixed  $p$  it converges to the success guarantee of  $ALG_{t^*}$  when  $n$  tends to infinity.

### 3.3 Discussion

Figure 1 presents a plot of the obtained success guarantees for ROS $p$  and AOS $p$ . As one can see, our results interpolate between the setting with unknown values ( $p = 0$ ) and the full-information setting ( $p = 1$ ).

An interesting connection arises between our model and results when  $p$  is close to 1, and the so-called full information case of the secretary problem where the elements' values are taken as i.i.d. random variables from a known distribution. It

may seem natural that our guarantee matches this quantity as  $p \rightarrow 1$ . However, this is far from obvious. Indeed, for the prophet inequality with i.i.d. values from an unknown distribution, Rubinstein et al. [6] proved that with  $O(n)$  samples one can achieve the best possible performance guarantee of the case with known distribution. This is in line with our result here since for  $p$  close to, but strictly less than 1, the size of the sample set is linear in the size of  $V$ .

A more intriguing connection to the full information case pops up in the adversarial order case. Although this problem has a similar flavor as our  $\text{AOS}_p$ , and the optimal guarantee is the same, we are unaware of a precise connection.

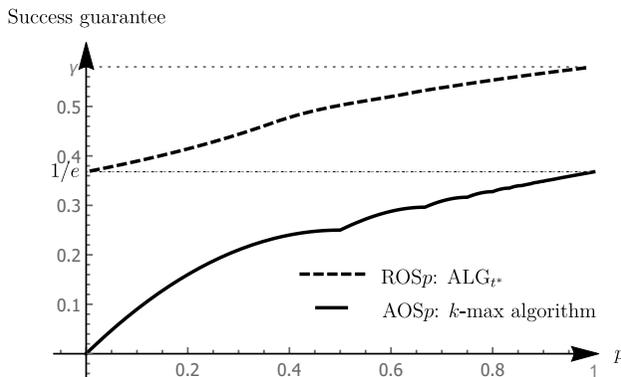


Figure 1: The best possible success guarantee for  $\text{ROSp}$  and  $\text{AOS}_p$  as a function of  $p$ .

## References

- [1] Allaart, P., Islas, J. A sharp lower bound for choosing the maximum of an independent sequence. *Journal of Applied Probability*, 53:1041–1051, 2015.
- [2] Dynkin, E.B. The optimum choice of the instant for stopping a Markov process. *Soviet Math. Dokl.* 4:627–629, 1963.
- [3] Esfandiari, H., HajiAghayi, M., Lucier, B., Mitzenmacher, M. Prophets, secretaries, and maximizing the probability of choosing the best, AISTATS 2020.
- [4] Gilbert, J., Mosteller, F. Recognizing the maximum of a sequence. *Journal of the American Statistical Association*, 61:35–73, 1966.
- [5] Lindley D.V. Dynamic programming and decision theory. *Journal of the Royal Statistical Society (Series C Applied Statistics)*, 10:39–51, 1961.
- [6] Rubinstein, A., Wang, J.Z., Weinberg, S.M. Optimal single-choice prophet inequalities from samples. ITCS 2020.
- [7] Samuels, S. Secretary Problems. In *Handbook of Sequential Analysis*, Chapter 16, (B. Ghosh, P. Sen, Eds.), CRC Press, 1991.

## Collective schedules: how to find a consensus schedule?

Martin Durand \* (Speaker)

Fanny Pascual \*

---

The *Collective Schedules* [4] problem deals with a set  $\mathcal{J}$  of  $n$  tasks shared by  $v$  agents. Tasks have to be executed on a single machine, without preemption and idle times. Each task  $i$  has a processing time  $p_i$ . Each agent  $k$  has a preferred schedule  $V^k$  corresponding to the order in which she would like the  $n$  tasks to be executed. The  $v$  preferred schedules form a preference profile  $P$ . Given a preference profile  $P$ , our goal is to return a schedule – called a *consensus schedule* – of tasks  $\mathcal{J}$ , which satisfies the agents as much as possible. We call *aggregation rule* a mapping between a preference profile and a single schedule. We will study axiomatic and algorithmic properties of two aggregation rules.

**Example 1.** *A city has decided the construction of several public infrastructures over the next years (park, pool, library, ...). Each project (task) has a defined length. Because of budget and workforce constraints the city cannot realize projects simultaneously and has to decide in which order these infrastructures will be built. Citizens express their preferences over this order. In such a context, if this were possible, all the citizens would agree to schedule all the tasks simultaneously as soon as possible.*

**Example 2.** *Workers share different works that have to be done in teams. Each work (task) has a given duration and can imply a different investment of each worker (professional travel, staggered working hours, ...). Each worker indicates his or her favorite schedule according to his or her personal constraints and preferences. In this setting, it is natural to try to fit as much as possible to the schedules wanted by the workers – and scheduling tasks earlier than wanted by the agents is not a good thing.*

In the particular case in which all the tasks have the same length, our problem is the consensus ranking problem, a classical problem in social choice [1]. In this problem, agents express their preferences over a set of candidates as a ranking of these candidates: from the most preferred, to the least preferred. The goal is to find a consensus ranking of the candidates, aggregating the preferences of all the agents. This is similar to our problem in which a unit task is a candidate and a schedule is a ranking. However, when tasks have different length, traditional aggregation rules cannot be used directly.

---

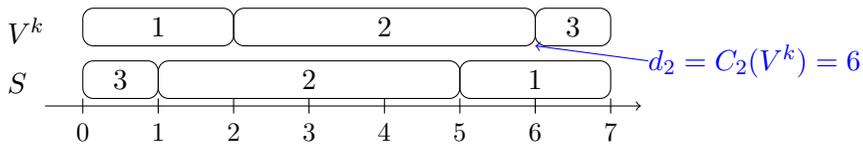
\*martin.durand@lip6.fr,fanny.pascual@lip6.fr. Sorbonne Université, CNRS, LIP6, 4 place Jussieu, 75005 Paris, France.

The Collective schedules problem has been introduced in [4], where the authors present how traditional scheduling criteria [2] can be used to design aggregation rules. In particular, they introduce and study the  $\Sigma T$  rule that we will present below.

**Contributions.** In this work, we focus on two rules for the Collective schedules problem, and we study them on an axiomatic and algorithmic point of view. In section 1, we introduce these rules, called  $\Sigma T$  and  $\Sigma D$ . In section 2, we study the axiomatic properties of these rules. Several of these properties are classical axioms from the computational social choice field. We also introduce new axioms, specific to our setting. In section 3, we study the computational properties of these rules and propose an efficient heuristic for the  $\Sigma D$  rule.

## 1 Two aggregation rules

The rules that we study now are based on the completion times of tasks in the schedules preferred by the agents. These completion times are interpreted as due dates for the tasks.



To measure how close a schedule  $S$  is to a schedule  $V^k$ , we measure how close each task is in  $S$  compared to its due date in  $V^k$ . We will focus on the tardiness of a task, and on the deviation of a task. These are two standard criteria in scheduling [2]. If we denote by  $d_i$  the completion time of task  $i$  in  $V^k$ , then the *tardiness* of task  $i$  in  $S$  is  $T_i(S) = \max(0, C_i(S) - d_i)$ , where  $C_i(S)$  is the completion time of task  $i$  in  $S$ . For example, in the figure above, we have  $d_1 = 2, C_1(S) = 7$  and thus  $T_1(S) = 5$ , while  $d_2 = 6, C_2(S) = 5$  and thus  $T_2(S) = 0$ . The *deviation* of task  $i$  in a schedule  $S$  is the absolute value of the difference between the completion time of  $i$  in  $S$  and its due date:  $D_i(S) = |C_i(S) - d_i|$ . For example,  $D_1(S) = 5$ , while  $D_2(S) = 1$ . We can sum the tardiness (or deviations) of all the tasks to obtain a metric measuring the difference between two schedules.

**Minimizing the sum of tardiness.** The tardiness of a schedule  $S$  is the sum of the tardiness of the tasks in  $S$  in comparison to the preferred schedules of the agents. By denoting  $C_i(V^k)$  the completion time of task  $i$  in  $V^k$ , we get:

$$\Sigma T(S, P) = \sum_{V^k \in P} \sum_{i \in \mathcal{J}} \max(0, C_i(S) - C_i(V^k))$$

The  $\Sigma T$  rule [4] returns a schedule minimizing the tardiness over all the agents and all the tasks. In contexts similar to example 1, minimizing the tardiness seems natural.

**Minimizing the sum of deviations.** The deviation of a schedule  $S$  is the sum of the deviations of the tasks in  $S$  in comparison to the preferred schedules of the agents:

$$\Sigma D(S, P) = \sum_{V^k \in P} \sum_{i \in \mathcal{J}} |C_i(S) - C_i(V^k)|$$

The  $\Sigma D$  rule returns a schedule minimizing the deviation with the preference profile. The deviation criterion makes particular sense when applied on situations like the one in example 2, in which each task should be scheduled as close as possible to its position in the preferred schedules of the agents.

## 2 Axiomatic study

In this section, we consider several axioms from the computational social choice field [1] as well as new axioms, specific to our context. Among the classical axioms, we focus on:

- **Neutrality (N).** An aggregation rule  $R$  is *neutral* if it treats equally all the candidates (in our case, the tasks).
- **Reinforcement (R).** An aggregation rule  $R$  fulfills *reinforcement* [1] if, when a same ranking  $r$  is returned by  $R$  on two distinct subsets of voters  $A$  and  $B$ , the rule  $R$  also returns the ranking  $r$  when applied on  $A \cup B$ .

We introduce two new axioms : *PTA-Neutrality* (PTA-N), where PTA stands for Processing-Time Aware, and *length reduction monotonicity* (LRM).

- **PTA-Neutrality.** A rule  $R$  is *PTA-neutral* if it treats equally all the tasks that have the same processing time.
- **Length reduction monotonicity.** Let  $P$  and  $P'$  be two preference profiles which are similar except that a task  $i$  in  $P$  is replaced by a task  $i'$  in  $P'$  with  $p_{i'} < p_i$ . A rule  $R$  fulfills length reduction monotonicity if, in the schedules  $S$  and  $S'$  returned by  $R$  on  $P$  and  $P'$  respectively, the task  $i'$  does not start later in  $S'$  than  $i$  in  $S$ . Intuitively, this means that a task cannot start later because it is shorter.

We also study the PTA-Condorcet consistency (PTA-C) axiom, introduced in [4], an extension of the Condorcet consistency, a classical axiom in social choice, and we studied whether the  $\Sigma T$  and  $\Sigma D$  are distances. Our results are summarized in Table 1.

Rule	N	PTA-N	R	LRM	PTA-C	Distance
$\Sigma T$	✗	✓	✓ [4]	?	✗ [4]	✗
$\Sigma D$	✗	✓	✓ [4]	✗	✗	✓

Table 1: Axioms fulfilled (✓) or not (✗) by the  $\Sigma T$  and  $\Sigma D$  rules.

We also studied incompatibilities between axioms. For example we showed that the neutrality and the PTA-Condorcet consistency axioms are incompatible. Additionally, we prove that the schedule returned by the  $\Sigma D$  rule can have a total sum of tardiness arbitrarily far from the minimum sum of tardiness.

### 3 Complexity and algorithms

It has already been proved that the  $\Sigma T$  rule solves a strongly NP-hard problem [4]. We prove that the  $\Sigma D$  rule also solves a strongly NP-hard problem. A linear program allows us to solve optimally this problem up to 14 tasks and 500 agents.

In order to solve larger instances, we propose a polynomial time heuristic. For each task, we compute its median completion time in the preference profile. Tasks are then scheduled by increasing median completion time. In the worst-case, this heuristic can return a solution arbitrarily far from the optimal one, but it behaves very well in practice. We complete this heuristic with a local search step, based on swapping consecutive tasks in the solution. In our experiments, this heuristic returns schedules with total deviation 1% higher than the optimal sum of deviations. In terms of computation times, it takes on average 0.25 second to solve instances with 10 tasks and 100 agents, whereas linear programming takes 2 seconds.

Finally, we compare, in term of sum of tardiness and deviation, the schedules returned by the two rules  $\Sigma T$  and  $\Sigma D$ , as well as the schedules returned by another rule that we introduce and which extends the Kemeny rule [3], a classical rule in social choice.

### 4 Discussion

This work is a first step towards the study of collective decision processes to schedule common tasks. In this work, we considered that all the tasks have to be scheduled sequentially. Some other settings, where tasks could be scheduled in parallel (which corresponds to the parallel machines scheduling problem case), could be considered. It would also be interesting to study the case where there are precedence constraints between the tasks. These precedence constraints could be either given, or inferred from the preferences of the voters (this could for example lead to rules fulfilling the unanimity axiom – if each voter schedules task  $a$  before task  $b$  then in the returned schedule  $a$  should be scheduled before  $b$ ).

**Acknowledgements.** We acknowledge a financial support from the project THEMIS ANR-20-CE23-0018 of the French National Research Agency (ANR).

### References

- [1] F.BRANDT, V.CONITZER, U.ENDRISS, J.LANG, A.D.PROCACCIA (2016). *Handbook of Computational social choice*, Cambridge University Press.
- [2] P.BRUCKER (1995). *Scheduling algorithms*, Springer.
- [3] J.G.KEMENY (1959) *Mathematics without numbers*, Daedalus.
- [4] F.PASCUAL, K.RZADCA AND P.SKOWRON (2018). *Collective Schedules: Scheduling Meets Computational Social Choice*. AAMAS.

# Tight Competitive Analyses of Online Car-sharing Problems

Ya-Chun Liang (Speaker) \*    Kuan-Yun Lai †    Ho-Lin Chen ‡  
Kazuo Iwama §    Chung-Shou Liao ¶

---

## 1 Introduction

Our problem in this study is the *online car-sharing problem*. In car-sharing (not only for cars, but also for other resources like bikes and shuttle-buses), there are several service stations in the city, for instance in residential areas and downtown, at popular sightseeing spots, and so on. Customers can make a request with a pick-up time and place and a drop-off time and place. The decision for accepting or rejecting a request should be made in an online fashion and we want to maximize the profit by accepting as many requests as possible. Relocation of (unused) resources is usually possible with a much smaller or even negligible costs. (It is seen occasionally that a truck is carrying bikes for this purpose.) Theoretical studies of this problem have started rather recently and turned out to be nontrivial even for two locations.

We basically follow the problem setting of previous studies by Luo et al. [1, 2, 4, 3] with main focus to that of two locations and  $k$  servers (i.e. cars). The two locations are denoted by 0 and 1 and  $k(\geq 2)$  servers are initially located at location 0. The travel time from 0 to 1 and 1 to 0 is the same, denoted by  $t$ . The problem for  $k$  servers and two locations is called the  $kS2L$  problem for short.

We denote the  $i$ -th request by  $r_i = (\tilde{t}_i, t_i, p_i)$  which is specified by the *release time* or the *booking time*  $\tilde{t}_i$ , the *start time*  $t_i$ , and the *pick-up location*  $p_i \in \{0, 1\}$  (the *drop-off location* is  $1 - p_i$ ). If  $r_i$  is accepted, the server must pick up the customer at  $p_i$  at time  $t_i$  and drop off the customer at  $1 - p_i$  at time  $t_i + t$ .

---

\*ycliang512@gapp.nthu.edu.tw. Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 30013, Taiwan.

†lj841113@gapp.nthu.edu.tw. Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 30013, Taiwan

‡holinchen@ntu.edu.tw. Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan

§iwama@ie.nthu.edu.tw. Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 30013, Taiwan

¶csliao@ie.nthu.edu.tw. Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 30013, Taiwan

Suppose for each  $r_i$ ,  $t_i$  is an integer multiple of the travel time between location 0 and 1, i.e.,  $t_i = vt$  for some  $v \in \mathbb{N}$ . We assume that  $t_i - \tilde{t}_i$  is equal to a fixed value  $a$ , where  $a \geq t$  for all requests. Without loss of generality, assume  $a = t$ . Then we are only interested in a discrete-time stage, denoted by  $0, 1, 2, \dots$

Each server can only serve one request at a time. Serving a request yields a fixed positive profit  $y$ . A server used for a request with  $p_i = 0$  ( $p_i = 1$ , resp.) cannot be used for a request with  $p_{i+1} = 0$  ( $p_{i+1} = 1$ , resp.) in the next stage. We allow *empty movements*, i.e., a server can be moved from one location to the other without serving any request. An empty movement spends time  $t$ , but takes no cost. The goal of the  $kS2L$  problem is to maximize the total profit by serving a set of online requests.

## 2 Simultaneous Decision Model

Recall that in the  $kS2L$  model, two (or more) inputs with *the same booking time* still have an order. Thus we can equivalently think that if  $r_1, \dots, r_d$  are requests with booking time  $t$ , they are coming later than  $t - 1$  and before or at  $t$ , one by one. Each of them should get a decision (accept or reject) immediately before the next request. The adversary can change  $r_i$  after looking at the response of the online algorithm against  $r_1, \dots, r_{i-1}$ .

This setting sounds reasonable as an online model, but the following question seems also natural; what if requests with the same booking time come exactly at the same time, the online player can see all of them and can make decisions all together simultaneously at the booking moment (equivalently the requests arrive in the same fashion as above but the player can delay his/her online decisions until the booking moment). In this study we also consider this new model, denoted by  $kS2L-S$ . We further extend the model, assuming that the number of requests with the same booking time is at most  $Rk$  for some constant  $R$ . We call the generalized model  $RkS2L-S$ . Notice that having more than  $k$  requests at the same location with the same booking time never helps. Therefore, we only need to study the range  $0 \leq R \leq 2$  and  $kS2L-S$  means the special case that  $R = 2$ . Our algorithm for  $RkS2L-S$  is *adaptive* in the sense that it automatically accommodates the value of  $R$ , which does not have to be known in advance.

The tight competitive ratio (CR) of  $kS2L-F$  is 1.5 for  $k = 3i$  ( $i \in \mathbb{N}$ ) [3] and 2 for  $k = 2$  [2], but open for other  $k$ 's. In this study, we show that it is  $\frac{2k}{k + \lceil k/3 \rceil}$  for all  $k \geq 2$  and 1.5 for all  $k \geq 2$  if randomization is allowed. For  $kS2L-S$  that allows the online player to delay its decision, it is shown that we can indeed take this advantage. Namely the tight CR for  $kS2L-S$  is  $\frac{2k}{k + \lceil k/2 \rceil}$  for all  $k \geq 2$  and  $4/3$  for all  $k \geq 2$  if randomization is allowed. For  $RkS2L-S$  (we can assume  $1 \leq R \leq 2$  without loss of generality), it is shown that the CR is strictly improved if  $R < 2$ , namely the tight CR (for randomized algorithms) is improved to  $(2 + R)/3$ . Note that if  $R = 1.1$  (the number of requests at each stage exceeds  $k$  by at most 10%), the CR becomes at most 1.034.

The basic idea of our algorithms is “greedy” and “balanced”. Both notions have

already appeared in [3], but our implementation of them is significantly different from theirs. More importantly, our analysis is completely new; namely we use a simple mathematical induction (augmented by two interesting parameters other than the profit itself) while a classification of request types was used in [3].

### 3 Main Results

**Theorem 1** *GBA is a  $1/\delta$ -competitive algorithm for  $kS2L-S$  for any  $k \geq 2$ , where  $\delta = \frac{k + \lfloor k/2 \rfloor}{2k}$ .*

Note that we use ALG to denote an online algorithm and OPT an offline optimal algorithm in general in the rest of this study. As seen in GBA, a dangerous situation for the online player is that ALG accepts too many requests of one direction when it is possible. If ALG knows the total number of requests in each direction in advance, we can avoid this situation rather easily. Now we discuss  $kS2L-F$ , in which ALG does not know the total number of requests in advance. A simple and apparent solution is to stop accepting requests of one direction when its number gets to some value, even if more requests of that direction are coming and could be accepted. In the next algorithm, ARGBA, we set this value as  $2k/3$ . It then turns out, a little surprisingly, that the analysis for Theorem 1 is also available for the new algorithm almost as it is.

**Theorem 2** *ARGBA is a  $1/\delta$ -competitive algorithm for  $kS2L-F$  for any  $k \geq 2$ , where  $\delta = \frac{k + \lfloor k/3 \rfloor}{2k}$ .*

Notice that the CR of GBA is 2 when  $k = 2$ . The reason is simple, i.e., the existence of the ceiling function, namely if we can accept a fractional request, our CR would be  $4/3$ . Of course it is impossible to accept a request by one third, but it is possible to accept that request with probability  $1/3$ , which has the same effect as accepting it by one third in terms of an expected number.

**Theorem 3** *PrGBA is a  $4/3$ -competitive algorithm for  $kS2L-S$  for any  $k \geq 2$ .*

**Theorem 4** *PrARGBA is a  $1.5$ -competitive algorithm for  $kS2L-F$  for any  $k \geq 2$ .*

#### 3.1 Adaptive GBA

$kS2L-S$  gives the online player the advantage of an advance knowledge of the number of requests in the current stage. GBA does exploit this advantage, but not fully. Suppose  $(I\ell_1, Ir_1) = (50, 100)$ . GBA accepts the same number, 50, of  $(0,1)$ 's and  $(1,0)$ 's in stage 1. Then the adversary sends  $(I\ell_2, Ir_2) = (100, 0)$ , resulting in that only 50  $(0,1)$ 's can be accepted by GBA in stage 2, but 100  $(0,1)$ 's by OPT which could accept 100  $(1,0)$ 's in stage 1. Thus the CR in these two steps is  $4/3$ .

Now what about accepting roughly 28.57 (0,1)'s and 71.43 (1,0)'s in stage 1 (recall we can handle fractional numbers due to randomized rounding)? Then the best the adversary can do is to provide  $(Il_2, Ir_2) = (100, 0)$  or  $(0, 50)$ , in both of which the CR is  $200/171.43 \approx 150/128.57 \approx 1.17$ , significantly better than 1.5 of GBA. This is the basic idea of our new algorithm, AGBA. These key values 28.57 and 71.43 are denoted by  $\alpha_i$  and  $\beta_i$ , respectively. The ultimate goal of AGBA is to accept exactly  $\alpha_i$  (0,1)'s and  $\beta_i$  (1,0)'s while the ultimate goal of GBA was to accept  $k/2$  (0,1)'s and  $k/2$  (1,0)'s. If this goal is unachievable, to be figured out from the values of  $Il_i, Ir_i, Gl_{i-1}, Gr_{i-1}$  and  $Gf_{i-1}$ , both algorithms simply turn greedy.

**Theorem 5** *Suppose the number of requests is limited to at most  $Rk$  in all stages for some  $R$  such that  $1 \leq R \leq 2$  and  $Rk$  is an integer. Then AGBA solves  $RkS2L-S$  and is  $1/\delta$ -competitive, where  $\delta = 3/(2 + R)$ .*

## References

- [1] Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-sharing between two locations: Online scheduling with flexible advance bookings. In *24th International Computing and Combinatorics Conference (COCOON'18)*, volume 10976 of *LNCS*, pages 242–254, 2018.
- [2] Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-Sharing between Two Locations: Online Scheduling with Two Servers. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS'18)*, volume 117 of *LIPICs*, pages 50:1–50:14, 2018.
- [3] Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Online Scheduling of Car-Sharing Requests Between Two Locations with Many Cars and Flexible Advance Bookings. In *29th International Symposium on Algorithms and Computation (ISAAC'18)*, volume 123 of *LIPICs*, pages 64:1–64:13, 2018.
- [4] Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-Sharing on a Star Network: On-Line Scheduling with  $k$  Servers. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS'19)*, volume 126 of *LIPICs*, pages 51:1–51:14, 2019.

## SPT optimality via linear programming

Woo-Hyung Cho (Speaker) \*    David Shmoys †    Shane Henderson ‡

---

One of the oldest results in scheduling theory is that the Shortest Processing Time (SPT) rule finds an optimal solution to the problem of scheduling jobs on identical parallel machines to minimize the sum of job completion times [2], which is denoted  $P||\sum C_j$  in the notation of Graham et al. [5]. More precisely, there is a set  $\mathcal{N}$  of  $n$  jobs to be scheduled on  $m$  identical parallel machines. Each job  $j \in \mathcal{N}$  requires  $p_j$  units in uninterrupted processing time, and each machine can only process one job at a time. The goal is to find a nonpreemptive schedule that minimizes  $\sum_{j \in \mathcal{N}} C_j$ . The SPT rule works as follows: whenever a machine  $i$  is idle, begin processing a job  $j$  not yet processed with the shortest processing time  $p_j$ .

We present a new proof of correctness of SPT via linear programming (LP). We use an LP formulation that generalizes earlier works of Wolsey [11] and Queyranne [9]. Earlier proofs of correctness of SPT rely on coefficient matching (see [1, 8], for example), but to the best of our knowledge, this is the first LP-based proof.

Our proof relies on a generalization of a single-machine result that yields an equivalence between  $P||\sum C_j$  and  $P|p_j = p|\sum w_j C_j$ . For the latter problem, consider the linear program

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{N}} w_j C_j \\ \text{s.t.} \quad & \sum_{j \in \mathcal{S}} C_j \geq f(\mathcal{S}) \quad \text{for all } \mathcal{S} \subseteq \mathcal{N}, \end{aligned}$$

where  $f(\mathcal{S}) = \frac{p}{2} \left( \left\lceil \frac{|\mathcal{S}|}{m} \right\rceil^2 \cdot (|\mathcal{S}| \bmod m) + \left\lfloor \frac{|\mathcal{S}|}{m} \right\rfloor^2 \cdot (m - |\mathcal{S}| \bmod m) + |\mathcal{S}| \right)$ . Let  $\mathcal{P}$  denote the feasible region of this LP.

The linear inequalities defining  $\mathcal{P}$  tighten the valid inequalities for  $P||\sum w_j C_j$  derived by Schulz [10] and Hall et al. [6] in the special case where all jobs have equal processing time requirements. In fact, this strengthened class of inequalities gives a complete characterization of the scheduling polyhedron for  $P|p_j = p|\sum w_j C_j$ , which is a key element of our main result.

---

\*wc563@cornell.edu. School of Operations Research and Information Engineering, Cornell University, Ithaca, NY, USA.

†david.shmoys@cornell.edu. School of Operations Research and Information Engineering, Cornell University, Ithaca, NY, USA.

‡sgh9@cornell.edu. School of Operations Research and Information Engineering, Cornell University, Ithaca, NY, USA.

**Theorem 1**  $\mathcal{P}$  is a supermodular polyhedron with integer vertices that completely describes the scheduling polyhedron for  $P|p_j = p|\sum w_j C_j$ .

Suppose without loss of generality that jobs are sorted in nonincreasing order of weights  $w_1 \geq w_2 \geq \dots \geq w_n$ . Supermodularity of  $\mathcal{P}$  implies that a greedy algorithm can be used to obtain an optimal solution to the LP by letting

$$\begin{aligned} C_j &= f(\{1, \dots, j\}) - f(\{1, \dots, j-1\}) \\ &= p \left( \left\lfloor \frac{j-1}{m} \right\rfloor + 1 \right) \end{aligned}$$

for each  $j \in \mathcal{N}$ . That is, in an optimal solution, jobs are processed in sorted order whenever a machine becomes idle. In the absence of release dates, job completion times are always at integer multiples of  $p$ , so  $C$  is indeed a completion time vector in the scheduling polyhedron.

To complete our proof of SPT optimality, we observe that there is a natural generalization of a well-known equivalence between an input for  $1||\sum w_j C_j$  and what we shall call a *flipped input*, where  $p_j$  and  $w_j$  are interchanged for every job  $j$  and the order of jobs is also reversed (see, e.g., [4, 8, 3]). A flipped input for parallel machines can be generated in an analogous manner. The equal-weighted nature of our particular problem  $P||\sum C_j$  with general processing times creates a flipped instance with equal processing times and general weights, i.e., an input for  $P|p_j = p|\sum w_j C_j$ . Then, as a consequence of Theorem 1, an optimal solution to  $P||\sum C_j$  processes jobs sorted in *nondecreasing* order of *processing times* such that  $p_1 \leq p_2 \leq \dots \leq p_n$ . This proves SPT optimality.

**Extensions** Our methods and the geometric insights therein may find further uses; to demonstrate this, we apply the same principles in two generalized settings. Both problems discussed below are polynomial-time solvable via coefficient matching (see, e.g., [1, 7, 8]).

One simple extension is for the problem  $Q||\sum C_j$  with *uniformly-related* machines, where each machine  $i$  has speed  $s_i$  and so processing job  $j$  on machine  $i$  takes  $p_j/s_i$  time units. A polyhedral approach similar to Theorem 1 can be used to solve  $Q|p_j = p|\sum w_j C_j$ , where, in an optimal solution, jobs are sorted in nonincreasing order of weights  $w_1 \geq w_2 \geq \dots \geq w_n$  and processed in that order whenever a machine becomes idle. Letting  $t_1 \leq t_2 \leq \dots \leq t_n$  be the  $n$  smallest numbers in the following multiset of possible job completion times

$$\left\{ \frac{p}{s_1}, \dots, \frac{p}{s_m}, \frac{2p}{s_1}, \dots, \frac{2p}{s_m}, \dots, \frac{np}{s_1}, \dots, \frac{np}{s_m} \right\},$$

an optimal schedule processes job  $k$  for completion at time  $t_k$ . When  $t_k$  takes the form  $t_k = \ell p/s_i$ , job  $k$  is the  $\ell$ th job scheduled on a machine with speed  $s_i$  in  $Q|p_j = p|\sum w_j C_j$ . By the equivalence created by flipped inputs, job  $k$  is the  $\ell$ th *last* job scheduled on machine  $i$  in our original problem  $Q||\sum C_j$ .

Our techniques go beyond offering simpler, geometric explanations for known results. We give a new result on SPT optimality for a generalization of  $P||\sum C_j$  in which each job  $j$  may only be processed on some subset of machines  $\mathcal{M}_j$ . This problem with machine eligibility constraints is denoted  $P|\mathcal{M}_j|\sum C_j$  (or sometimes as  $R|p_{ij} \in \{p_j, \infty\}|\sum C_j$ ). Suppose we have a highly structured input in which machine eligibility sets  $\mathcal{M}_j$  are *nested* and jobs are sorted such that  $|\mathcal{M}_1| \leq |\mathcal{M}_2| \leq \dots \leq |\mathcal{M}_n|$ . Let us further assume that  $w_1 \leq w_2 \leq \dots \leq w_n$  holds. We are able to show that, in an optimal solution to  $P|\mathcal{M}_j, p_j = p|\sum w_j C_j$ , job  $j$  chooses an eligible machine  $i \in \mathcal{M}_j$  with the *smallest total job weight*, then is inserted into the front of the schedule on that machine. By constructing flipped inputs, we recover our original problem  $P|\mathcal{M}_j|\sum C_j$  and process jobs in sorted order on an eligible machine with the *shortest total processing time*.

## References

- [1] P. BRUCKER, *Scheduling Algorithms*, Springer Berlin Heidelberg, 2007.
- [2] R. W. CONWAY, W. L. MAXWELL, AND L. W. MILLER, *Theory of Scheduling*, Addison-Wesley Publishing Company, 1967.
- [3] W. L. EASTMAN, S. EVEN, AND I. M. ISAACS, *Bounds for the optimal scheduling of  $n$  jobs on  $m$  processors*, Management Science, 11 (1964), pp. 268–279.
- [4] M. X. GOEMANS AND D. P. WILLIAMSON, *Two-dimensional Gantt charts and a scheduling algorithm of Lawler*, SIAM Journal on Discrete Mathematics, 13 (2000), pp. 281–294.
- [5] 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*, in Discrete Optimization II, vol. 5 of Annals of Discrete Mathematics, Elsevier, 1979, pp. 287–326.
- [6] L. A. HALL, A. S. SCHULZ, D. B. SHMOYS, AND J. WEIN, *Scheduling to minimize average completion time: Off-line and on-line approximation algorithms*, Mathematics of Operations Research, 22 (1997), pp. 513–544.
- [7] E. L. LAWLER, J. K. LENSTRA, A. H. G. RINNOOY KAN, AND D. B. SHMOYS, *Chapter 9 Sequencing and scheduling: Algorithms and complexity*, in Logistics of Production and Inventory, vol. 4 of Handbooks in Operations Research and Management Science, Elsevier, 1993, pp. 445–522.
- [8] J. K. LENSTRA AND D. B. SHMOYS, *Elements of scheduling*, CoRR, arXiv: <https://arxiv.org/abs/2001.06005> (2020).
- [9] M. QUEYRANNE, *Structure of a simple scheduling polyhedron*, Mathematical Programming, 58 (1993), pp. 263–285.

- [10] A. S. SCHULZ, *Polytopes and Scheduling*, PhD thesis, TU Berlin, 1996.
- [11] L. A. WOLSEY, *Mixed integer programming formulations for production planning and scheduling problems*, in 12th International Symposium on Mathematical Programming, 1985.

# Polynomial-Size ILP formulations for the Total Completion Time Problem on a Parallel Batching Machine

Alessandro Druetto (Speaker) <sup>\*†</sup>      Andrea Grosso <sup>\*‡</sup>

---

## 1 Introduction

We present polynomial-size arc-flow models with related lower bounds and heuristics for the total completion time parallel batching problem.

A set of jobs  $j \in N$ ,  $N = \{1, 2, \dots, n\}$ , with processing times  $p_j$  and sizes  $s_j$ , have to be processed on a parallel batching machine with a given limited capacity  $b$ . Jobs must be arranged into a batch sequence  $S = (B_1, B_2, \dots, B_t)$  in order to minimize the total completion time  $f(S) = \sum_{j \in N} C_j$  of all jobs. All the jobs in the same batch  $B_k$  are processed in parallel and all share the batch completion time  $C_{B_k}$ :  $C_j = C_{B_k}$  for all  $j \in B_k$ . The longest job in a batch  $B_k$  determines the batch processing time: the whole batch has a duration of  $p_{B_k} = \max\{p_j : j \in B_k\}$ . The total size of the jobs in a batch cannot exceed the machine capacity:  $\sum_{j \in B_k} s_j \leq b$ . In the classical three-fields notation the problem is denoted  $1|p\text{-batch}, s_j|\sum_j C_j$ .

Parallel batching problems often arise in the management of furnaces-like facilities, notably in the semiconductor industry. See [3] for a survey. The  $1|p\text{-batch}, s_j|\sum_j C_j$  problem has been first tackled in [1], where it is recognized as NP-hard and a first exact branch-and-bound algorithm is sketched. In [2] an improved branch and bound is proposed, while a recent metaheuristic approach, via ant-colony optimization, is proposed in [4]. Exact and heuristic approaches based on very large Integer Linear Programming formulations and column generation techniques are proposed in [6] and in [7].

## 2 Arc-flow models

Arc-flow models for parallel batching problems are a recent development where batch schedules are mapped on flow configurations on a suitable network. In [5] an arc-flow model for minimizing makespan in a single parallel batching machine

---

\*Dipartimento di Informatica, Università di Torino, via Pessinetto 12, 10149 Torino, Italy

†alessandro.druetto@unito.it

‡andrea.grosso@unito.it

is proposed, with very good results. In [6] an arc-flow model is proposed for the  $1|p\text{-batch}, s_j|\sum_j C_j$  problem, leading to handle 100-jobs instances. This arc-flow model is exponential in size and must be handled via column generation techniques; here we develop, from the same modeling ideas, arc-flow models whose size is polynomial in the number on jobs  $n$ .

For arc-flow representations of  $1|p\text{-batch}, s_j|\sum_j C_j$ , we recall the key modeling idea from [6]. Consider a graph  $G(V, A)$  with node set  $V = \{1, 2, \dots, n + 1\}$  and arc sets  $A = \{(i, k) : i, k \in V; i < k\}$ ; the structure of a batch sequence  $S$  can be mapped onto a path  $P_S$  from node 1 to node  $n + 1$  in the graph  $G$ . Each arc  $(i, k) \in P_S$  corresponds to a batch  $B_{ik}$  in the sequence where exactly  $k - i$  jobs are processed, with  $i - 1$  jobs processed in the preceding batches; by suitably partitioning the job set  $N$  on the arcs of  $P_S$ , such that  $|B_{ik}| = k - i$  and  $\sum_{j \in B_{ik}} s_j \leq b$ , the path  $P_S$  correctly represents a feasible batch sequence. Then, by assigning to each arc  $(i, k)$  in  $P_S$  the cost  $c_{ik} = p_{B_{ik}}(n - i + 1) = \max\{p_j : j \in B_{ik}\}(n - i + 1)$ , the total completion time of the batch sequence  $S$  (refer to [6] for the proof of a very similar result) equals the computed cost of path  $P_S$ :  $f(S) = \sum_{j \in N} C_j = \sum_{(i,k) \in P_S} c_{ik} = c(P_S)$ .

### 3 A polynomial-size flow-based model

$$\text{minimize } \sum_{\substack{(i,k) \in A \\ t \in \mathcal{T}}} c_{ikt} x_{ikt} \quad (1)$$

subject to

$$\sum_{\substack{(i,k) \in A \\ t \in \mathcal{T}}} x_{ikt} - \sum_{\substack{(k,i) \in A \\ t \in \mathcal{T}}} x_{kit} = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \dots, n \\ -1 & i = n + 1 \end{cases} \quad (2)$$

$$\sum_{\substack{(i,k) \in A \\ t \in \mathcal{T}}} y_{jikt} = 1 \quad \forall j \in N \quad (3)$$

$$\sum_{j \in N} s_j y_{jikt} \leq b x_{ikt} \quad \forall (i, k) \in A, t \in \mathcal{T} \quad (4)$$

$$\sum_{j \in N} y_{jikt} \leq (k - i) x_{ikt} \quad \forall (i, k) \in A, t \in \mathcal{T} \quad (5)$$

$$y_{jikt} \leq x_{ikt} \quad \forall j \in N, (i, k) \in A, t \in \mathcal{T} \quad (6)$$

$$x_{ikt}, y_{jikt} \in \{0, 1\} \quad \forall j \in N, (i, k) \in A, t \in \mathcal{T} \quad (7)$$

We formulate an ILP model that calls for finding a minimum-cost path  $P_S$  from node 1 to node  $n + 1$  on graph  $G$ , partitioning jobs on the arcs of  $P_S$ . Let

$\mathcal{T} = \{p_j : j \in N\}$  be the set of all processing times listed in the considered problem instance. We use a set of decision variables  $x_{ikt}$ , where  $x_{ikt} = 1$  iff an arc  $(i, k)$  is in the paths and the corresponding batch  $B_{ik}$  has processing time  $p_{B_{ik}} = t \in \mathcal{T}$ . Another set of decision variables  $y_{jikt}$  is defined, with  $y_{jikt} = 1$  iff job  $j$  is in the batch  $B_{ik}$  and the latter has batch processing time  $p_{B_{ik}} = t$ . The cost  $c_{ikt}$  for an arc whose batch  $B_{ik}$  has processing time  $t$  is defined as  $c_{ikt} = t(n - i + 1)$ .

Objective function (1) requires to minimize the sum of total completion time for all selected arcs (=batches). Constraints in (2) represents a classical flow model with unitary flow, from source 1 to destination  $n + 1$ , counting as positive the flow to outgoing arcs and as negative the flow from incoming arcs. Exact partitioning for all jobs is enforced in (3), and capacity constraints (4) are defined for all selected arcs.

## 4 Computational experience

Model (1)–(7) requires  $\mathcal{O}(n^4)$  variables and constraints; although polynomial in size, computational experience shows that it still has limitations. The lower bound delivered by the continuous relaxation of (1)–(7) is excellent, but on larger instances the time required for CPLEX to solve the problem is quite large.

Computational experience shows that completely relaxing constraints (6) does not impact substantially the quality of the lower bound, while improving a lot CPLEX computational times, sensibly reducing the number of rows to be handled by the solver. Still the number of variables is quite high, and we decided to develop a column generation approach to avoid having all variables in the problem from the start. Column generation applied to a polynomial number of columns is often referred to as *sifting* in literature.

A variable rounding strategy is developed in order to build a heuristic integer solution exploiting the sifted columns.

Computational results — to be presented at the conference — show that this approach is quite fast and delivers high quality solutions with narrow, certified optimality gaps.

## References

- [1] UZSOY, R. (1994). *Scheduling a single batch processing machine with non-identical job sizes*. International Journal Of Production Research.
- [2] AZIZOGLU, M. AND WEBSTER, S. (2000). *Scheduling a batch processing machine with non-identical job sizes*. International Journal Of Production Research.
- [3] MÖNCH, L., FOWLER, J., DAUZÈRE-PÉRÈS, S., MASON, S. AND ROSE, O. (2011). *A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations*. Journal Of Scheduling.

- [4] PARSA, N., KARIMI, B. AND HUSSEINI, S. (2016). *Minimizing total flow time on a batch processing machine using a hybrid max–min ant system*. Computers & Industrial Engineering.
- [5] TRINDADE, R., DE ARAÚJO, O. AND FAMPA, M. (2021). *Arc-flow approach for single batch-processing machine scheduling*. Computers & Operations Research.
- [6] ALFIERI, A., DRUETTO, A., GROSSO, A. AND SALASSA, F. (2021). *Column generation for minimizing total completion time in a parallel-batching environment*. Journal Of Scheduling.
- [7] DRUETTO, A. AND GROSSO, A. (2022). *Column generation and rounding heuristics for minimizing the total weighted completion time on a single batching machine*. Computers & Operations Research.

# A Competitive Algorithm for Throughout Maximization on Identical Machines

Benjamin Moseley\*

Kirk Pruhs (Speaker)<sup>†</sup>

Clifford Stein<sup>‡</sup>

Rudy Zhou\*

---

## 1 Introduction

We consider the basic problem of preemptively scheduling jobs that arrive online with sizes and deadlines on  $m$  identical machines to maximize the number of jobs that complete by their deadline.

Formally, let  $J$  be a collection of jobs such that each  $j \in J$  has a *release time*  $r_j$ , a *processing time* (or *size*)  $x_j$ , and a *deadline*  $d_j$ . The jobs arrive online at their release times, at which the scheduler becomes aware of job  $j$  and its  $x_j$  and  $d_j$ . At each moment of time, the scheduler can specify up to  $m$  released jobs to run at this time, and the remaining processing time of the jobs that are run is decreased at a unit rate (so we assume that the online scheduler is allowed to produce a migratory schedule.) A job is completed if its remaining processing time drops to zero by the deadline of that job. The objective is to maximize the number completed jobs. A key concept in designing algorithms for this problem is the *laxity* of a job. The *laxity* of a job  $j$  is  $\ell_j = (d_j - r_j) - x_j$ , which is the maximum amount of time we can not run  $j$  and still possibly complete it. We measure the performance of our algorithm by the *competitive ratio*, which is the maximum over all instances of the ratio of the objective value of our algorithm to the objective value of the optimal offline schedule that is aware of all jobs in advance.

This problem is well understood for the  $m = 1$  machine case. No  $O(1)$ -competitive deterministic algorithm is possible [BKM<sup>+</sup>92], but there is a randomized algorithm that is  $O(1)$ -competitive against an oblivious adversary [KP03], and there is a scalable ( $O(1 + \epsilon)$ -speed  $O(1/\epsilon)$ -competitive) deterministic algorithm [KP00]. The scalability result in [KP00] was extended to the case of  $m > 1$

---

\*Tepper School of Business, Carnegie Mellon University. B. Moseley and R. Zhou were supported in part by NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909. Benjamin Moseley was additionally supported in part by a Google Research Award, an Infor Research Award, and a Carnegie Bosch Junior Faculty Chair.

<sup>†</sup>Computer Science Department, University of Pittsburgh. Supported in part by NSF grants CCF-1907673, CCF-2036077 and an IBM Faculty Award.

<sup>‡</sup>Industrial Engineering and Operations Research, Columbia University. Research partly supported by NSF Grants CCF-1714818 and CCF-1822809.

machines in [LMNY13]. Whether an  $O(1)$ -competitive algorithm exists for  $m > 1$  machines has been open for twenty years. Previous results for the multiple machines setting require resource augmentation or assume that all jobs have high laxity [LMNY13, EMS20].

The main issue in removing these assumptions is how to determine which machine to assign a job to. If an online algorithm could determine which machine each job was assigned to in Opt, we could obtain an  $O(1)$ -competitive algorithm for  $m > 1$  machines by a relatively straight-forward adaptation of the results from [KP03]. However, if the online algorithm ends up assigning some jobs to different machines than Opt, then comparing the number of completed jobs is challenging. Further, if jobs have small laxity, then the algorithm can be severely penalized for small mistakes in this assignment. One way to view the speed augmentation (or high laxity assumption) analyses in [LMNY13, EMS20] is that the speed augmentation assumption allows one to avoid having to address this issue in the analyses.

*Our main result is an  $O(1)$ -competitive deterministic algorithm for Throughput Maximization on  $m > 1$  machines.*

A simple consequence of the results in [KP01] and [KP03] is an  $O(1)$ -competitive algorithm in the case that  $m = O(1)$ . Thus we concentrate on the case that  $m$  is large. Also note that since there is an  $O(1)$ -approximate non-migratory schedule [KP01], changing the number of machines by an  $O(1)$  factor does not change the optimal objective value by more than an  $O(1)$  factor. These observations about the structure of near-optimal schedules allow us to design a  $O(1)$ -competitive algorithm that is a combination of various deterministic algorithms. In particular, on instance  $J$  our algorithm, FINALALG will run a deterministic algorithm LMNY on  $m/3$  machines on the subinstance  $J_{hi} = \{j \in J \mid \ell_j > x_j\}$  of high laxity jobs, a deterministic algorithm SRPT on  $m/3$  machines on the subinstance  $J_{lo} = \{j \in J \mid \ell_j \leq x_j\}$  of low laxity jobs, and a deterministic algorithm MLAX on  $m/3$  machines also on the subinstance  $J_{lo}$  of low laxity jobs. Roughly, each of the three algorithms is responsible for a different part of Opt.

The critical component our result is the design of MLAX, which intuitively we want to be competitive on low-laxity jobs in Opt that are preempted. But before describing the algorithm MLAX, we need to make some definitions. Let  $\alpha = O(1)$  be a sufficiently large constant. MLAX maintains  $m$  stacks (last-in-first-out data structures) of jobs (one per machine),  $H_1, \dots, H_m$ . The stacks are initially empty. At all times, MLAX runs the top job of stack  $H_i$  on machine  $i$ . We define the *frontier*  $F$  to be the set consisting of the top job of each stack (i.e. all currently running jobs.) The pseudo-release time (if it exists)  $\tilde{r}_j$  of job  $j$  is the earliest time in  $[r_j, r_j + \frac{\ell_j}{2}]$  such that there are at least  $\frac{7}{8}m$  jobs  $j'$  on the frontier satisfying  $\alpha x_{j'} \geq \ell_j$ . We say a job  $j$  is *viable* if  $\tilde{r}_j$  exists and *non-viable* otherwise.

There are two types of events that cause MLAX to update the  $H_i$ 's: reaching a job's pseudo-release time or completing a job. At job  $j$ 's pseudo-release time (note  $\tilde{r}_j$  can be determined online by MLAX), MLAX does the following:

1. If there exists a stack whose top job  $j'$  satisfies  $\alpha x_{j'} \leq \ell_j$ , then *push*  $j$  onto any such stack.

2. Else if there exist at least  $\frac{3}{4}m$  stacks whose second-top job  $j''$  satisfies  $\alpha x_j \leq \ell_{j''}$  and further some such stack has top job  $j'$  satisfying  $\ell_j > \ell_{j'}$ , then on such a stack with minimum  $\ell_{j'}$ , *replace* its top job  $j'$  by  $j$ .

When MLAX completes a job  $j$  that was on stack  $H_i$ , MLAX does the following:

- c) *Pop*  $j$  off of stack  $H_i$ .
- d) Keep *popping*  $H_i$  until the top job of  $H_i$  is feasible.

There are three main steps in showing that FINALALG is  $O(1)$ -competitive. The first step is to show how to modify the optimal schedule to obtain certain structural properties that facilitate the comparison with SRPT and MLAX. The second step is to show that SRPT is competitive with the low-laxity, non-viable jobs. Intuitively, the jobs that MLAX is running that prevent a job  $j$  from becoming viable are so much smaller than job  $j$ , and they provide a witness that SRPT must also be working on jobs much smaller than  $j$ . The third step is to show that SRPT and MLAX together are competitive with the low-laxity, viable jobs. First, we show that SRPT is competitive with the number of non-preempted jobs in Opt. We then essentially show that MLAX is competitive with the number of preempted jobs in Opt. The key component in the design of MLAX is the condition that a job  $j$  won't replace a job on the frontier unless there are at least  $\frac{3}{4}m$  stacks whose second-top job  $j''$  satisfies  $\alpha x_j \leq \ell_{j''}$ . This is the condition that intuitively most differentiates the MLAX from  $m$  copies of the LAX algorithm in [KP03]. This also is the condition that allows us to surmount the issue of potentially assigning a job to a “wrong” processor. Jobs that satisfy this condition are highly flexible about where they can go on the frontier.

## References

- [BKM<sup>+</sup>92] Sanjoy K. Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis E. Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real Time Systems*, 4(2):125–144, 1992.
- [EMS20] Franziska Eberle, Nicole Megow, and Kevin Schewior. Optimally handling commitment issues in online throughput maximization. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *European Symposium on Algorithms*, volume 173 of *LIPICs*, pages 41:1–41:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- [KP00] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000. Also 1995 Symposium on Foundations of Computer Science.
- [KP01] Bala Kalyanasundaram and Kirk Pruhs. Eliminating migration in multi-processor scheduling. *Journal of Algorithms*, 38(1):2–24, 2001.

- [KP03] Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. *Journal of Algorithms*, 49(1):63–85, 2003. Also 1998 European Symposium on Algorithms.
- [LMNY13] Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs. In Guy E. Blelloch and Berthold Vöcking, editors, *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 305–314. ACM, 2013.

# Scheduling with Machine Conflicts

Moritz Buchem (Speaker) \*      Linda Kleist †  
Daniel Schmidt genannt Waldschmidt ‡

---

## 1 Introduction

We consider a variant of the well-studied scheduling problem of makespan minimization when machine conflicts are taken into account. We are particularly interested in the situation when external pre- and post-processing of jobs is necessary before and after the job is internally processed by a machine. Such conflicts of pre- and post-processing may be due to shared resources or spatial constraints. Shared resources can be found in computing problems where processors may share different databases or external processors that must be accessed before and after executing tasks on the processor. These external resources can only be accessed by one processor at a time and different processors may share different external resources. An up-to-date example of spatial conflicts occurs in pandemics when schedulers are faced with potentially infectious jobs which should keep sufficient distance to each other, e.g., in testing or vaccination centers. Similarly, spatial conflicts play a crucial role when jobs may have private information or data that should not be shared; e.g., the interrogation of suspects in multiple rooms.

## 2 Problem Formulation and Related Work

SCHEDULINGWITHMACHINECONFLICTS (SMC) is a scheduling problem in which jobs on conflicting machines are processed such that certain blocking intervals of their processing time do not overlap. The objective is to minimize the makespan. An instance of SMC is defined by a set of jobs  $\mathcal{J}$  and a conflict graph  $G = (V, E)$  on a set of machines  $V$  where two machines  $i$  and  $i'$  are *in conflict* if and only if  $\{i, i'\} \in E$ . In contrast to classical scheduling problems, each job  $j$  has three parameters  $(b_j^1, p_j, b_j^2)$ , where  $b_j^1$  and  $b_j^2$  denote the first and second *blocking time* of  $j$ , respectively, and  $p_j$  denotes its *processing time*. Together they constitute the

---

\*m.buchem@maastrichtuniversity.nl. Dept. of Quantitative Economics, Maastricht University, P.O. Box 616, 6200 MD, Maastricht, The Netherlands

†kleist@ibr.cs.tu-bs.de. Institute of Operating Systems and Computer Networks, TU Braunschweig, Mühlenpfordtstrasse 23, 38106, Braunschweig, Germany

‡dschmidt@math.tu-berlin.de. Institute of Mathematics, TU Berlin, Straße des 17. Juni 136, 10623, Berlin, Germany

system time  $q_j = b_j^1 + p_j + b_j^2$ ; note that the order  $b_j^1, p_j, b_j^2$  must be maintained and jobs need to be scheduled non-preemptively. We seek schedules in which the blocking times of no two jobs on conflicting machines intersect. Formally, a (*conflict-free*) schedule  $\Pi$  is an assignment of jobs to machines and starting times such that

- for each point in time, every machine executes at most one job,
- for every edge  $\{i, i'\} \in E$  and two jobs  $j, j' \in \mathcal{J}$  assigned to machines  $i$  and  $i'$ , respectively, the intervals of the blocking times of  $j$  and  $j'$  do not overlap interiorly in time.

The classical scheduling problem of *makespan minimization on parallel identical machines* is well studied in the literature ranging from approximation results due to list scheduling algorithms [5, 6] to (efficient) polynomial time approximation schemes [7, 8]. Scheduling with job conflicts has been considered in various different settings (see e.g. [1, 2, 3]), whereas machine conflicts have not been taken into account in the literature. *Scheduling with pre- and post-processing* has been considered in different models in the literature such as the master-and-slave problem introduced by Kern and Nawijn [9] and termed by Sahni [11] or scheduling jobs with segmented self-suspension with a single suspension component introduced by Rajkumar et al. [10].

### 3 Our contribution

We first consider SMC with identical jobs (SMC-ID). Identifying intrinsic connections to maximum independent sets, we show that even when  $(b^1, p, b^2)$  are fixed positive parameters for any  $\epsilon > 0$  there is no  $\mathcal{O}(m^{1-\epsilon})$ -approximation for SMC-ID, unless  $P = NP$ . Complementary, we derive polynomial time approximation algorithms given a (collection of) maximum independent set(s). These yield an approximation guarantee better than 2 depending on the job parameters.

Furthermore, we observe that when any feasible schedule uses at most one independent set of machines at a time, the problem reduces to finding a maximum independent set and evenly distributing the jobs among these machines results in an optimal schedule. This relation motivates us to consider a special case of identical jobs for which in any feasible solution machines that process jobs in parallel form at most two independent sets: unit jobs where all job parameters are equal to 1. Since finding a maximum independent set and two independent sets of maximum size can be done in polynomial time on bipartite graphs, we closely investigate the case of unit jobs for bipartite conflict graphs. For this case we design a polynomial time algorithm to compute optimal schedules.

**Theorem 1** *For every bipartite graph  $G$  and every  $n$ , an optimal schedule for SMC with unit jobs can be computed in polynomial time.*

To obtain this result we develop a divide-and-conquer algorithm based on structural insights for stars. In particular, we show that optimal schedules on stars only use two different types of patterns. Either jobs are scheduled in parallel only on the leaves of a star or an additional job is scheduled on the center of a star in an intertwined manner. Using these structural insights we consider a spanning forest whose components are stars. The spanning forest is chosen in such a way that we can adjust an optimal solution with respect to the spanning forest such that it also gives an optimal solution with respect to the original graph. Specifically, the spanning forest allows to feasibly patch together optimal schedules on the star components to give a feasible schedule on the original graph. Finally, we complement these results by introducing a polynomial time algorithm to find the desired spanning forest. For more details, we refer to [4].

## References

- [1] Brenda S. Baker and Edward G. Coffman Jr. Mutual exclusion scheduling. *Theoretical Computer Science*, 162(2):225–243, 1996.
- [2] Hans L. Bodlaender and Klaus Jansen. On the complexity of scheduling incompatible jobs with unit-times. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pages 291–300. Springer, 1993.
- [3] Hans L. Bodlaender, Klaus Jansen, and Gerhard J. Woeginger. Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55(3):219–232, 1994.
- [4] Moritz Buchem, Linda Kleist, and Daniel Schmidt genannt Waldschmidt. Scheduling with machine conflicts. *CoRR*, abs/2102.08231, 2021.
- [5] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.
- [6] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.
- [7] Dorit S. Hochbaum. Various notions of approximations: Good, better, best and more. *Approximation algorithms for NP-hard problems*, 1997.
- [8] Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45(4):1371–1392, 2020.
- [9] Walter Kern and Wim N. Nawijn. Scheduling multi-operation jobs with time lags on a single machine. In *Proceedings of the 2nd Twente Workshop on Graphs and Combinatorial Optimization*, 1991.

- [10] Ragnathan Rajkumar, Lui Sha, and John P Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, volume 88, pages 259–269, 1988.
- [11] Sartaj K. Sahni. Scheduling master-slave multiprocessor systems. *IEEE transactions on Computers*, 45(10):1195–1199, 1996.

# Load Balancing: The Long Road from Theory to Practice\*

Sebastian Berndt <sup>†</sup>    Max A. Deppert (Speaker) <sup>‡</sup>    Klaus Jansen <sup>§</sup>  
Lars Rohwedder <sup>¶</sup>

---

## Introduction

Makespan minimization on identical parallel machines (often denoted by  $P||C_{\max}$ ) asks for a distribution of a set  $J$  of  $n = |J|$  jobs to  $m \leq n$  machines. Each job  $j \in J$  has a processing time  $p_j$  and the objective is to minimize the makespan, i.e., the maximum sum of processing times of jobs assigned to a single machine. More formally, a *schedule*  $\sigma: J \rightarrow \{1, \dots, m\}$  assigns jobs to machines. The load  $\ell_{\sigma,i}$  of machine  $i$  in schedule  $\sigma$  is defined as  $\sum_{j \in \sigma^{-1}(i)} p_j$  and the *makespan*  $\mu(\sigma) = \max_i \{\ell_{\sigma,i}\}$  is the maximal load. The goal is to find a schedule  $\sigma$  minimizing  $\mu(\sigma)$ . This is a widely studied problem both in operations research and in combinatorial optimization and has led to many new algorithmic techniques. For example, it has led to one of the earliest examples of an approximation scheme and the use of the dual approximation technique [5]. The problem is known to be strongly NP-hard and thus we cannot expect to find an exact solution in polynomial time. Many approximation algorithms that run in polynomial time and give a non-optimal solution have been proposed for this problem. From a theory point of view, the strongest approximation result is a polynomial time approximation scheme (PTAS) which gives a  $(1 + \varepsilon)$ -approximation, where the precision  $\varepsilon > 0$  can be chosen arbitrarily small and is given to the algorithm as input. This goes back to a seminal work by Hochbaum and Shmoys [5]. The running time of such schemes for  $P||C_{\max}$  were drastically improved over time [1, 6, 9] and the best known running time is  $2^{O(1/\varepsilon \log^2(1/\varepsilon))} \log(n) + O(n)$  due to Jansen and Rohwedder [8] (extension of [7] by the theory of discrepancy), which is subsequently called the JR-algorithm. The JR-algorithm is in fact an algorithm for integer programming, but gives this running time when applied to a natural formulation of  $P||C_{\max}$ . A PTAS with a running time of  $f(1/\varepsilon) \cdot n^{O(1)}$  like in the JR-algorithm is called

---

\*Published on ALENEX 2022 [3] and GitHub [2]. Supported by German Research Foundation (DFG) project JA 612/20-1

<sup>†</sup>s.berndt@uni-luebeck.de. University of Lübeck, Germany

<sup>‡</sup>made@informatik.uni-kiel.de. Kiel University, Germany

<sup>§</sup>kj@informatik.uni-kiel.de. Kiel University, Germany

<sup>¶</sup>l.rohwedder@maastrichtuniversity.nl. Maastricht University, Netherlands

an efficient polynomial time approximation scheme (EPTAS). It follows from the strong NP-hardness that no fully polynomial time approximation scheme (FPTAS), an approximation scheme polynomial in both  $n$  and  $1/\epsilon$ , exists unless  $P = NP$ . Furthermore, for any  $\delta > 0$ , there is no PTAS for  $P||C_{\max}$  in time  $2^{O((1/\epsilon)^{1-\delta})} + n^{O(1)}$ , unless the exponential time hypothesis (ETH) fails [4].

PTAS's are often believed to be impractical. They tend to yield extremely high (though polynomial) running time bounds even for moderate precisions  $\epsilon$ , see Marx [10]. By some, the research on PTAS's has even been considered damaging for the large gap between theory and practice that it creates [12]. Although EPTAS's (when FPTAS's are not available) are sometimes proposed as a potential solution for this situation [10], we are not aware of a practical implementation of an EPTAS. For example, an approximation scheme for euclidean TSP was implemented by Rodeker et al., but the algorithm was merely inspired by an EPTAS and it does not retain the theoretical guarantee [11]. Although this is an interesting research direction as well, it remains an intriguing question whether one can obtain a practically relevant EPTAS implementation with actual theoretical guarantees. On the one hand, we believe that this is an important question to ask concerning the relevance of such a major field of research. On the other hand, such a PTAS implementation has great advantages in itself, since it exhibits a clean and generic design that is not specific to any concrete precision, as well as a (theoretically) unlimited potential of the precision.

## Our Results

As a major milestone we obtain a generic PTAS implementation that achieves in reasonable time a precision which beats the best known guarantee of a polynomial time non-PTAS algorithm. This precision to the best of our knowledge is  $2/11 \approx 18.2\%$ , which is guaranteed by the MULTIFIT algorithm. The claim might appear vague, since the running time depends not only on  $\epsilon$ , but also on the instance. We believe that it is plausible nevertheless: The algorithm we use, which is based on the JR-algorithm, reduces the problem to performing  $O(\log(n))$  many fast Fourier transformations (FFTs), where the size of the FFT input depends only on  $\epsilon$  and not the instance itself. Hence, the running time for all instances (using the same precision) is very stable and predictable. This is in the spirit of an EPTAS running time. We successfully run experiments of our implementation for a precision of  $\epsilon < 2/11$  and thus make the claim that this precision is practically feasible in general. This is also the main message of our paper. For completeness, we provide comparisons of the solution quality obtained empirically. While the theoretical guarantee of the PTAS is better, the difference to non-PTAS algorithms is marginal at this state and it is not yet evident in the experiments. The execution of the PTAS is computationally expensive and the considered precision is on the edge of what is realistic for our implementation. However, we believe that further optimization or more computational resources can lead to also empirically superior results. Nevertheless, the successful execution with a low precision value forms a proof of concept for practical PTAS's.

Towards obtaining such an implementation we need to fine-tune the JR-algorithm significantly. In particular, it requires non-trivial theoretical work and novel algorithmic ideas. In fact, our variant has a slightly better dependence on the precision, namely  $2^{O(1/\epsilon \log(1/\epsilon) \log \log(1/\epsilon))}$ , giving the best known running time for this problem. Our approach also greatly reduces the constants hidden by the  $O$ -notation. We first construct an integer program (IP) — the well-known *configuration* IP — that implies a  $(1 + \epsilon)$ -approximation by rounding the processing times. This IP has properties that allow sophisticated algorithms to solve it efficiently. We present several reduction steps to simplify and compress the IP massively. As extensions of this configuration IP are widely used, we believe this to be of interest in itself. For the makespan minimization problem, we obtain an IP where the columns of the constraint matrix have  $\ell_\infty$ -norms bounded by 2 and  $\ell_1$ -norms bounded by  $O(\log(1/\epsilon))$ . In contrast, in the classical configuration integer program used in many of the previous PTAS's both of these norms are bounded by  $O(1/\epsilon)$ . This allows us to greatly reduce the size of the FFT instances in the JR-algorithm without losing the theoretical guarantee. For example, for  $\epsilon \approx 17.29\%$ , our reduced IP lowers the instance sizes for FFT from  $49^{12}$  words for the configuration IP to  $5^{12}$  words. Another important aspect in the algorithm is the rounding of the processing times. In general, one needs to consider only  $O(1/\epsilon \log(1/\epsilon))$  different rounded processing times (to guarantee a precision of  $\epsilon$ ). This number has great impact on the size of the FFT instances. For concrete  $\epsilon$  the general rounding scheme might not give the optimal number of rounded processing times. We present a mixed integer linear program that can be used to generically optimize the rounding scheme for guaranteeing a fixed precision  $\epsilon$  (or equivalently, for a fixed number of rounded processing times).

## References

- [1] Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.
- [2] Sebastian Berndt, Max A. Deppert, Klaus Jansen, and Lars Rohwedder. Implementation of the BDJR algorithm. <https://github.com/made4this/BDJR>, 2021.
- [3] Sebastian Berndt, Max A. Deppert, Klaus Jansen, and Lars Rohwedder. Load balancing: The long road from theory to practice. In *Proc. ALENEX 2022*, 2022.
- [4] Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *J. Comput. Syst. Sci.*, 96:1–32, 2018.

- [5] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987.
- [6] Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Math. Oper. Res.*, 45(4):1371–1392, 2020. doi:10.1287/moor.2019.1036.
- [7] Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *Proc. ITCS 2019*, pages 43:1–43:17, 2019.
- [8] Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution. 2019. Extension of [7] by the theory of discrepancy. arXiv:1803.04744v3.
- [9] Joseph Y.-T. Leung. Bin packing with restricted piece sizes. *Inf. Process. Lett.*, 31(3):145–149, 1989. doi:10.1016/0020-0190(89)90223-8.
- [10] Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.
- [11] Bárbara Rodeker, M. Virginia Cifuentes, and Liliana Favre. An empirical analysis of approximation algorithms for euclidean TSP. In *Proc. CSC 2009*, pages 190–196. CSREA Press, 2009.
- [12] Frances Rosamond. Parameterized complexity-news. *The Newsletter of the Parameterized Complexity Community Volume*, 2006.

# Rescheduling with new orders under a maximum completion time disruption constraint

Stefan Lendl <sup>\*</sup>    Ulrich Pferschy <sup>†</sup>    Elena Renner (Speaker) <sup>‡</sup>  
Fabio Salassa <sup>§</sup>    Vincent T'kindt <sup>¶</sup>

---

## 1 Introduction

Rescheduling problems may arise after a scheduling phase when some unpredicted event occurs, like the arrival of new orders or jobs' delays, that affects the original schedule. A solution sought is one that is able to answer both to the optimization of an objective function and to a limited amount of change of the original schedule in order to be somehow compliant to the previous plan.

We consider the rescheduling problem first introduced by [1], where an initial set of jobs has to be rescheduled when new jobs arrive. Each job belongs either to the old set  $J^O = \{1, \dots, n_O\}$  or the new set  $J^N = \{1, \dots, n_N\}$  and has a processing time  $p_j$ , a due date  $d_j$ , and a weight  $w_j \forall j \in J^O \cup J^N$ . The old jobs are assumed to be optimally ordered in the original schedule  $\pi$ . We refer to their completion time in such a schedule by  $C(\pi)_j, \forall j \in J^O$ . The goal of the rescheduling is to build a new schedule  $\sigma$  of all jobs that minimizes some regular objective function, subject to a maximum amount  $\varepsilon$  of disruption of the original schedule. As introduced by [1], the disruption of an old job is measured as the absolute deviation of completion time, i.e.  $\Delta_j = |C(\sigma)_j - C(\pi)_j|, \forall j \in J^O$ . The overall disruption is then measured as the maximum deviation  $\Delta_{max} = \max_{j \in J^O}(\Delta_j)$ .

Our contributions to the problem are:

- to state which problems may lead to the insertion of idle times, depending on the objective function;

---

<sup>\*</sup>[stefan.lendl@uni-graz.at](mailto:stefan.lendl@uni-graz.at). Department of Operations and Information Systems, University of Graz, Universitaetsstrasse 15/E3, 8010 Graz, Austria.

<sup>†</sup>[ulrich.pferschy@uni-graz.at](mailto:ulrich.pferschy@uni-graz.at). Department of Operations and Information Systems, University of Graz, Universitaetsstrasse 15/E3, 8010 Graz, Austria.

<sup>‡</sup>[elena.rener@polito.it](mailto:elena.rener@polito.it). Department of Management and Production Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy.

<sup>§</sup>[fabio.salassa@polito.it](mailto:fabio.salassa@polito.it). Department of Management and Production Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy.

<sup>¶</sup>[tkindt@univ-tours.fr](mailto:tkindt@univ-tours.fr). Département Informatique - Polytech Tours 64 avenue Jean Portalis 37200 Tours, France.

- for the particular problem  $1 \mid \Delta_{max} \leq \varepsilon \mid \overline{wC}$  to settle the complexity and to provide a pseudo-polynomial time dynamic programming algorithm for its exact solution with running time complexity  $O(n_N \varepsilon^2)$ .

## 2 Jobs sequencing and idle times insertion

Despite the fact that the disruption measure and any regular objective would try, if considered on its own, to push any job to the left, combining the two aspects in the same problem, there may arise the need of inserting idle times. This is because the objective function can require the resequencing of old jobs when combined with the new ones. As a result, inserting idle times may decrease the disruption of some of the old jobs that finish earlier than their original completion time. The related problems  $(\overline{C}, L_{max})$  that have been considered in the literature so far, do not require any idle times. However, we can show with some small examples that when the total number of tardy jobs or the total tardiness are considered, idle times may well be required.

For the case of minimizing the total weighted completion times, we show that the sequencing of the old jobs with respect to their initial WSPT optimal sequence is maintained and there are no idle times, similar to  $\overline{C}$  and  $L_{max}$ .

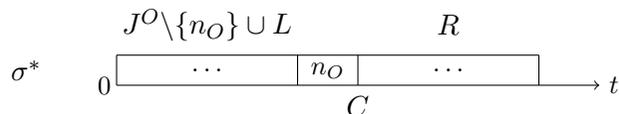
## 3 Complexity and solution for $1 \mid \Delta_{max} \leq \varepsilon \mid \overline{wC}$

**Theorem 1** *The recognition version of the problem  $1 \mid \Delta \leq \varepsilon \mid \overline{wC}$  is binary NP-complete.*

The proof is given by reduction from the PARTITION problem ([2]).

In the following, an algorithm for the exact solution of the problem is provided. First, it is to be noted that, since the old jobs are sequenced as in  $\pi$ , in an optimal schedule for problem  $1 \mid \Delta \leq \varepsilon \mid \overline{wC}$ ,  $\Delta_{max} = \Delta_{n_O}$ . In other words, the maximum disruption is always equal to the disruption of the last job in  $J^O$ .

Given that, we define two sets  $L$  and  $R$  of new jobs, where  $L$  contains all new jobs scheduled before  $n_O$  in an optimal schedule and  $R$ , starting at some time  $C$ , all those scheduled later. Notice that only new jobs  $j$  for which  $\frac{w_j}{p_j} > \frac{w_{n_O}}{p_{n_O}}$  holds, will be scheduled before  $n_O$ . An optimal schedule will look like this:



The two subsets to the left and to the right of job  $n_O$  contain jobs scheduled by WSPT since this is optimal for old jobs and does not affect the disruption when applied to the new jobs (but still minimizes the objective function). Also, the completion time of  $n_O$  will never exceed  $P_O + \varepsilon$ , where  $P_O = \sum_{j \in J^O} p_j$ .

Stated as above, the problem reduces to finding the optimal subset of new jobs that should be scheduled before  $n_O$ . This problem is solved via dynamic programming by the algorithm following below. The optimal solution is found by iteratively computing the best subset of new jobs to be put in  $L$ , when the jobs in  $L$  have a total length of exactly  $C - P_O$ .

We define a function  $DP_j[C, P]$  that computes the minimum total weighted completion time for a schedule that contains all old jobs and new jobs  $1, \dots, j$ , where  $R$  has a starting time  $C$  and the total length to be occupied by new jobs in  $L$  is  $P \in \{0, 1, \dots, \varepsilon\}$ . For notational convenience we also include infeasible cases with  $P < 0$ . The value of  $C$  is bounded to lie in the interval  $[P_O, P_O + \varepsilon]$  to make the schedule feasible.

We also write as  $i \in J^O(J^N) : i \rightarrow j$ , a job  $i$  belonging to the set  $J^O(J^N)$  that precedes job  $j$  in a WSPT order and  $P_N = \sum_{j \in J^N} p_j$ .

### Algorithm

*Input*

$\pi, J^N, \varepsilon \leq P_N$

*Preprocessing*

Order and index all jobs by WSPT.

*Boundary Conditions*

$DP_j[C, 0] = f(\pi), \quad \forall j > 0, \forall C \geq 0$

$DP_0[C, P] = \infty, \quad \forall P > 0, \forall C \geq 0$

$DP_j[C, P] = \infty, \quad \forall P < 0, \forall j > 0, \forall C \geq 0$

*Recurrence Relation*

$$DP_j[C, P] = \min \left( DP_{j-1}[C, P] + w_j \left( C + \sum_{i \in J^N : i \rightarrow j} p_i - P + p_j \right), \right. \\ \left. DP_{j-1}[C, P - p_j] + \left( \sum_{i \in J^O : j \rightarrow i} w_i \right) p_j + w_j \left( P + \sum_{i \in J^O : i \rightarrow j} p_i \right) \right)$$

Given the above algorithm, we seek the optimal solution to the problem given by

$$\min_{P_O \leq C \leq P_O + \varepsilon} DP_{n_N}[C, C - P_O]. \quad (1)$$

At every stage of the recursion, the algorithm decides whether to put job  $j$  in  $R$  (first term of the recurrence relation) or  $L$  (second term). The value of the objective function is given in the first case by the contribution given by the previous new jobs scheduled in  $L$  and the weighted completion time of job  $j$  scheduled at the end of set  $R$ . In the second case, it is given by the old jobs scheduled after  $j$  that are shifted to the right by  $p_j$  time units and by the contribution of  $j$  that will be scheduled to the end of the available "slot" of  $P$  time units.

Notice that the recursion returns the optimal solution when the whole available space to place jobs in  $L$  is filled and there is no idle time between the last old job and the starting time  $C$  of jobs in  $R$ . However, solutions where  $C$  implies slack are

required during the recursion. Iterating over all possible values of  $C$  guarantees that the optimal total length of  $L$  is found. The time complexity of the algorithm is bounded by  $O(n_O + n_N \varepsilon^2)$ .

## References

- [1] N. HALL AND C. POTTS (2004). *Rescheduling for new orders*. Operations Research, 52(3):440-453.
- [2] M.R. GAREY AND D.S. JOHNSON (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman.

# An Exact Algorithm for the Linear Tape Scheduling Problem

Valentin Honoré\*    Bertrand Simon (Speaker)<sup>†</sup>    Frédéric Suter<sup>‡</sup>

---

## 1 Introduction

Initially designed for media recording, the usage domain of magnetic tapes has broadened over the decades and remains a real competitor to disk storage even for scientific data. The main advantages of this storage medium are a large storage capacity for a reasonable price, and better data preservation, security, and energy efficiency. Indeed, it has been estimated that total costs are reduced by a factor of 6 when archiving data on tape rather than disks. When a file is needed, the tape it is on will be fetched by a robotic arm, brought to a tape drive, and loaded. Mounting a tape into a drive requires a delay of about a minute, so it is more beneficial to mount a tape when many files have to be read. Moving the reading head to the next file needed is also a time-expensive operation, and the order in which files are read has to be optimized, which is the focus of this paper.

Recent tapes can store up to 10 terabytes of data on a one-kilometer-long physical storage, longitudinally divided into hundreds of parallel tracks logically linked as a coiled serpentine. We consider a simplified model of tapes on which optimal decisions are still not well understood and which models local considerations. Specifically, we consider magnetic tapes composed of a single linear track divided successively in disjoint files  $(f_1, \dots, f_{n_f})$  of integer size  $s(f_i)$ . Let  $\ell(f_i)$  be the *length* between the left of the tape and the left of the file  $f_i$  and  $r(f_i) = \ell(f_i) + s(f_i)$ . We are given a set of  $n$  read requests on  $n_{req}$  files among the files of the tape, with duplicates, where each request is a file.

The objective is to design a schedule (*i.e.*, a trajectory of the reading head on the linear tape) to read all the requested files when the reading head is initially positioned on the right of the tape. The specificity of this problem is that all files have to be read from the left to the right. We consider the average service time as a metric, to ensure a fair service among all requests. In order to model the temporality of a given schedule, we assume that the speed of the tape is constant, although it is a mechanical device with inertia. We moderate this inaccuracy by

---

\*valentin.honore@cc.in2p3.fr. IN2P3 Computing Center, CNRS, Villeurbanne, France

<sup>†</sup>bertrand.simon@cc.in2p3.fr. IN2P3 Computing Center, CNRS, Villeurbanne, France

<sup>‡</sup>frederic.suter@cc.in2p3.fr. IN2P3 Computing Center, CNRS, Villeurbanne, France

taking into account the deceleration induced by a U-turn of the tape as a nominal penalty. Note that we only consider read requests as write requests are usually processed separately. We also exclude update requests as they damage nearby data on the tapes. We therefore consider the following problem.

**Definition 1 (Linear Tape Scheduling Problem (LTSP))** *Given a set of requests on files, find a schedule of all requests minimizing the average service time.*

This problem has been previously studied by [Cardonha and Real, 2016, Cardonha and Real, 2018] without U-turn penalties and conjectured to be impossible to solve efficiently. Indeed, even simpler variations restricting either file requests to be unique or file sizes to be equal have been conjectured NP-hard. We solve this open problem in this paper by providing a polynomial algorithm optimally solving the unrestricted LTSP problem, also considering the deceleration as a U-turn penalty  $U$ . The objective is to design a schedule (*i.e.*, a trajectory of the reading head on the linear tape) to read all the requested files on a tape when the reading head is initially positioned on the right of the tape. We show that a carefully designed Dynamic Programming implementation (technique which has been considered in [Cardonha and Real, 2018] but was deemed not conclusive) allows us to compute an optimal schedule in a reasonable polynomial time.

## 2 Solving the LTSP problem

The main contribution of this paper is a dynamic programming algorithm called **DP**<sup>1</sup>. It uses carefully selected memoization to store the cost of specific solutions used to build an optimal schedule. Structural results show that optimal solutions can be described as a list of consecutive and possibly intricate *detours*: a detour  $(a, b)$  with  $a$  and  $b$  being two requested files with  $a$  on the left of  $b$  means that the head goes directly to  $r(b)$  then back to  $\ell(a)$  after first attaining  $\ell(a)$ , see Figure 1. The table of **DP** is indexed by three parameters: the delimiters of a tape portion and the number of file requests on the right of this portion that have not been read yet in a partial solution satisfying a few properties on the detours used. Each cell then computes the optimal way to continue any of these partial solutions on the considered portion and returns the corresponding overhead on the total cost. We show that this is sufficient to compute the optimal schedule.

**Theorem 2** **DP** solves optimally LTSP in time  $O(n_{req}^3 \cdot n)$ .

The complexity of **DP** may be prohibitive for an input containing hundreds of requested files. To address this issue, we provide a lighter algorithm named **LogDP** that reduces both the table dimensions and complexity to query a single

---

<sup>1</sup>Concurrently to this paper, a similar algorithm has also been proposed [Cardonha et al., 2021] with a slightly less general model.

cell and leads to a time complexity of  $O(n_{req} \cdot n \cdot \log^2 n_{req})$ . Only detours of span at most  $\lambda \cdot \log n_{req}$  are then considered, and the solution returned is optimal among this class of schedules. The parameter  $\lambda$  can be adjusted to trade accuracy for computing time. This solution is a 3-approximation if  $U = 0$  following a result from [Cardonha and Real, 2018].

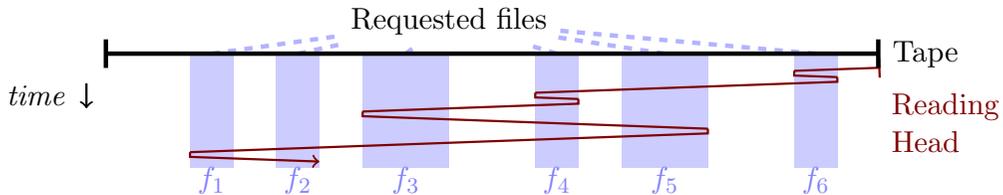


Figure 1: Example of schedule for reading six files described by the detour list  $[(f_6, f_6), (f_4, f_4), (f_3, f_5)]$ . Remaining files ( $f_1$  and  $f_2$ ) are read in a single pass.

Note that **DP** depends on  $n$  and not only on  $n_{req}$  so this algorithm is polynomial only if the input is given as a list of requests and not a list of files associated with the multiplicity of their requests. This is motivated by the fact that multiple file requests come from different applications, so the input size is linked to the number of requests. The complexity of the high-multiplicity version of this problem therefore remains open.

### 3 Performance evaluation

We compare the performance of our exact algorithm, **DP**, its suboptimal version **LogDP**, and existing algorithms [Cardonha and Real, 2018] using a real-world dataset.

#### 3.1 Contribution 1: dataset from production logs

The IN2P3 Computing Center, from which our dataset comes, uses tape storage for long-term projects in High Energy Physics and Astroparticles physics. Its tape library is currently composed of 48 TS1160 drives and can store up to 6,700 20TB IBM Jaguar E tapes. We applied several filtering steps to the raw dataset (that covers two weeks of activity) to obtain the inputs needed by the algorithms. We selected a set of 169 tapes of interest storing 3,387,669 files. The processed dataset corresponds to a total of 119,877 files stored on the 169 tapes. To the best of our knowledge, this is the first time that a realistic dataset for magnetic tape storage is made publicly available (<https://figshare.com/s/a77d6b2687ab69416557>). This dataset could be used in other contexts as well. It corresponds to 169 distinct instances of LTSP to solve.

### 3.2 Contribution 2: Performance evaluation of algorithms

We compared **DP** and **LogDP** ( $\lambda \in \{1, 5\}$ ) with state-of-the-art algorithms from [Cardonha and Real, 2018], modified to take the U-turn penalty  $U$  into account. Details concerning our implementation can be found in the full paper, and in the source code that is freely accessible online (<https://figshare.com/s/80cee4b7497d004dbc70>).

All algorithms have been evaluated with different values of  $U$ . Overall, our results validate the performance gain with our exact algorithm **DP**, and also show that **LogDP** is able to maintain a good trade-off between performance and computational cost.

## References

- [Cardonha and Real, 2016] Cardonha, C. and Real, L. C. V. (2016). Online Algorithms for the Linear Tape Scheduling Problem. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*, London, UK.
- [Cardonha and Real, 2018] Cardonha, C. and Real, L. C. V. (2018). Theoretical and practical aspects of the linear tape scheduling problem. *CoRR*, abs/1810.09005v1.
- [Cardonha et al., 2021] Cardonha, C. H., Ciré, A. A., and Real, L. C. V. (2021). On exact and approximate policies for linear tape scheduling in data centers. *CoRR*, abs/2112.07018.
- [Honoré et al., 2021] Honoré, V., Simon, B., and Suter, F. (2021). An exact algorithm for the linear tape scheduling problem. *CoRR*, abs/2112.09384.

# Additive Approximation Schemes for Load Balancing Problems

Moritz Buchem <sup>\*</sup>   Lars Rohwedder <sup>†</sup>   Tjark Vredeveld (Speaker) <sup>‡</sup>  
Andreas Wiese <sup>§</sup>

---

## 1 Introduction

We consider a family of load balancing problems on identical parallel machines. In these problems,  $n$  jobs need to be processed by  $m$  machines such that each job is completely processed by one single machine. Each job  $j$  has a given processing time  $p_j$  and the load of a machine  $i$  is the sum of the processing times of the jobs assigned to  $i$ . The goal is to find a schedule that optimizes some objective function over the machine loads. In this work, we consider three specific load balancing objectives: (1) makespan minimization, (2) the *Santa Claus* problem in which we want to maximize the minimum machine load and (3) the *envy-minimizing Santa Claus* problem with the goal of minimizing the difference between the maximum and minimum machine load.

All three load balancing problems are well-known to be strongly NP-hard (see [4]). This led to an extensive research on approximation algorithms for these problems. The problem of minimizing the makespan is the one of the most classical scheduling problems on parallel machines. The first approximation algorithms based on list scheduling were introduced by Graham [5, 6]. When the number of machines is a constant Sahni [9] showed that the problem admits an FPTAS. In the case that the number of machines is part of the input Hochbaum and Shmoys [7] found a PTAS which led to a lively line of research improving the running to an EPTAS [8]. The second load balancing objective we consider, the Santa Claus objective, has been considered in the restricted assignment case by Bansal and Sviridenko [2] and an (E)PTAS was presented in the identical case by Alon et al. [1] and Jansen et al. [8].

---

<sup>\*</sup>m.buchem@maastrichtuniversity.nl. Dept. of Quantitative Economics, Maastricht University, P.O. Box 616, 6200 MD, Maastricht, The Netherlands

<sup>†</sup>l.rohwedder@maastrichtuniversity.nl. Dept. of Quantitative Economics, Maastricht University, P.O. Box 616, 6200 MD, Maastricht, The Netherlands

<sup>‡</sup>t.vredeveld@maastrichtuniversity.nl. Dept. of Quantitative Economics, Maastricht University, P.O. Box 616, 6200 MD, Maastricht, The Netherlands

<sup>§</sup>a.wiese@vu.nl. VU Amsterdam, School of Business and Economics, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands

In contrast to previous work, we study approximation algorithms and schemes for which we measure their performance as the absolute difference between the value of their computed solution and the value of the optimal solution. To ensure that these additive approximation algorithms lead to same results on scaled instances, we express the absolute difference in terms of an instance dependent parameter  $h(I)$ . This parameter may be much smaller than  $OPT(I)$ .

**Definition 1** *For a given optimization problem, consider a quantity defined by  $h(I)$  for each instance  $I$ . An algorithm is an additive approximation algorithm w.r.t.  $h$  if for any instance  $I$  it computes in polynomial time a solution with value  $A(I)$  satisfying  $|A(I) - OPT(I)| \leq h(I)$ . An additive approximation scheme w.r.t.  $h$  is a family of algorithms containing an additive approximation algorithm w.r.t.  $\epsilon \cdot h$  for each  $\epsilon > 0$ .*

Our main result is the introduction of an additive approximation scheme for the considered load balancing problems on identical parallel machines.

**Theorem 2** *There is an algorithm for load balancing on identical parallel machines that for any  $\epsilon > 0$  computes a solution with value  $A(I)$  such that*

$$|A(I) - OPT(I)| \leq \epsilon p_{\max},$$

where  $p_{\max}$  is the maximum processing time of the jobs, in running time of  $m^{O(2^{1/\epsilon})} \cdot n^{O(1/\epsilon \cdot 2^{1/\epsilon})}$ .

To obtain this result, we present a fractional relaxation for this general class of load balancing problems, which we call the *slot-MILP*. In this MILP we assign jobs fractionally, however, for each job type we need to assign an integral number of slots to each machine. The key idea of the additive approximation scheme is to first find a solution to the slot-MILP and, finally, round the solution to obtain an integral assignment of jobs to machines without deviating from the slot-MILP solution by more than  $\epsilon p_{\max}$  on each machine.

## 2 The slot-MILP

The slot-MILP can be interpreted as a strengthened variant of the assignment-LP. In the assignment-LP jobs are fractionally assigned to machines. In the slot-MILP we first group jobs of similar sizes without rounding the job sizes. Let  $\epsilon > 0$  and assume w.l.o.g. that  $1/\epsilon \in \mathbf{N}$ . First, we partition the jobs into sets  $\mathcal{J}_1, \dots, \mathcal{J}_{1/\epsilon}$ , where for  $k = 1, \dots, 1/\epsilon$  the set  $\mathcal{J}_k$  contains all jobs  $j \in \mathcal{J}$  with  $p_j \in ((k-1)\epsilon \cdot p_{\max}, k\epsilon \cdot p_{\max}]$ . We define fractional assignment variables  $x_{i,j}$  for each job-machine pair as well as integral slot variables  $y_{i,k}$  for each job-size-machine pair indicating the number of jobs in  $\mathcal{J}_k$  assigned to machine  $i$ . Finally, we use upper and lower bound variables on the machine loads to define the objective functions.

$$\begin{aligned}
\sum_{i \in \mathcal{M}} x_{i,j} &= 1 && \forall j \in \mathcal{J} \\
\ell \leq \sum_{j \in \mathcal{J}} p_j x_{i,j} &\leq u && \forall i \in \mathcal{M} \\
\sum_{j \in \mathcal{J}_k} x_{i,j} &= y_{i,k} && \forall i \in \mathcal{M}, \forall k \in \{1, \dots, 1/\epsilon\} \\
x_{i,j} &\geq 0 && \forall j \in \mathcal{J}, i \in \mathcal{M} \\
y_{i,k} &\in \mathbf{N}_0 && \forall i \in \mathcal{M}, k \in \{1, \dots, 1/\epsilon\}
\end{aligned} \tag{1}$$

### 3 Rounding the relaxation

Given a solution  $(x, y)$  to the slot-MILP, we compute an initial solution by assigning each job  $j \in J_k$  to an arbitrary slot of type  $k$ . In this solution there might be a machine  $i$  whose load is not in  $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$ , i.e., the load is too small or too large. We present a local search algorithm that repeatedly swaps pairs of jobs from the same set  $J_k$  such that eventually each machine  $i$  has a load in  $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$ . As we only swap jobs within a set  $J_k$ , we ensure that the number of jobs from set  $J_k$  on machine  $i$ , remains  $y_{ik}$ . Using a potential function, we show that after  $O(n^3)$  swaps, each machine has load in  $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$  and that a swap can be found in  $O(n^2)$  time. For more details, we refer to [3]

### Acknowledgements

We wish to thank José Verschae, Alexandra Lassota and Klaus Jansen for helpful discussions.

### References

- [1] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.
- [2] N. Bansal and M. Sviridenko. The santa claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC*, pages 31–40, 2006.
- [3] M. Buchem, L. Rohwedder, T. Vredeveld, and A. Wiese. Additive approximation schemes for load balancing problems. In *Proceedings of the 48th ICALP*, pages 42:1 – 42:17, 2021.
- [4] M. R. Garey and D. S. Johnson. “strong” NP-completeness results: motivation, examples, and implications. *Journal of the ACM*, 25:499–508, 1978.

- [5] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
- [6] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.
- [7] 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.
- [8] K. Jansen, K.-M. Klein, and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45:1371–1392, 2020.
- [9] S. K. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23(1):116–127, 1976.

# A Primal-Dual and Primal-Greedy Approximation Framework for Weighted Covering Problems

Britta Peis <sup>\*</sup>    Niklas Rieken <sup>†</sup>    José Verschae (speaker) <sup>‡</sup>  
Andreas Wierz <sup>§</sup>

---

## 1 Introduction

We consider a general framework for covering problems. Our model aims to capture the basic properties that allow for two natural and widely used greedy algorithms to be applied. Given a finite set  $E$  of resources, a cost function  $c: E \rightarrow \mathbb{R}_+$ , a demand function  $r: 2^E \rightarrow \mathbb{R}$ , and set-specific weights  $a_{S,e} \in \mathbb{R}_+$  for every set  $S \subseteq E$  and every resource  $e \in E$ , we are interested in solving the following integer linear program,

$$\min_{x \in \mathbb{Z}_+} \left\{ \sum_{e \in E} c_e x_e \mid \sum_{e \in S} a_{S,e} x_e \geq r(S), S \subseteq E \right\}. \quad (\hat{P})$$

Written in matrix form, the problem can be stated as  $\min\{c^T x \mid Ax \geq r, x \in \mathbb{Z}_+^E\}$ , where  $A = [a_{S,e}]_{S \subseteq E, e \in E} \in \mathbb{R}_+^{2^E \times E}$  denote the underlying constraint matrix. We assume that  $a_{S,e} = 0$  for  $e \notin S$ . Hence, this problem can be seen as follows: for any subset  $S$  of elements, we must pick a subset in  $S$  (with multiplicities) in order to satisfy a demand of  $r(S)$ . Picking an element  $e$  once covers an amount of  $a_{S,e}$  of the demand.

Many fundamental combinatorial optimization problems can be formulated as such a weighted integer covering problem, e.g., set cover [4, 6], knapsack cover [2], the problem to minimize a linear function over a contra-polymatroid [5], or the submodular cover problem [7]. Is it worth noticing that the formulation  $(\hat{P})$  arises naturally in this sort of problems. To exemplify the idea, consider a set cover instance given by a ground set  $U$  and a collection  $E$  of sets  $A_1, \dots, A_n \subseteq U$ , each with a cost  $c_i$ . The objective is to find a collection  $E' \subseteq E$  that covers all  $U$  at

---

<sup>\*</sup>peis@oms.rwth-aachen.de. Fakultät für Wirtschaftswissenschaften, RWTH Aachen University, Kackertstr. 7, Aachen, Germany

<sup>†</sup>rieken@oms.rwth-aachen.de. Fakultät für Wirtschaftswissenschaften, RWTH Aachen University, Kackertstr. 7, Aachen, Germany

<sup>‡</sup>jverschae@uc.cl. Instituto de Ingeniería Matemática y Computacional, Pontificia Universidad Católica de Chile, Av Vicuña Mackenna 4860, Santiago, Chile

<sup>§</sup>wierz@oms.rwth-aachen.de. Fakultät für Wirtschaftswissenschaften, RWTH Aachen University, Kackertstr. 7, Aachen, Germany

minimum cost. Given a subset  $S \subseteq E$ , let us assume for a moment that the sets in  $E \setminus S$  are already chosen in our solution. Hence, with the sets in  $S$  we must cover  $r(S) := |U \setminus \bigcup_{A \in E \setminus S} A|$  elements in  $U$ , where each set  $A \in S$  can cover at most  $a_{S,A} = |A \setminus \bigcup_{B \notin S} B|$  many of them. This yields a formulation as in  $(\hat{P})$  for set cover. The other mentioned covering problems can be handled similarly.

It is also worth noticing that several of the mentioned covering problems constitute building blocks to other more advanced questions. These include different scheduling problems; see for example the work by Bansal and Pruhs [1].

Our interest lies in understanding the structural properties of the system  $(\hat{P})$  that allow the use of two common algorithm based on the primal and the dual LP-relaxations:

$$(P) \quad \min_{x \geq 0} \{c^T x \mid Ax \geq r\} \qquad (D) \quad \max_{y \geq 0} \{r^T y \mid A^T y \leq c\}.$$

## 2 The Primal-Dual and Primal Greedy Algorithms

Primal-dual algorithms are a popular way of designing exact or approximate algorithms for covering problems. In our setting, we construct greedily a dual solution, and then raise the primal variables of tight dual inequalities with the aim of obtaining a feasible primal solution. More precisely, we initialize a dual solution  $y \equiv 0$  and consider the set  $S_1 = E$ . In iteration  $i$  we raise variable  $y_{S_i}$  until a dual inequality indexed by an element  $e \in E$  becomes tight, that is, until the dual constraint  $\sum_{S \ni e} a_{S,e} y_S \leq c_e$  is satisfied with equality. For the bottleneck element  $e$ , we define the primal variable  $x_e = \left\lceil \frac{r(S) - r(S \setminus \{e\})^+}{a_{S_i,e}} \right\rceil$ . Then we iterate by defining  $S_{i+1} = S_i \setminus \{e\}$ . The algorithm terminates when the current set satisfies that  $r(S_i) \leq 0$  or  $S_i = \emptyset$ . It is not hard to see that the dual solution is always feasible, and that the constructed primal solution is feasible on the chain  $S_\ell \subseteq \dots \subseteq S_1$ .

A different algorithm that has also been widely used is a natural primal-greedy algorithm. Here, we also start with a set  $S_1 = E$ . In iteration  $i$  we choose an element  $e$  that minimizes the ratio  $\frac{c_e}{a_{S_i,e}}$ , that is, the element that gives you the most *bang for the buck* for set  $S_i$ . Then we define  $S_{i+1} = S_i \setminus \{e\}$  and set  $x_e = \left\lceil \frac{r(S) - r(S \setminus \{e\})^+}{a_{S_i,e}} \right\rceil$ . The algorithm finishes when  $r(S_i) \leq 0$  or  $S_i = \emptyset$ . As before, the algorithm guarantees the feasibility of the primal solution for the inequalities indexed by the chain  $S_\ell \subseteq \dots \subseteq S_1$ .

There are several natural questions regarding these algorithms and problem  $(\hat{P})$ . The first one is under which conditions do these algorithms yield a feasible primal solution, as the procedure only guarantees feasibility on the mentioned chain. The second one is what are the approximation factors.

We say that  $r$  is *weighted supermodular* w.r.t.  $A$  if for all  $S \subseteq T \subseteq E$  and  $e \in S$  with  $a_{S,e}, a_{T,e} \neq 0$  it holds that

$$\frac{r(S) - r(S \setminus \{e\})}{a_{S,e}} \leq \frac{r(T) - r(T \setminus \{e\})}{a_{T,e}}. \quad (1)$$

We say that  $(A, r)$  is a *greedy system* if the following three properties hold: (P1)  $S \subseteq T \subseteq E$  implies  $r(S) \leq r(T)$ , (P2)  $S \subseteq T \subseteq E$  implies  $a_{S,e} \leq a_{T,e}$  for all  $e \in E$ , and (P3)  $r$  is weighted supermodular w.r.t.  $A$ .

**Theorem 1.** *If  $(A, r)$  is a greedy system, then the primal-greedy and primal-dual algorithms are guaranteed to construct a feasible solution  $x$  of  $(\hat{P})$ .*

Moreover, we can show that each of the three properties in the definition of greedy-systems are necessary to guarantee feasibility of the constructed primal solutions.

### 3 Performance Guarantees

It is not hard to see that greedy systems can have unbounded integrality gaps, even for simple problems as knapsack cover. Following the idea by Carr et al. [3] for the so-called knapsack-cover inequalities, we can truncate greedy systems by considering a modified matrix  $A'$  with entries  $a'_{S,e} = \min\{a_{S,e}, (r(S)^+ - r(S \setminus \{e\})^+)\}$  for all  $S \subseteq E, e \in E$ . If  $(A, r)$  is a greedy system, we say that  $(A', r)$  is a *truncated greedy system*. We can show that truncation does not affect the feasibility of  $(\hat{P})$  and, similarly as in Theorem 1, that the solutions returned by the primal-greedy and primal-dual are feasible.

We define two parameters,  $\delta_0$  and  $\delta$ , that we use to derive approximation factors for our algorithms. Let  $E^* = \{e_1, \dots, e_\ell\}$  denote the set of bottleneck elements as selected by the primal-dual algorithm, and recall that  $S_{i+1} = S_i \setminus \{e_i\}$ . Then, the parameters are

$$\delta_0 = \max_{\substack{S \subseteq E \\ e \in E: a'_{S,e} > 0}} \frac{a'_{E,e}}{a'_{S,e}} \quad \text{and} \quad \delta = \max_{i \in [\ell]} \frac{a'_{E,e_i}}{a'_{S_i,e_i}}.$$

Note that  $\delta \leq \delta_0$ .

**Theorem 2.** *The primal-dual algorithm is a  $2\delta$ -approximation for  $(\hat{P})$  when applied to the truncated system  $(A', r)$ . The primal-greedy algorithm is a  $2(1 + \log \delta_0)$ -approximation for  $(\hat{P})$  when applied to  $(A', r)$ .*

It is worth noticing  $\delta = 1$  for contra-polymatroids and knapsack cover, but that  $\delta$  can be as large as the size of the ground set for set cover. Hence, this theorem in particular recovers the 2-approximation for knapsack-cover via the primal-dual framework, and a small extension also the exact optimal algorithm for contra-polymatroids. Similarly, we also recover the classic  $O(\log(n))$ -approximation for set cover, among several other results.

### References

- [1] Bansal, N., Pruhs, K., The Geometry of Scheduling. SICOMP. 43:1684–1698, 2014.

- [2] Carnes, T., Shmoys, D.B., Primal-dual schema for capacitated covering problems. *Math. Program.* 153:289–308, 2015.
- [3] Carr, R.D., Fleischer, L.K., Leung, V.J., Phillips, C.A., Strengthening integrality gaps for capacitated network design and covering problems, *SODA 2000*, pp. 106–115.
- [4] Chvatal, V., A Greedy Heuristic for the Set-Covering Problem. *Math. of OR.* 4:233–235, 1979.
- [5] J. Edmonds, Submodular Functions, Matroids, and Certain Polyhedra. *Combinatorial Structures and their Applications*, 1970.
- [6] Lovász, L. On the ratio of optimal integral and fractional covers. *Discrete Math.* 13:383–390, 1975.
- [7] Wolsey, L. A., An analysis of the greedy algorithm for the submodular set covering problem, *Combinatorica*, 2:385–393, 1982.

# On Flow Shop Scheduling with Job-Dependent Storage Requirements

Alexander Kononov (Speaker) \*      Marina Pakulich †

---

## 1 Introduction

We consider buffer-constrained flow shop problems that arise when scheduling the loading of media objects from online databases for an automatically assembled multimedia presentation. The media objects are presented sequentially without a predefined order in a given presentation. Each object has a download time and a playback time. The presentation of an object cannot be started earlier than it will finish downloading. It is required to determine the sequence of loading and presentation of objects to minimize the completion time of the entire presentation. This technique has significant value in digital library/museum applications. In this application, the amount of buffer by a job is job-dependent and linearly proportional to its download time. Without loss of generality, we assume that the loading time of an object is equal to its file size. Indeed, we can scale file size such that a unit of size is loaded per unit of time. We also suppose that the presentation process is non-resumable, i.e., it cannot be interrupted and resumed later.

We consider two different models for loading the objects to the buffer. In [8], the authors considered the case when the loading process is non-resumable and loading a new media object is impossible if there is not enough free space in the buffer to store it. That is, if the size of the incoming media object is greater than the size of free space available in the buffer, we have to wait until the completion of the presentation(s) of one or more buffered jobs. We call such a problem as PP-problem. In [6], the authors considered the case where the download process is resumable, i.e., the download of an incoming media object can be interrupted and continued after the completion of some buffered job. In such a case, we require that the total size of the buffered and partially buffered jobs at any instant not exceed the size of the buffer. We call the second problem as AP-problem.

We are given a set of  $n$  jobs  $J = 1, 2, \dots, n$ , two machines, and a buffer with size  $\Omega$ . Each job has two operations. The first operation of each job has to be executed on the first machine, and the second operation of each job has to be

---

\*alvenko@math.nsc.ru. Sobolev Institute of Mathematics, ak Koptyuga 4, 630090, Novosibirsk, Russia.

†pakulichmarina@gmail.com Novosibirsk State University, ak Koptyuga 2, 630090, Novosibirsk, Russia.

executed on the second machine. The second operation of each job cannot start earlier than the completion of the first operation of the same job. Let  $a_j$  and  $b_j$  be the processing times of the first and the second operations of job  $j$ , respectively. Let  $J_t$  be a subset of jobs that reside or are being loaded in the buffer at time  $t$  and  $a_j(t)$  be the part of the first operation of job  $j$  that is completed at time  $t$ . Then the following constraint must hold in the PP-problem:  $\sum_{i \in J_t} a_i \leq \Omega$  and in the AP-problem:  $\sum_{i \in J_t} a_i(t) \leq \Omega$ . Each machine can process at most one operation at a time. Preemption is allowed on the first machine, but preemption is not allowed on the second machine. The objective is to minimize the makespan.

## 2 Related results

The flow shop problems with a buffer have been extensively studied in the literature on scheduling, but most of these publications consider an intermediate buffer between stages and assume that this buffer limits only the number of jobs that have completed one operation and are waiting for the commencement of the next one [2, 4, 9, 10]. In contrast, we consider the scheduling problem, in which the buffer requirement varies from job to job, and each job seizes the required storage space during its processing on the machines as well as during its waiting time between the operations. Our problem also differentiates from the resource-constrained scheduling that assumes that the additional resource is consumed by a job only during its processing on the machines but it is released between the operations [1, 3].

Due to space limitation, we list only the main results directly related to our study. A more detailed overview of the results can be found in [7]. Both the PP-problem and AP-problem are strongly NP-hard [8, 6]. For the AP-problem, there exists an optimal schedule in which both machines process the jobs in the same order [6]. For the PP-problem, it has been shown in [5] that there are instances for which the set of optimal schedules does not contain a permutation schedule.

Note that if the buffer size is large enough, for example, it may contain all jobs, then the problem is equivalent to the two-machine flow shop problem without buffer, and it can be solved in  $O(n \log n)$  time by Johnson's algorithm. In [7], the question of how the buffer size affects the computational complexity of the PP-problem was studied. The authors presented a  $O(n \log n)$ -time algorithm for the case when  $3.5 \max a_i + \max\{0.5 \max a_i, \max b_i\} \leq \Omega$  and proved that the PP-problem remains NP-hard even if for any  $\delta > 0$ ,  $\frac{4 \max\{a_i, b_i\}}{1+\delta} \leq \Omega$  holds.

## 3 Our results

In this paper we improve the result obtained in [7] by extending the class of polynomially solvable instances of the PP-problem.

**Theorem 1** *Any instance of the PP-problem that satisfies  $3 \max a_i + \max b_i \leq \Omega$  can be solved in  $O(n \log n)$  time.*

Thus, Theorem 1, together with the NP-hardness result from [7], establishes the exact borderline between the NP-hard and polynomial-time solvable instances of the PP-problem with respect to the ratio between the buffer size and the maximum operation length.

We also get the similar results for the AP-problem.

**Theorem 2** *Any instance of the AP-problem that satisfies  $2 \max a_i + \max b_i \leq \Omega$  can be solved in  $O(n \log n)$  time.*

**Theorem 3** *The AP-problem remains NP-hard even if for any  $\delta > 0$ ,  $\frac{3 \max\{a_i, b_i\}}{1+\delta} \leq \Omega$  holds.*

## References

- [1] BLAZEWICZ, J., LENSTRA, J.K., KAN, A.R. (1983). *Scheduling subject to resource constraints: classification and complexity*. Discrete Appl. Math. 5(1), 11–24.
- [2] BRUCKER, P., HEITMANN, S., HURINK, J. (2003). *Flow-shop problems with intermediate buffers*. OR Spectr. 25(4), 549–574.
- [3] BRUCKER, P., KNUST, S. (2012). *Complex Scheduling*. Springer, Berlin.
- [4] EMMONS, H., VAIRAKTARAKIS, G. (2013). *Flow Shop Scheduling*. Springer, Berlin.
- [5] FUNG, J. AND ZINDER, Y. (2016). *Permutation schedules for a two-machine flow shop with storage*. Oper. Res. Lett. 44(2), 153–157.
- [6] KONONOV, A., HONG, J.S., KONONOVA, P., LIN, F.C. (2012). *Quantity-based buffer-constrained two-machine flowshop problem: active and passive prefetch models for multimedia applications*. J. Sched. 15(4), 487–497.
- [7] KONONOV, A., MEMAR J., ZINDER Y. (2021). *On a borderline between the NP-hard and polynomial-time solvable cases of the flow shop with job-dependent storage requirements*. appear in Journal of Global Optimization, <https://doi.org/10.1007/s10898-021-01097-w>.
- [8] LIN, F.-C., HONG, J.-S., LIN, B. M. T. (2008). *A two-machine flow shop problem with processing time-dependent buffer constraintsan application in multimedia problem*. Computers & Operations Research, 36(4), 1158–1175.
- [9] PAPADIMITRIOU, C.H., KANELLAKIS, P.C. (1980). *Flowshop scheduling with limited temporary storage*. J. ACM 27(3), 533–549.
- [10] PINEDO, M.L. (2016) *Scheduling: Theory, Algorithms, and Systems*. Springer, Berlin.

- [11] ZINDER Y., KONONOV A., FUNG J. (2021). *A 5-parameter complexity classification of the two-stage flow shop scheduling problem with job dependent storage requirements*. Journal of Combinatorial Optimization Vol. 42, P. 276–309.

# Flow Time Scheduling and Prefix Beck-Fiala

Nikhil Bansal \*

Lars Rohwedder (Speaker) †

Ola Svensson ‡

---

## 1 Introduction

We consider the general *unrelated machines model* with arbitrary release times; with input a set  $J$  of  $n$  jobs, a set  $M$  of  $m$  machines, arbitrary processing times  $(p_{ij})_{i \in M, j \in J}$ , and release times  $(r_j)_{j \in J}$ . A job must be scheduled on a single machine and we allow preemptions<sup>1</sup>. We consider the objectives of minimizing the maximum flow time,  $\max_{j \in J} F_j$ , and the total flow time,  $\sum_{j \in J} F_j$ , where  $F_j$  denotes the flow time of  $j$ .

Minimizing flow time on unrelated machines is notoriously difficult, and the best known approximation ratios are  $O(\log n)$  and  $O(\log n \log P)$  for max and total flow time [2], where  $P$  is the ratio of maximum to minimum (finite)  $p_{ij}$ , and can be bounded by  $\text{poly}(n)$ . These algorithms are based on rounding the natural LP relaxations, and also give the best known integrality gap for these problems. The best known hardness bounds are  $3/2$  [5] and  $\Omega(\log P / \log \log P)$  [4]. So, roughly there is a factor  $\log n$  gap for both problems.

In this paper, we relate flow time scheduling to prefix discrepancy, yielding new improved bounds for the problems above and new directions in discrepancy.

**Prefix Discrepancy.** In standard discrepancy, we are given a collection of vectors  $v^{(1)}, v^{(2)}, \dots, v^{(n)} \in \mathbb{R}^m$  and the goal is to find signs  $\epsilon_1, \epsilon_2, \dots, \epsilon_n \in \{-1, 1\}$  to minimize the  $\ell_\infty$ -norm of the signed sum  $\epsilon_1 v^{(1)} + \epsilon_2 v^{(2)} + \dots + \epsilon_n v^{(n)}$ . Beck and Fiala [3] showed that given arbitrary vectors  $v^{(1)}, v^{(2)}, \dots, v^{(n)} \in \mathbb{R}^m$  of bounded  $\ell_1$ -norm  $\|v^{(j)}\|_1 \leq 1$ , there exist signs  $\epsilon_1, \dots, \epsilon_n \in \{-1, 1\}$  such that

$$\|\epsilon_1 v^{(1)} + \dots + \epsilon_n v^{(n)}\|_\infty \leq 2.$$

In the *prefix Beck-Fiala* problem we wish to find signs  $\epsilon_1, \epsilon_2, \dots, \epsilon_n \in \{-1, 1\}$  so that for every prefix  $k = 1, 2, \dots, n$  that

$$\|\epsilon_1 v^{(1)} + \epsilon_2 v^{(2)} + \dots + \epsilon_k v^{(k)}\|_\infty \leq C.$$

---

\* [bansal@gmail.com](mailto:bansal@gmail.com). Dept. of Computer Science, U. Michigan, Ann Arbor, USA.

† [l.rohwedder@maastrichtuniversity.nl](mailto:l.rohwedder@maastrichtuniversity.nl). Sch. of Business and Economics, Maastricht U., Maastricht, Netherlands.

‡ [ola.svensson@epfl.ch](mailto:ola.svensson@epfl.ch). Sch. of Computer and Communication Sciences, EPFL, Lausanne, Switzerland

<sup>1</sup>This is necessary to get any meaningful guarantee for total flow time.

Banaszczyk [1] developed an ingenious technique using deep ideas from convex geometry, to bound the prefix Beck-Fiala discrepancy by  $O(\sqrt{\log n})$ .

## 2 Results

Our main result is a general reduction that allows us to transfer the techniques from discrepancy to flow time scheduling.

**Theorem 1.** *If the discrepancy of the prefix Beck-Fiala problem is bounded by  $C$ , then integrality gaps of the standard LP relaxations are upper bounded by  $O(C)$  and  $O(\min\{\log n, \log P\} \cdot C)$  for max flow time and total flow time, respectively.*

Using  $C = O(\sqrt{\log n})$  by the result of Banaszczyk [1], this gives improved bounds on the integrality gaps of  $O(\sqrt{\log n})$  for maximum flow time and a  $O(\min\{\log n, \log P\} \cdot \sqrt{\log n})$  for total flow time.

The prefix Beck-Fiala problem and its generalization called the prefix Komlós problem, are interesting problems on their own and it is been conjectured that the discrepancy for these problems may be  $O(1)$ . If true, Theorem 1 would imply tight integrality gaps of  $O(1)$  and  $O(\min\{\log n, \log P\})$  for maximum and total flow time.

**The idea.** The idea in the proof is to define a prefix Beck-Fiala instance based on the LP solution with one vector per job, and given a low discrepancy  $\pm 1$  coloring of vectors, use the signs to determine which machine to assign the corresponding job to. In general, a job might have  $m$  potential machines where it can be assigned, which is not a binary decision. The key is to reduce the problem to that of rounding a half-integral solution, which relates the problem to discrepancy in a clean way. This reduction from general to half-integral solutions is quite standard in discrepancy, however our reduction is somewhat different and requires more care.

**Algorithmic aspects.** Our reduction in Theorem 1 is constructive in both cases: if a discrepancy  $C$  coloring for prefix Beck-Fiala can be constructed in polynomial time, we get a  $O(C)$ -approximation algorithm for max flow time and a  $O(\min\{\log(n), \log(P)\} \cdot C)$ -approximation algorithm for total flow time.

Banaszczyk’s proof does not imply an efficient algorithm to find the signs  $\epsilon_1, \dots, \epsilon_n$ , and despite a lot of progress in algorithmic discrepancy the best bound achievable in polynomial time is  $O(\log n)$ . So our bounds in Theorem 1 do not improve the known approximation results. However, they do give improved efficient *estimation* algorithms.

**Conjectures for prefix discrepancy.** In fact, in our reductions in Theorem 1, the vectors in resulting prefix Beck-Fiala instances have sparsity only two (i.e., two non-zero entries). So we highlight this seemingly very special case.

**Conjecture 2.** *The discrepancy for the prefix Beck-Fiala problem where each vector  $v^{(i)}$  has sparsity 2 is bounded by a constant.*

Proving Conjecture 2 together with Theorem 1 would give tight bounds on the integrality gap in both variants of flow time, and moreover an algorithmic proof of the conjecture would give an optimal algorithm for the flow time problems.

Interestingly, the 2-sparse case of the prefix Beck-Fiala problem has been studied before in the further special case where the vectors have entries  $\{0, 1\}$ , due to the close connection with the classical 2-permutation problem in discrepancy, and it is known that the prefix discrepancy here is at most 1. Moreover, this can be achieved by a simple algorithm. This makes Conjecture 2 even more plausible.

**From max flow to total flow.** Our reduction from flow time to prefix Beck-Fiala actually shows an equivalence between max flow time and a special case of 2-sparse prefix Beck-Fiala. We make this explicit and use it to relate the two objectives.

**Theorem 3.** *If the integrality gap of the standard LP of max flow time is  $O(C)$ , then the integrality gap of the standard LP of total flow time is  $O(\min\{\log(n), \log(P)\} \cdot C)$ .*

In particular, improving our bound for max flow time would immediately imply also an improvement for total flow time.

## References

- [1] W. BANASZCZYK (2012). *On series of signed vectors and their rearrangements*. Random Struct. Algorithms, 40, 301–316.
- [2] N. BANSAL AND J. KULKARNI (2015). *Minimizing Flow-Time on Unrelated Machines*, STOC, 851–860.
- [3] J. BECK AND T. FIALA (1981). *“Integer-making” theorems*. Discrete Appl. Math., 3, 1–8.
- [4] N. GARG AND A. KUMAR (2007). *Minimizing Average Flow-time : Upper and Lower Bounds*. FOCS, 603–613.
- [5] J. K. LENSTRA, D. B. SHMOYS AND E. TARDOS (1990). *Approximation Algorithms for Scheduling Unrelated Parallel Machines*. Math. Program., 46, 259–271.

# An Inclusion-Exclusion based algorithm for permutation flowshop scheduling with job precedences

Olivier Ploton (Speaker) \*      Vincent T'kindt †

---

## 1 Introduction

In this work we consider a permutation flowshop scheduling problem with  $n$  jobs and  $m$  machines. Each job  $i$  has  $m$  operations  $O_{i1}, \dots, O_{im}$ , which must be processed on machines 1 to  $m$ , in this order. Each operation  $O_{ij}$  has a processing time  $p_{ij}$ , and each machine can process only one operation at a time. All machines must process operations in the same job order, and a schedule is essentially defined by this order. Moreover, this order may be constrained by precedences between jobs. To each job  $i$  is assigned a cost, obtained by computing a cost function  $f_i$  of the completion time  $C_{im}$  of the last operation of  $i$ . Then, the objective function is defined as  $f_{\max} = \max_{1 \leq i \leq n} f_i(C_{im})$ . Each cost is assumed to be regular, i.e. non-decreasing, and polynomially bounded with respect to its argument  $C_{im}$ , as it is the case for, e.g. makespan, lateness, or tardiness. Using the Graham notation [2], this problem is denoted by  $F|prmu, prec|f_{\max}$ . It is strongly NP-hard [1].

On a theoretical point of view, we are interested in the time and space worst-case complexities of algorithms solving this problem to optimality. The size of an instance  $\mathcal{I}$  is the number of jobs:  $|\mathcal{I}| = n$ . The measure of an instance  $\mathcal{I}$  is defined as the sum of processing times:  $||\mathcal{I}|| = \sum_{i,j} p_{ij}$ . As far as we now, while many specialized algorithms have been described for particular subinstances of this problem, the best known algorithm for the general case uses classical dynamic programming over job subsets [5] and runs with  $O^*(2^n ||\mathcal{I}||^m)$  time and space complexities. Our main contribution is an algorithm based on Inclusion-Exclusion, which achieves, without degrading time complexity, reduction of space complexity by a factor of  $2^{|M|}$ , where  $|M|$  is the number of jobs without a successor in the precedence constraints.

---

\*[olivier.ploton@univ-tours.fr](mailto:olivier.ploton@univ-tours.fr). Université de Tours, Laboratoire d'Informatique Fondamentale et Appliquée (LIFAT, EA 6300), EMR CNRS 7002 ROOT, Tours, France

†[tkindt@univ-tours.fr](mailto:tkindt@univ-tours.fr). Université de Tours, Laboratoire d'Informatique Fondamentale et Appliquée (LIFAT, EA 6300), EMR CNRS 7002 ROOT, Tours, France

## 2 Inclusion-Exclusion for the $F|prmu, prec|f_{\max}$ problem

To solve the problem, it suffices to count, for each threshold objective value  $\varepsilon$ , the number  $N(\varepsilon)$  of (semi-active) schedules with objective value at most  $\varepsilon$ . Then, the optimum objective value is computed by binary search, and an optimum schedule is determined using self-reduction, as described in [3]. We are about to compute  $N(\varepsilon)$  using Inclusion-Exclusion. For that, we relax the problem by allowing schedules with duplicated or absent jobs, and for any jobset  $J$ , we define  $N_J(\varepsilon)$  as the number of (relaxed) schedules using only jobs of  $J$ . The formula states [3, 4]:

$$N(\varepsilon) = \sum_{J \subset \{1, \dots, n\}} (-1)^{n-|J|} N_J(\varepsilon)$$

To compute  $N_J(\varepsilon)$ , we denote by  $\vec{C}$  the completion times of operations of a job (one per machine), and we define  $\vec{C} \bullet i$  as the completion times of operations of job  $i$  executed as soon as possible after  $\vec{C}$ . We also define  $Prec(i)$  as the set of predecessors of job  $i$ , and  $M$  as the set of maximal jobs, i.e. jobs without a successor. Then, we have  $N_J(\varepsilon) = N_{J,\varepsilon}[\emptyset, \vec{0}, n]$  where  $N_{J,\varepsilon}[S, \vec{C}, \ell]$  is the number of schedules composed of  $\ell$  jobs of  $J$ , with objective at most  $\varepsilon$ , starting from  $\vec{C}$ , after  $(n-\ell)$  jobs have been scheduled, including all jobs of  $S$ , each once, where  $S$  contains only non-maximal jobs. We compute it using this dynamic programming scheme:

$$\begin{aligned} N_{J,\varepsilon}[S, \vec{C}, 0] &= 1 \text{ if } S = \{1, \dots, n\} \setminus M, \text{ 0 otherwise} \\ N_{J,\varepsilon}[S, \vec{C}, \ell] &= \sum_{\substack{i \in J \\ i \notin S \\ Prec(i) \subset S \\ f_i(\vec{C} \bullet i) \leq \varepsilon}} N_{J,\varepsilon}[S \cup \{i\} \setminus M, \vec{C} \bullet i, \ell - 1] \quad \forall \ell > 0 \end{aligned}$$

We derive the following complexity results. Proposition 1 applies in the general case, and Proposition 2 applies when precedences are chains.

**Proposition 1** *For an instance  $\mathcal{I}$  of the  $F|prmu, prec|f_{\max}$  problem, with  $|M|$  jobs without successors, the algorithm computes an optimal solution in  $O^*(2^n \|\mathcal{I}\|^m)$  time and  $O^*(2^{n-|M|} \|\mathcal{I}\|^m)$  space.*

**Proposition 2** *For an instance  $\mathcal{I}$  of the  $F|prmu, chains|f_{\max}$  problem, with  $c$  chains of length  $\ell_1, \dots, \ell_c$ , the algorithm computes an optimal solution in  $O^*(2^c \ell_1 \dots \ell_c \|\mathcal{I}\|^m)$  time and  $O^*(\ell_1 \dots \ell_c \|\mathcal{I}\|^m)$  space.*

## References

- [1] M.R. GAREY, D.S. JOHNSON AND R. SETHI (1976). *The complexity of flow-shop and jobshop scheduling*. Mathematics of Operations Research, 1(2): 117–129.

- [2] R. L. GRAHAM, E. L. LAWLER, J. K. LENSTRA AND A. H. G. RINNOOY KAN (1979). *Optimization and approximation in deterministic sequencing and scheduling: a survey*. Annals of discrete mathematics, 5(2): 287–326.
- [3] F.V. FOMIN AND D. KRATSCH (2010). *Exact Exponential Algorithms*. Springer.
- [4] J. NEDERLOF (2008). *Inclusion exclusion for hard problems*. Master Thesis, Utrecht University.
- [5] G. J. WOEGINGER (2003). *Exact Algorithms for NP-Hard Problems: A Survey*. Combinatorial Optimization – Eureka, You Shrink! 185–207

# Scheduling with Speed Predictions

Eric Balkanski <sup>\*</sup>      Tingting Ou <sup>†</sup>      Clifford Stein <sup>‡</sup>  
Hao-Ting Wei (Speaker) <sup>§</sup>

---

## 1 Introduction

In many optimization problems, the decision maker faces crucial information limitations due to the input not being completely known in advance. A natural goal in such incomplete information settings is to find solutions that have a good worst-case performance over all potential input instances. However, even though worst-case analysis provides a useful measure for the robustness of an algorithm, it is also known to be a measure that often leads to needlessly pessimistic results.

A very recent, yet extensive, line of work on algorithms with predictions models the partial information that is often available to the decision maker and overcomes worst-case bounds by leveraging machine-learned predictions about the inputs (see [9] for a survey). In this line of work, the algorithm is given some type of prediction about the input, but the predictions are not necessarily accurate. The goal is to design novel algorithms that achieve stronger bounds when the provided predictions are accurate, which are called *consistency* bounds, but also maintain worst-case *robustness* bounds that hold even when the predictions are inaccurate. In the context of scheduling, [10, 8, 6, 2, 5] have studied different scheduling problems with predictions about the processing times of the jobs.

In addition to the jobs, the machines are another input to scheduling problems that are often not completely known in advance. Different models that capture this uncertainty about the machines have been suggested (e.g. [1, 4, 11]). In the scheduling with an unknown number of machines problem, introduced by Stein & Zhong [11], there is a first *partitioning* stage where jobs must be partitioned into bags without knowing the number of machines available and then, in a second *scheduling* stage, the algorithm learns the number of machines and the bags must be scheduled on the machines without being split up.

---

<sup>\*</sup>[eb3224@columbia.edu](mailto:eb3224@columbia.edu). Department of Industrial Engineering and Operations Research, Columbia University in the City of New York, New York, United States.

<sup>†</sup>[to2372@columbia.edu](mailto:to2372@columbia.edu). Department of Industrial Engineering and Operations Research, Columbia University in the City of New York, New York, United States.

<sup>‡</sup>[cliff@ieor.columbia.edu](mailto:cliff@ieor.columbia.edu). Department of Industrial Engineering and Operations Research, Columbia University in the City of New York, New York, United States.

<sup>§</sup>[hw2738@columbia.edu](mailto:hw2738@columbia.edu). Department of Industrial Engineering and Operations Research, Columbia University in the City of New York, New York, United States.

This model is motivated by applications where partial packing decisions have to be made with only partial information about the machines. As discussed in [11], such applications include MapReduce computations in shared data centers where data is partitioned into groups by a mapping function that is designed without full information about the machines that will be available in the data center, or in a warehouse where items are grouped into boxes without full information about the trucks that will be available to ship the items. Since it is not just the number of machines that might be unknown but also the speeds of these machines, Eberle et al. [3] study an extension of this model called speed-robust scheduling where the speeds of the machines are unknown in the partitioning stage and are revealed in the scheduling stage.

In this paper, we introduce and study the problem of scheduling with machine-learned predictions about the speeds of the machines. In the two applications mentioned above, MapReduce computations and package shipping, it is natural to have some relevant historical data about the computing resources or the trucks that will be available, which can be used to obtain machine-learned predictions about these quantities. In the scheduling with speed predictions problems, jobs must first be partitioned into bags given predictions about the speeds of the machines and then, when the true speeds of the machines are revealed, the bags must be scheduled on the machines without being split up. The goal is to use the predictions to design algorithms that achieve improved guarantees for speed-robust scheduling. The fundamental question we ask is:

*Can speed predictions be used to simultaneously obtain improved guarantees for scheduling when the predictions are accurate as well as bounded guarantees even when the prediction errors are arbitrarily large?*

## 2 Problem definition and results

We first describe the speed-robust scheduling problem proposed by Eberle et al.. There are  $n$  jobs with processing times  $\mathbf{p} = (p_1, \dots, p_n) \geq 0$  and  $m$  machines with speeds  $\mathbf{s} = (s_1, \dots, s_m) > 0$  such that the time needed to process job  $j$  on machine  $i$  is  $p_j/s_i$ .<sup>1</sup> The speed-robust scheduling problem is the following two-stage problem. In the first stage, called the partitioning stage, the speeds of the machines are unknown and the jobs must be partitioned into  $m$  bags  $B_1, \dots, B_m$  such that  $\cup_{i \in [m]} B_i = [n]$  (where  $[n] = \{1, \dots, n\}$ ) and  $B_{i_1} \cap B_{i_2} = \emptyset$  for all  $i_1, i_2 \in [m]$ ,  $i_1 \neq i_2$ . In the second stage, called the scheduling stage, the speeds  $\mathbf{s}$  are revealed to the algorithm and each bag  $B_i$  created in the partitioning stage must be assigned, i.e., scheduled, on a machine without being split up.

The objective considered in [3] is the classical makespan minimization objective, i.e., with  $\mathcal{M}_i$  being the collection of bags assigned to machine  $i$ , the goal is to

---

<sup>1</sup>The non-zero speed assumption is for ease of notation. Having a machine with speed  $s_i = 0$  is equivalent to  $s_i = \epsilon$  for  $\epsilon$  arbitrarily small since in both cases no schedule can assign any job to machine  $i$  without the completion time of this job being arbitrarily large.

minimize  $\max_{i \in [m]} (\sum_{B \in \mathcal{M}_i} \sum_{j \in B} p_j) / s_i$ . An algorithm for speed-robust scheduling is  $\beta$ -robust if it achieves an approximation ratio of  $\beta$  compared to the optimal schedule that knows the speeds in advance, i.e.,  $\max_{\mathbf{p}, \mathbf{s}} \text{alg}(\mathbf{p}, \mathbf{s}) / \text{opt}(\mathbf{p}, \mathbf{s}) \leq \beta$  where  $\text{alg}(\mathbf{p}, \mathbf{s})$  and  $\text{opt}(\mathbf{p}, \mathbf{s})$  are the makespans of the schedule returned by the algorithm (that learns  $\mathbf{s}$  in the second stage) and the optimal schedule (that knows  $\mathbf{s}$  in the first stage).

We augment the speed-robust scheduling problem with predictions about the speeds of the machines and call this problem Scheduling with Speed Predictions (SSP). The difference between SSP and speed-robust scheduling is that, during the partitioning stage, the algorithm is now given access to, potentially incorrect, predictions  $\hat{\mathbf{s}} = (\hat{s}_1, \dots, \hat{s}_m) > 0$  about the speeds of the machines. The true speeds of the machines  $\mathbf{s}$  are revealed during the scheduling stage, as in the speed-robust scheduling problem. We also want to minimize the makespan.

Consistency and robustness are two standard measures in algorithms with predictions [7]. An algorithm is  $\alpha$ -consistent if it achieves an  $\alpha$  approximation ratio when the predictions are correct, i.e., if  $\max_{\mathbf{p}, \mathbf{s}} \text{alg}(\mathbf{p}, \mathbf{s}, \mathbf{s}) / \text{opt}(\mathbf{p}, \mathbf{s}) \leq \alpha$  where  $\text{alg}(\mathbf{p}, \hat{\mathbf{s}}, \mathbf{s})$  is the makespan of the schedule returned by the algorithm when it is given predictions  $\hat{\mathbf{s}}$  in the first stage and speeds  $\mathbf{s}$  in the second stage. An algorithm is  $\beta$ -robust if it achieves a  $\beta$  approximation ratio when the predictions can be arbitrarily wrong, i.e., if  $\max_{\mathbf{p}, \hat{\mathbf{s}}, \mathbf{s}} \text{alg}(\mathbf{p}, \hat{\mathbf{s}}, \mathbf{s}) / \text{opt}(\mathbf{p}, \mathbf{s}) \leq \beta$ . We note that a  $\beta$ -robust algorithm for speed-robust scheduling is also a  $\beta$ -robust (and  $\beta$ -consistent) algorithm for SSP which ignores the speed predictions.

**Our results.** Our main result is an algorithm for the SSP problem that achieves a  $\min\{\eta^2(1 + \epsilon)^2(1 + \alpha), (1 + \epsilon)(2 + 2/\alpha)\}$ -approximation for any constants  $\alpha, \epsilon \in (0, 1)$ , where  $\eta = \max_{i \in [m]} \frac{\max\{\hat{s}_i, s_i\}}{\min\{\hat{s}_i, s_i\}}$  is the maximum prediction error between the predicted speed  $\hat{s}_i$  and the true speed  $s_i$  of the  $m$  machines. In particular, this implies that our algorithm is  $(1 + \epsilon)^2(1 + \alpha)$ -consistent and  $(1 + \epsilon)(2 + 2/\alpha)$ -robust. When the predictions are accurate, this approximation improves over the best-known approximation for speed-robust scheduling without predictions of  $2 - 1/m$  [3], while simultaneously maintaining a worst-case approximation of  $(1 + \epsilon)(2 + 2/\alpha)$  even when the predictions are arbitrarily wrong. In addition, we show that

- for the cases where the job sizes are equal or infinitesimal, our algorithm is still  $(1 + \epsilon)^2(1 + \alpha)$ -consistent, but it also achieves an improved robustness of  $(1 + \epsilon)(2 + 3/(2\alpha))$  and  $(1 + \epsilon)(1 + 3/(2\alpha))$  respectively;
- for any constant  $\alpha \in (0, 1)$ , any  $(1 + \alpha)$ -consistent algorithm has robustness that is at least  $1 + \frac{1-\alpha}{2\alpha} - O(\frac{1}{m})$ . If we ignore the constant factors in our result, our algorithm meets the optimal  $1/\alpha$  rate of increase of the robustness;
- even when the prediction errors are large, our algorithm often empirically outperforms existing speed-robust scheduling algorithms that do not use predictions.

## References

- [1] Susanne Albers and Guenter Schmidt. Scheduling with unexpected machine breakdowns. *Discrete Applied Mathematics*, 110(2-3):85–99, 2001.
- [2] Etienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15350–15359. Curran Associates, Inc., 2020.
- [3] Franziska Eberle, Ruben Hoeksma, Nicole Megow, Lukas Nölke, Kevin Schewior, and Bertrand Simon. Speed-robust scheduling - sand, bricks, and rocks. In *Integer Programming and Combinatorial Optimization - 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*, pages 283–296, 2021.
- [4] Leah Epstein, Asaf Levin, Alberto Marchetti-Spaccamela, Nicole Megow, Julian Mestre, Martin Skutella, and Leen Stougie. Universal sequencing on an unreliable machine. *SIAM Journal on Computing*, 41(3):565–586, 2012.
- [5] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 285–294, 2021.
- [6] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1859–1877, 2020.
- [7] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018.
- [8] Michael Mitzenmacher. Scheduling with Predictions and the Price of Misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:18, 2020.
- [9] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *arXiv preprint arXiv:2006.09123*, 2020.
- [10] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018.
- [11] Clifford Stein and Mingxian Zhong. Scheduling when you do not know the number of machines. *ACM Trans. Algorithms*, 2019.

# Non-Clairvoyant Scheduling with Predictions Revisited

Alexander Lindermayr (Speaker)\*

Nicole Megow\*

---

Non-clairvoyant scheduling is a fundamental problem and has been studied extensively in many variations [9, 7, 2, 5]. We consider non-clairvoyant scheduling problems with the objective of minimizing the sum of weighted completion times in different settings. We are given a set of jobs  $\{1, \dots, n\} =: [n]$ , each job  $j$  arriving online at its release date  $r_j$ , with an individual weight  $w_j$  and with an unknown processing requirement  $p_j$ . These jobs must be scheduled preemptively on a single or identical parallel machines. Using classical scheduling notation, we refer to these problems as the non-clairvoyant versions of  $1|pmtn|\sum w_j C_j$  and  $P|r_j, pmtn|\sum w_j C_j$ . We also consider non-clairvoyant scheduling on unrelated machines,  $R|r_j, pmtn|\sum w_j C_j$ , where jobs may have very different processing times on each of the machines, but a machine-dependent processing rate  $\ell_{ij}$  is given.

The assumption of non-clairvoyance seems too strong for many applications. While the exact processing time might be unknown, often some estimate is available, e.g., extracted information from past data is commonly used to predict the future. The recently emerging line of research on *learning-augmented* algorithms proposes to design algorithms that have access to additional (possibly erroneous) input, called a *prediction*. Ideally, the performance of a learning-augmented algorithm is a monotone function of the quality of the prediction for some error measure, which definition is a key task.

Recent work on non-clairvoyant scheduling with predictions [10, 11, 6] studies the problem  $1|pmtn|\sum C_j$  with predicted processing requirements  $\{y_j\}_{j \in J}$ , which we call *length predictions*. In their seminal paper [10], Kumar et al. propose an algorithm that is controlled by a parameter  $\lambda \in (0, 1)$ , which can be seen as an indicator of the algorithm’s trust in the accuracy of the prediction. Measuring the quality of a prediction  $\{y_j\}_{j \in J}$  w.r.t. the actual processing requirements  $\{p_j\}_{j \in J}$  by the  $\ell_1$  metric ( $\ell_1 = \sum_{j \in J} |p_j - y_j|$ ), they prove a competitive ratio of at most  $\min\{(1/(1 - \lambda))(1 + n\ell_1/\text{OPT}), 2/\lambda\}$ .

However, the  $\ell_1$ -metric does not seem to distinguish well between “good” and “bad” predictions, as has been noted recently by Im et al. [6]. They propose a different error measure  $\nu$  that satisfies certain desired properties and is based on the optimal solution of artificial instances mixing  $y_j$  and  $p_j$ . Using this error, they design a randomized algorithm with guarantees depending on  $\nu$ . Their algorithm is quite sophisticated, requires large constants to diverge from RR, and it seems

---

\*{linderal,nmegow}@uni-bremen.de. Faculty of Mathematics and Computer Science, University of Bremen, Germany.

very challenging to generalize it to scheduling settings with release dates, weights or even heterogenous machines. Further, the error measure  $\nu$  is still sensitive to changes in the predicted job lengths which would not affect an optimal schedule at all which seems an undesired property.

**Our contribution** In this work we contribute to non-clairvoyant scheduling with predictions in two ways: we propose a new prediction model and error definition and we present a framework for designing learning-augmented scheduling algorithms for more general scheduling settings, beyond the simple single-machine setting.

We propose a novel prediction model for scheduling problems, which we call *permutation prediction model*. In contrast to earlier works predicting processing requirements, we predict a permutation  $\hat{\sigma} : [n] \rightarrow [n]$  on the job set  $[n]$ , which suggests an algorithm a priority order for scheduling. For unrelated machines, we also include a job-to-machine assignment in the prediction model. The idea is that, instead of predicting job lengths, we take structural properties of an input instance into account that an optimal algorithm may exploit. Notice that for minimizing the sum of weighted completion time, the *Weighted Shortest Remaining Processing Time (WSPT)* order, i.e., jobs in order of their densities  $\mu_j = w_j/p_j$ , has proven to be useful in various settings. Indeed, for the non-clairvoyant version of  $1|pmtn|\sum w_j C_j$ , knowing the WSPT order of jobs would be sufficient to determine an optimal schedule. Therefore, we use this order to denote a *perfect* prediction  $\sigma$  of an instance. While this knowledge is not sufficient for optimally scheduling with release dates and/or on multiple machines, it still admits strategies with good approximations on an optimal solution [8, 1, 4].

Clearly, a WSPT-based permutation prediction could be derived from a length prediction. The advantage of our model is that it is much more compact, seems to capture a crucial structural property of an optimal solution and makes error measures less vulnerable to small noise in the prediction compared to the length prediction model.

As a key contribution, we define new and meaningful error measures  $\eta^S$  (for identical machine(s)) and  $\eta^R$  (for unrelated machines) that quantify the impact of an error in the prediction to an algorithm's cost explicitly in terms of the objective function. They have desirable properties, also in comparison to previous measures, and are efficiently learnable, in both theory and practice. Intuitively,  $\eta^S$  measures how much an inverted pair of jobs in  $\hat{\sigma}$  influences the objective value compared to  $\sigma$ . For example, on a single machine without release dates, if  $j$  and its successor  $j'$  in  $\hat{\sigma}$  are swapped in  $\sigma$ , the schedule that follows  $\hat{\sigma}$  pays an additional cost of  $w_j p_j$  but saves  $w_j p_{j'}$  compared to the schedule that follows  $\sigma$ . Since  $\sigma$  corresponds to the WSPT order, this value is non-negative. For the case of unrelated machines and individual release dates, we further include possible errors due to wrong machine assignments and preemptive scheduling into the more general definition  $\eta^R$ .

For the algorithmic part, we revisit the technique of time sharing introduced by Kumar et al. [10] and extend it to a general framework, which we call *Preferential Time Sharing*. As a main contribution, we give the first algorithm for

non-clairvoyant scheduling with predictions on unrelated machines with release dates and prove strong performance bounds, smoothly degrading with prediction quality.

**Theorem 1** *Preferential Time Sharing gives for every  $\lambda \in (0, 1)$  learning-augmented non-clairvoyant online algorithms on*

(i) *a single machine,  $1|pmtn| \sum w_j C_j$ , with a competitive ratio of at most*

$$\min \left\{ \frac{1}{1-\lambda} \left( 1 + \frac{\eta^S}{\text{OPT}} \right), \frac{2}{\lambda} \right\},$$

(ii)  *$m$  identical machines with release dates,  $P|r_j, pmtn| \sum w_j C_j$ , with a competitive ratio of at most*

$$\min \left\{ \frac{1}{1-\lambda} \left( 2 + \frac{\eta^S}{m \cdot \text{OPT}} \right), \frac{3}{\lambda} \right\}, \text{ and}$$

(iii) *unrelated machines with release dates,  $R|r_j, pmtn| \sum w_j C_j$ , with a competitive ratio of at most*

$$\min \left\{ \frac{1}{1-\lambda} \left( 5.8284 + \frac{\eta^R}{\text{OPT}} \right), \frac{128}{\lambda} \right\}.$$

Our framework requires a clairvoyant and a non-clairvoyant algorithm for a given scheduling problem, both of them must satisfy a certain monotonicity property. Then, we design a learning-augmented variation of the clairvoyant algorithm that admits a competitive ratio as a function of an error measure.

While we use non-clairvoyant algorithms as a black box from the literature, the new contribution lies in proving error-dependent competitive ratios for monotone clairvoyant algorithms that use predictions as input. This may require designing new algorithms. In particular, we show a competitive ratio of  $3 + 2\sqrt{2} \approx 5.8284$  for a natural Greedy algorithm that schedules jobs preemptively in WSPT order for the clairvoyant problem  $R|r_j, pmtn| \sum w_j C_j$ . This does not match the recent and best known deterministic bound of 3 [3], but our algorithm satisfies the desired properties of being error-sensitive and monotone.

## References

- [1] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241. SIAM, 2012.
- [2] Olivier Beaumont, Nicolas Bonichon, Lionel Eyraud-Dubois, and Loris Marchal. Minimizing weighted mean completion time for malleable tasks scheduling. In *IPDPS*, pages 273–284. IEEE Computer Society, 2012.

- [3] Marcin Bienkowski, Artur Kraska, and Hsiang-Hsuan Liu. Traveling repairperson, unrelated machines, and other stories about average completion times. In *ICALP*, volume 198 of *LIPICs*, pages 28:1–28:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- [4] Varun Gupta, Benjamin Moseley, Marc Uetz, and Qiaomin Xie. Greed works - online algorithms for unrelated machine stochastic scheduling. *Math. Oper. Res.*, 45(2):497–516, 2020.
- [5] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. *J. ACM*, 65(1):3:1–3:33, 2018.
- [6] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *SPAA*, pages 285–294. ACM, 2021.
- [7] Jae-Hoon Kim and Kyung-Yong Chwa. Non-clairvoyant scheduling for weighted flow time. *Inf. Process. Lett.*, 87(1):31–37, 2003.
- [8] Nicole Megow and Andreas S. Schulz. On-line scheduling to minimize average completion time revisited. *Oper. Res. Lett.*, 32(5):485–490, 2004.
- [9] Rajeev Motwani, Steven J. Phillips, and Eric Torng. Non-clairvoyant scheduling. *Theor. Comput. Sci.*, 130(1):17–47, 1994.
- [10] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *NeurIPS*, pages 9684–9693, 2018.
- [11] Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *NeurIPS*, pages 8042–8053, 2020.

# Contract Scheduling with Predictions

Spyros Angelopoulos (Speaker) \*      Shahin Kamali †

---

## 1 Introduction

One of the central objectives in the design of intelligent systems is the provision of *anytime* capabilities. In particular, applications such as medical diagnostic systems and motion planning require that the system outputs a reasonably efficient solution given the unavoidable constraints on computation time. *Anytime algorithms* [6] offer such a tradeoff between computation time and quality of the output.

Russell and Zilberstein [10] introduced a useful distinction between two different types of anytime algorithms. On the one hand, there is the class of *contract* algorithms, which are given the amount of allowable computation time (i.e, the intended query time) as part of the input. However, if the algorithm is interrupted at any point before this “contract time” expires, the algorithm may output a result that is meaningless. On the other hand, the class of *interruptible* algorithms consists of algorithms whose allowable running time is not known in advance, and thus can be interrupted (queried) at any given point throughout their execution.

Although less flexible than interruptible algorithms, contract algorithms are often easier to implement and maintain [5]. Hence a natural question arises: how can one convert a contract algorithm to an interruptible equivalent, and at which cost? This question can be addressed using a simple technique that consists of repeated executions of the contract algorithm with increasing runtimes (also called *lengths*). For example, consider a *schedule* of executions of the contract algorithm in which the  $i$ -th execution has length  $2^i$ . Assuming that an interruption occurs at time  $t$ , then the above schedule guarantees the completion of a contract algorithm of length at least  $t/4$ , for any  $t$ . The factor 4 measures the performance of the schedule, and quantifies the penalty due to the repeated executions.

Formally, given a contract algorithm  $A$ , a *schedule*  $X$  is an increasing sequence  $(x_i)$  in which  $x_i$  is the length of the  $i$ -th execution of  $A$ . We call the  $i$ -th execution of  $A$  in  $X$  the  $i$ -th *contract*, and we call  $x_i$  its *length*. The *acceleration ratio* of  $X$ , denoted by  $\text{acc}(X)$ , relates an interruption  $T$  to the length of the largest contract

---

\*spyros.angelopoulos@lip6.fr. CNRS and LIP6-Sorbonne University, 4 place Jussieu, Paris 75252, France.

†shahin.kamali@umanitoba.ca. Department of Computer Science, University of Manitoba, E2-EITC, Winnipeg, Manitoba R3T 2N2, Canada.

that has completed by time  $T$  in  $X$ , which we denote by  $\ell(X, T)$ :

$$\text{acc}(X) = \sup_T \frac{T}{\ell(X, T)} \quad (1)$$

Contract scheduling has been studied in a variety of settings. It has long been known that the schedule  $X = (2^i)$  has optimal acceleration ratio equal to 4 [10]. Many other settings have been studied; see e.g. [3] and references therein.

## 1.1 Our setting: contract scheduling with predictions

Previous work on this problem has mostly assumed that the interruption is unknown to the scheduler, and thus can be chosen adversarially. In practice, however, the scheduler may have a certain *prediction* concerning the interruption. E.g., in a medical diagnostic system, the expert may know that the system will be likely queried around the time the medical intervention will take place.

We study two settings that capture the above scenarios. In the first setting, there is a prediction  $\tau$  concerning the interruption  $T$ . In the second setting, the prediction is in the form of responses to  $n$  *binary queries*, where  $n$  is a specified parameter. For example, a binary query can be of the form “Will the interruption occur within a certain subset of the timeline?”. For both settings, the prediction is not necessarily trustworthy, and comes with an unknown *error*  $\eta$ .

The performance of the schedule is determined by two parameters: the first is the *robustness*, which is the worst-case acceleration ratio of the schedule assuming adversarial error (i.e., an adversary manipulating the prediction). The second is the *consistency* of the schedule, which is the acceleration ratio assuming that the prediction is error-free. In between these extremes, the acceleration ratio will be, in general, a function of the prediction error. This follows the recent framework in machine learning of robust *online computation* with predictions, as introduced in [8] for the caching problem, and later applied in [9] for other online problem

We make a further distinction between schedules that know an upper bound  $H$  on the prediction error, and those that do not. Namely, an *H-aware* schedule is evaluated under the assumption that  $\eta \leq H$ , for some known  $H$ , whereas an *H-oblivious* schedule makes no assumption about the range of error.

## 2 Results

We first consider the setting in which the prediction  $\tau$  is the interruption time  $T$ . This prediction comes with an *error*  $\eta \in [0, 1]$  such that  $T \in [\tau(1-\eta), \tau(1+\eta)]$ . We further say that the error is *positive* if  $T > \tau$ , otherwise it is called *negative*. We obtain a Pareto-optimal schedule by showing a reduction from an online problem known as *online bidding* [7]. This allows us to use, as black-box, a Pareto-optimal algorithm of [2], and obtain a schedule with the same ideal performance. But there are two complications: this schedule cannot tolerate *any* errors, and is also fairly complex. We give another simple schedule with the same robustness and

consistency, and thus also Pareto-optimal. We then show how to extend this schedule to the realistic setting in which  $\eta \neq 0$ , and we complement the positive results with lower bounds on the performance of any schedule. We summarize one of the main results below. Define

$$c_r = \frac{r - \sqrt{r^2 - 4r}}{2} \text{ and } b_r = \frac{r + \sqrt{r^2 - 4r}}{2},$$

**Theorem 1** *There exists an  $H$ -aware schedule that is  $r$ -robust, and has acceleration ratio at most  $\min\{\frac{c_r(1+\eta)}{(1-H)}, r\}$  for positive error, and at most  $\min\{\frac{c_r(1-\eta)}{(1-H)}, r\}$  for negative error. Moreover, for any  $H$  that satisfies the condition  $\frac{1+H}{1-H} < \sqrt{\frac{c_r+1}{c_r}} - \delta$ , for any fixed  $\delta > 0$ , the acceleration ratio of any  $H$ -aware  $r$ -robust schedule is at least  $\min\{\frac{c_r(1+H)}{1-H}, r\}$ .*

In the second part, we study the setting in which the prediction is in the form of responses to  $n$  binary queries, for some given parameter  $n$ , i.e., we would like to combine the advice of  $n$  binary experts. Thus, the prediction is an  $n$ -bit string, and the prediction error  $\eta \in [0, 1]$  is defined as the fraction of the erroneous bits in the string. First, we show an information-theoretic lower-bound on the best-possible consistency one can hope to achieve in this setting, assuming optimal robustness equal to 4. We then present and analyze a family of schedules, parameterized by the range of error that each schedule can tolerate. There are several challenges here: the analysis must incorporate several parameters such as the error  $\eta$ , the number of queries  $n$  and the desired robustness  $r$ . Moreover, we need to define queries that are realistic and have a practical implementation. To this end, each query is a *partition* query of the form “Does interruption  $T$  belong to  $\mathcal{T}$ ?”, where  $\mathcal{T}$  is a subset of the timeline.

The following theorem summarizes some of the main results obtained.

**Theorem 2** *For every  $r \geq 4$ , define  $K$  to be equal to  $\frac{2pn+1}{n}$ , and  $d$  to be equal to  $b_r$ , if  $r \leq (1+K)^2/K$ , and  $1+K$ , otherwise. Then there exists a schedule that is  $r$ -robust and has acceleration ratio at most  $\frac{d^{1+\frac{1}{n}+2p}}{d-1}$ , assuming  $\eta \leq p \leq 1/2$ . Furthermore, for any binary prediction  $P$  of size  $n$ , any 4-robust schedule has consistency at least  $2^{1+\frac{1}{2n}}$ .*

The full paper is available in [4].

## References

- [1] Spyros Angelopoulos. Further connections between contract-scheduling and ray-searching problems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1516–1522, 2015.
- [2] Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *Proceedings of the 11th International Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 52:1–52:15, 2020.

- [3] Spyros Angelopoulos and Shendan Jin. Earliest-completion scheduling of contract algorithms with end guarantees. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 5493–5499, 2019.
- [4] Spyros Angelopoulos and Shahin Kamali, Contract Scheduling With Predictions. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, pages 11726–11733, 2021.
- [5] Daniel S. Bernstein, Lev Finkelstein, and Shlomo Zilberstein. Contract algorithms and robots on rays: Unifying two scheduling problems. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1211–1217, 2003.
- [6] Mark Boddy and Thomas L. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artif. Intell.*, 67(2):245–285, 1994.
- [7] Marek Chrobak and Claire Kenyon-Mathieu. SIGACT news online algorithms column 10: Competitiveness via doubling. *SIGACT News*, 37(4):115–126, 2006.
- [8] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 3302–3311, 2018.
- [9] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, volume 31, pages 9661–9670, 2018.
- [10] Stuart J. Russell and Shlomo Zilberstein. Composing real-time systems. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 212–217, 1991.

# The rectangle covering bound on the extension complexity of small cut polytopes

Wouter Fokkema (Speaker) <sup>\*†</sup>      Matthias Walter <sup>\*‡</sup>

---

## 1 Introduction

The *cut polytope* of a graph is given by the convex hull of the incidence vector of all cuts in the graph. Cut polytopes have been widely studied. One application in which the cut polytope is used is in solving the NP-hard max-cut problem [1]. The max cut problem is sometimes considered in relation to scheduling problems [2]. Cut polytopes are also closely related to correlation polytopes, which are tightly connected to combinatorial problems in the foundations of quantum mechanics, and to the Ising spin model [3].

There exists a strong connection between stable set polytopes and cut polytopes. In [4], it is shown that every inequality for the cut polytope can be mapped to an inequality for a stable set polytope of an appropriate graph. The stable set problem occurs as a sub-problem in scheduling, for example in time discretization formulations. Therefore, fundamental results for cut polytopes can be relevant for scheduling models.

An *extended formulation* of a polytope  $P$  is another polytope  $Q$  such that  $P$  is a projection of  $Q$ . An example is shown in Figure 1. Extended formulations can be used to simplify optimization over a polytope, when the extended formulation can be described by fewer inequalities than the original polytope. The *extension complexity* is the minimum number of inequalities required to represent any extended formulation of  $P$ .

We investigate the exact extension complexity of (small) cut polytopes. We restrict ourselves to cut polytopes of complete graphs with  $n$  nodes, which we will refer to as the cut polytope of size  $n$  from here on.

The extension complexity of cut polytopes of size  $n$  has a well-known upper bound of  $2^{n-1}$ , which is the number of vertices. This upper bound corresponds to writing the polytope as a convex combination of its vertices. The first exponential lower bound by Fiorini et al. [5] was considered a breakthrough result in polyhedral combinatorics. The best known asymptotic lower bound was given in [6], and is

---

<sup>\*</sup>Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.

<sup>†</sup>k.w.fokkema@utwente.nl

<sup>‡</sup>m.walter@utwente.nl

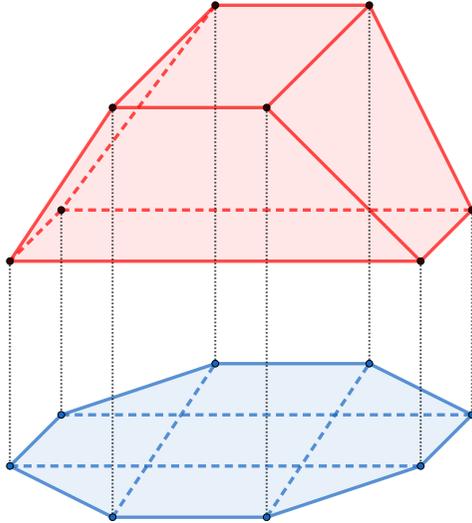


Figure 1: Example of an extended formulation. The 3-dimensional red polytope is an extended formulation of the 2-dimensional blue polytope. Notice that the blue polytope is defined by 8 inequalities and the red polytope by 6 inequalities.

equal to  $1.5^{n-1}$ . Until now, little was known about the exact value of the extension complexity, even for small cut polytopes. We compute this number for small  $n$  to shed light on the true behavior of this measure of complexity. Furthermore, results of our computations might give ideas for a theoretical proof of better asymptotic lower or upper bounds.

## 2 Our contribution

We computed lower bounds for the extension complexity of cut polytopes using the *rectangle covering bound*. This is a well known lower bound for the extension complexity, and is due to Yannakakis [7]. We can view this rectangle covering bound as acting on a binary matrix defined by the polytope. In this matrix, the rows are indexed by facets, the columns are indexed by vertices, and each matrix element is 0 if the corresponding vertex is contained in the corresponding facet, or 1 otherwise. The value of the rectangle covering bound on a matrix  $A$  is the minimum number of rank-1 binary matrices (called rectangles) such that the element-wise maximum of these matrices is equal to  $A$ . This can be viewed as covering the support of matrix  $A$  with rectangles, which gives the bound its name. An example is shown in Figure 2.

The rectangle covering bound was used by Fiorini et al. to show that the cut polytope and the TSP polytope have exponential extension complexity [5], and by Volker Kaibel and Stefan Weltge in their proof that the cut polytope has an

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} = \max \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \right\}$$

Figure 2: Example of a rectangle covering. The matrix on the left hand side is the element-wise maximum of the matrices on the right hand side. It is easy to prove that we also need at least 3 rectangles here, which implies that the rectangle covering bound of this matrix is 3.

extension complexity of at least  $1.5^{n-1}$ [6].

The rectangle covering bound can be relaxed to the *fractional rectangle covering bound*, which allows covering the matrix with rectangles that only count partially, both towards the value of the bound and towards covering elements. In the fractional rectangle covering bound, the fractions of rectangles covering each nonzero element of  $A$  must sum up to at least 1. The fractional rectangle covering bound is a lower bound on the rectangle covering bound.

We have modeled the rectangle covering bound as a mixed integer program. The linear programming relaxation of this program gives the fractional rectangle covering bound. We interpret the dual of this linear program and use the fact that solutions to this dual are again lower bounds for the fractional rectangle covering bound. Computing the value of this linear program, we obtain the following result:

**Theorem 1** *The extension complexity of cut polytopes of size  $n \leq 9$  is exactly equal to  $2^{n-1}$ .*

To achieve this result, we use several techniques to reduce computational complexity. Firstly, we consider a submatrix of the slack matrix of the cut polytope, by only considering certain *classes* of facets, called *pure hypermetric facets* [8]. For illustration, the cut polytope of size 9 is conjectured to have over  $12 \cdot 10^{12}$  facets [9], but we consider only 4912 of those. Furthermore, we use the symmetries of the cut polytope and the pure hypermetric facets, by arguing that any solution to the dual linear program can be made symmetric. We use this fact to remove symmetric constraints in the dual linear program, which greatly reduces the total number of constraints. Each of these constraints corresponds to a rectangle in our matrix, so this technique implies enumerating non-symmetric rectangles. Finally, when enumerating these rectangles we only consider rectangles that are not contained in any other rectangle.

After this, we construct rectangle coverings and fractional rectangle coverings for our submatrix. Using these constructions we prove the following limits to our approach:

**Theorem 2** *The fractional rectangle covering bound of the submatrix of pure hypermetric facets of the cut polytope of size  $n > 9$  is smaller than  $2^{n-1}$ .*

**Theorem 3** *The rectangle covering bound of the submatrix of pure hypermetric facets of the cut polytope of size  $n \geq 93$  is smaller than  $2^{n-1}$ .*

Therefore, to be able to prove via the rectangle covering bound that the extension complexity of the cut polytope is equal to  $2^{n-1}$  we would need to consider more classes of facets than the pure hypermetric facets. For our approach, this becomes computationally intractable, because other classes of facets contain more facets and have fewer symmetries than the pure hypermetric facets.

## References

- [1] Francisco Barahona and Ali Ridha Mahjoub. On the cut polytope. *Mathematical Programming*, 36(2):157–173, 1986.
- [2] Matthias Elf, Michael Jünger, and Giovanni Rinaldi. Minimizing breaks by maximizing cuts. *Operations Research Letters*, 31(5):343–349, 2003.
- [3] Itamar Pitowsky. Correlation polytopes: Their geometry and complexity. *Mathematical Programming*, 50(3):395–414, 1991.
- [4] Svatopluk Poljak Monique Laurent and Franz Rendl. Connections between semidefinite relaxations of the max-cut and stable set problems. *Mathematical Programming*, 77:225–246, 1997.
- [5] Samuel Fiorini, Serge Massar, Sebastian Pokutta, Hans Tiwary, and Ronald Wolf. Linear vs. semidefinite extended formulations: Exponential separation and strong lower bounds. *Proceedings of the Annual ACM Symposium on Theory of Computing*, 11 2011.
- [6] Volker Kaibel and Stefan Weltge. A short proof that the extension complexity of the correlation polytope grows exponentially. *Discrete & Computational Geometry*, 53(2):397–401, 2014.
- [7] Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441–466, 1991.
- [8] Michel Deza and Monique Laurent. Facets for the cut cone I. *Mathematical Programming*, 56:121–160, 8 1992.
- [9] Thomas Christof. SMAPO – Library of linear descriptions of small problem instances of polytopes in combinatorial optimization. Available at <http://comopt.ifi.uni-heidelberg.de/software/SMAPO/cut/cut.html>.

# Randomized Cup Game Algorithms Against Strong Adversaries (SODA '21)

Michael A. Bender \*      William Kuszmaul (Speaker) †

---

In the *cup emptying game*, there are  $n$  cups, each initially empty. In each step of the game, the *filler* distributes up to  $1 - \epsilon$  water among the cups, where  $0 \leq \epsilon < 1$  is the *resource-augmentation parameter*. The *emptier* then selects one cup and removes a unit of water from that cup (or all of the water in the cup, if the cup contains less than one unit). The emptier's goal is to minimize the *backlog* of the system, which is defined to be the height of the fullest cup. This corresponds naturally to a scheduling problem in which the cups represent threads, the water represents work that needs to be completed by each thread, and the emptier is a scheduler that perform up to 1 unit of work per time step.

Beginning in the 1960s [11, 12], much of the work on cup emptying games focused on *fixed-rate games*, in which each cup  $j$  receives the same amount of water  $f_j$  at every step (see the full version [6] of this paper for a detailed literature review). Baruah et al. [3], in particular, give an algorithm which achieves  $O(1)$  backlog in the fixed-rate game, and runs in polynomial time per step of the game.

In the past thirty years, researchers have turned their focus to the cup emptying game without the fixed-rate constraint. Adler et al. [1] showed that the simple *greedy emptying algorithm*, in which the emptier always removes water from the fullest cup, guarantees backlog  $O(\log n)$ . Moreover, the greedy emptying algorithm is known to be the optimal deterministic algorithm [1, 7]. Randomized algorithms, on the other hand, can achieve significantly smaller backlog [5, 8]. Bender et al. [5] showed that, as long as  $\epsilon \geq \frac{1}{\text{polylog } n}$ , then a randomized version of the greedy emptying algorithm can achieve backlog  $O(\log \log n)$  at each step with high probability in  $n$ , which is known to be the optimal backlog guarantee for randomized algorithms [5, 8]. Subsequent work by Kuszmaul [10] extends

---

\*bender@cs.stonybrook.edu. Stony Brook University, Stony Brook, NY 11794-2424 USA. Supported in part by NSF grants CCF-1725543, CSR-1763680, CCF-1716252, CCF-1617618, CNS-1938709, and XPS-1533644.

†kuszmaul@mit.edu. MIT CSAIL, 32 Vassar St, Cambridge, MA 02139 USA. Supported in part by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. The author was also supported in part by an NSF Graduate Fellowship and a Hertz Foundation Fellowship.

these guarantees to arbitrarily small  $\epsilon$  (even  $\epsilon = 0$ ) for games whose lengths are polynomially bounded.

The power of randomization comes from the fact that the filler is required to be an *oblivious adversary*, meaning that it must decide at the beginning of the game how much water to put into each cup at each step. Randomness in the emptying algorithm can therefore be used to add unpredictability to the emptier’s behavior, making it difficult for the filler to anticipate how much water is in each cup at a given step. The best known randomized algorithms against an oblivious adversary work by adding small “random perturbations” to the behavior of the standard greedy algorithm.

If, instead, the filler is an adaptive adversary, then no randomized emptying algorithm can achieve backlog  $o(\log n)$  [5, 7, 8]. Rather, the filler can achieve backlog  $\Theta(\log n)$  with the following simple  $(n - 1)$ -step construction. In the first step, the filler spreads 1 unit of water equally among the  $n$  cups. In the next step, the filler distributes 1 unit of water equally among the  $n - 1$  cups that have not yet been emptied. In each subsequent step  $t$ , the filler spreads 1 unit of water equally among  $n - t + 1$  cups that have not yet been touched by the emptier. After the  $(n - 1)$ -th step, one cup remains with  $1/n + 1/(n - 1) + \dots + 1/2 = \Theta(\log n)$  water. The fact that the filler is an adaptive adversary enables the filler to determine after each step which cups have not yet been touched by the emptier.

**Cup games with cyclic feedback.** Modeling the filler as an oblivious adversary makes sense as long as the arrival of water across steps is independent of the emptying decisions in previous steps. This is not true in many natural applications of cup games, however. Often, the work that is cleared from buffers on one step dictates which work arrives into buffers in subsequent steps. For example, in scheduling/queuing applications, the threads that are scheduled on a given step determine the tasks that are added to the queues on the next step. Similarly, in data-structure applications, where cup games schedule tasks within a data structure, the same type of cyclic feedback can occur [4].

As soon as there is cyclic feedback within the system, the filler can no longer be modeled as oblivious. This means that the randomized algorithms for the cup game [5, 8, 10] cannot be applied directly to the vast majority of the cup game’s applications.

In general, systems that exhibit cyclic feedback can often behave poorly under stress, even when the cyclic feedback is not *explicitly* malicious. A classic example of this phenomenon is *cache stampedes* which occur in caches on distributed databases: when the lease on a frequently accessed element of a cache expires, multiple requests to that item may trigger cache misses and start repopulating that item at the same time—by bogging down the system, this can then cause additional cache items to expire, resulting in a cyclic positive feedback loop [2, 14]. Another example of this phenomenon is the *double-disk failure problem*: when data is stored redundantly on two drives, the two drives often fail at close to the same time—this is because (a) the two drives experience similar access patterns over their lifetime, and because (b) the failure of one drive implicitly causes a

sudden unusual amount of activity on the other (due to rebuild efforts), thereby increasing the probability of failure on the second drive [9, 13].

When a system is under stress, the cyclic feedback between components of the system can have critical effects on the system’s behavior. For a cup game, a cup is “under stress” when it has significantly more water than average.

Thus we have the following natural question: in order for strong randomized guarantees to be possible, do we really need the adversary to be fully oblivious, or could we instead handle an adversary that has some information hidden from them, but that is able to detect which cups are under stress?

**This paper: strong guarantees against a nearly adaptive adversary.** To address this question, we consider an adversary that is fully adaptive *but that sees the fills of the cups at a low precision*: whenever a cup is nearly empty, containing 3 or fewer units of water, the adversary cannot see the precise amount of water that is in the cup. One can think of this adversary as being separated from the cups by a wall of height 3—we call this an *elevated adaptive adversary*,

We show that randomized emptying algorithms can be made effective against elevated adaptive adversaries, achieving the same bound on backlog as if the adversary were oblivious. In particular, we prove that, if the filler is an elevated adaptive adversary, and the emptier has resource augmentation  $\epsilon \geq \frac{1}{\text{polylog } n}$ , then there is a randomized emptying algorithm that, at any step  $t$ , achieves  $O(\log \log n)$  backlog with high probability in  $n$ . This guarantee continues to hold for each step  $t$ , even when  $t$  is arbitrarily large.

The elevated adaptive adversary does not necessarily apply out-of-the box to every application of cup games. For example, in scheduling applications, the decision of which cup is emptied affects which cups receive water in the next step, even if the emptied cup was under very low stress. On the other hand, our algorithm only removes water from cups that are under very low stress when the state of the system as a whole is good (i.e., all cups are under very low stress). This means that the elevated adaptive adversary *does* eliminate the possibility of a feedback loop within the system causing the system to perpetually travel towards a worse and worse state in terms of its backlog. That is, the only way that a filler can successfully behave maliciously is if that malicious behavior occurs when all of the cups in the system are under low stress.

## References

- [1] M. Adler, P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. Goldberg, and M. Paterson. A proportionate fair scheduling rule with good worst-case performance. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 101–108, 2003.
- [2] M. I. Alipio and N. M. C. Tiglao. Towards a taxonomy of cache-based transport protocols in wireless sensor networks. In *2017 IEEE Sensors Applications Symposium (SAS)*, pages 1–6. IEEE, 2017.

- [3] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, Jun 1996.
- [4] M. Bender, R. Das, M. Farach-Colton, R. Johnson, and W. Kuszmaul. Flushing without cascades. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.
- [5] M. Bender, M. Farach-Colton, and W. Kuszmaul. Achieving optimal backlog in multi-processor cup games. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, 2019.
- [6] M. A. Bender and W. Kuszmaul. Randomized cup game algorithms against strong adversaries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2059–2077. SIAM, 2021.
- [7] P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–372, 1987.
- [8] P. F. Dietz and R. Raman. Persistence, amortization and randomization. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 78–88, 1991.
- [9] S. Howard. Double disk failure, 2015.
- [10] W. Kuszmaul. Achieving optimal backlog in the vanilla multi-processor cup game. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.
- [11] C. L. Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary, 1969*, 1969.
- [12] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.
- [13] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? *File and Storage Technologies (FAST)*, 7(1):1–16, 2007.
- [14] A. Vattani, F. Chierichetti, and K. Lowenstein. Optimal probabilistic cache stampede prevention. *Proceedings of the VLDB Endowment*, 8(8):886–897, 2015.

# Multi-Purpose Machine Scheduling: An Application to Smart Cosmetic Manufacturing

Michele Barbato (Speaker) \*   Antonio Belotti †   Alberto Ceselli ‡  
Giovanni Righini §

---

We consider a problem arising in the context of smart manufacturing, and more specifically in the cosmetics industry [1].

The industrial partner produces, on a daily basis, a set of products. Before the actual production starts the necessary ingredients of each product must be gathered in the right quantities. In general, the same ingredient may be required by more than one product, a product may require more than one ingredient, also of the same type. The ingredients are stored in several boxes. Each box may contain multiple ingredients, and the same ingredient may appear in multiple boxes. Ingredients cannot be moved from one box to another.

For each product, the operations of *processing* (that is, weighing and grouping) all its required ingredients are accomplished by exactly one worker. Every worker can hold at most one box at a time. The *processing time* is the amount of time needed to process an ingredient for a specific product. Each worker may have more than a single product to process in its duty. In such a case, they can be interleaved and processed partly before completion. Each box can be in use of a single worker at a time. They can be moved from one worker to another by means of a single automated guided vehicle (AGV). The AGV can carry one box at a time. The *setup time* is the time needed to move boxes between workers.

The goal of the problem is to find a sequence (over time) of assignments of the boxes to the workers so to minimize the *makespan*, *i.e.*, the time needed to process all ingredients of all products taking into account both processing and setup times.

**Contributions and outline.** In this paper we operate the assumption that both the setup times and the processing times of all ingredients are equal to 1 for all workers. We formalize the problem, proving it to be **NP-hard**. Then, we consider

---

\*[michele.barbato@unimi.it](mailto:michele.barbato@unimi.it). Dipartimento di Informatica, Università degli Studi di Milano, via Celoria 18, 20133 Milano, Italy.

†[antonio.belotti@studenti.unimi.it](mailto:antonio.belotti@studenti.unimi.it). Dipartimento di Informatica, Università degli Studi di Milano, via Celoria 18, 20133 Milano, Italy.

‡[alberto.ceselli@unimi.it](mailto:alberto.ceselli@unimi.it). Dipartimento di Informatica, Università degli Studi di Milano, via Celoria 18, 20133 Milano, Italy.

§[giovanni.righini@unimi.it](mailto:giovanni.righini@unimi.it). Dipartimento di Informatica, Università degli Studi di Milano, via Celoria 18, 20133 Milano, Italy.

a relaxation, mapping to a scheduling problem with multi-purpose machines. We show how a slight variant of an algorithm from the literature allows its resolution in polynomial time.

## 1 Definition, Representation and Complexity of the Problem

Let  $\mathcal{T}$  denote a set of *ingredients*. We are given a set of *boxes*  $\mathcal{M} = \{M_1 \dots M_m : M_i \subseteq \mathcal{T} \text{ for } i = 1 \dots m\}$ , where  $t \in M_i$  if box  $M_i$  contains ingredient  $t$ . We are also given a set of *products*  $\mathcal{J} = \{J_1 \dots J_n : J_k \subseteq \mathcal{T} \text{ for } k = 1 \dots n\}$ , where  $t \in J_k$  if product  $J_k$  requires ingredient  $t$ . That is, each box and each product is a set of ingredients.

Our *cosmetics scheduling problem* (CSP) can be modeled as follows. Each box  $m \in \mathcal{M}$  acts as a *machine* that can stay in one of three states: being used by a worker for a specific ingredient, being transported by the AGV or being idle. Each product  $J \in \mathcal{J}$  acts as a *job*. Each ingredient  $t$  required in a product  $J$  acts as a *task* of the corresponding job.

In this abstract we assume that each worker has a single product in duty: only minor adaptations are needed to handle duties of more products.

A feasible CSP solution consists of a feasible scheduling of jobs  $\mathcal{J}$  to the machines  $\mathcal{M}$ , that is, a scheduling satisfying the following rules

- tasks of the same job can be processed in any order, but cannot be processed simultaneously,
- each machine can process one task at a time
- each machine can process only tasks of ingredients which are in the corresponding box, *i.e.*,  $M \in \mathcal{M}$  can process  $t \in J$  only if  $t \in M$ ,
- whenever a machine  $M$  switches from one job to another, a setup operation is needed on  $M$  (setup is not needed for the first job which is processed),
- at most one setup operation can be performed on all machines at a time.

A feasible CSP solution is optimal if it minimizes the makespan, that is, the completion time of the job that completes last.

## 2 Results

We observe that the key complicating factor of the problem is the presence and the structure of the setup operations in a schedule. In fact, it makes the problem theoretically difficult, as shown in the following proposition.

**Proposition 1** *The cosmetics scheduling problem is NP-hard.*

We reduce from the SET COVERING problem, which is **NP**-complete [5]. A formal proof is omitted in this abstract, but its structure relies on mapping of the SET COVERING ground set to ingredients  $\mathcal{T}$  and the covering sets to boxes  $\mathcal{M}$ . Dummy elements need to be added to reach a suitable mapping. Finally, products  $\mathcal{J}$  take copies of dummy elements and elements of the ground set. The number of setups in an optimal scheduling corresponds to the number of subsets used in a Set Covering solution.

In view of this result, we seek for a relaxation of the CSP that can be solved in polynomial time by relaxing the structure of setups. In particular, we first focus on the *setup-free relaxation*, that is a version of CSP in which no setup is needed. The objective is not changing, and any feasible CSP solution is valid for the setup-free relaxation (but the converse might not be true), thus making it valid.

The setup-free relaxation is a variant of *open shop problem with multi-purpose machines* [4] (OSP-MPM) as defined in [6] in which all processing times are equal to 1. It is surprising to notice that recent literature on related problems is very scarce. For instance, they are *not* covered in recent reviews like [2].

**Proposition 2** *The setup-free relaxation can be solved in polynomial time.*

Our proof is constructive: we provide a polynomial time algorithm. Its main idea is a slight variant of that reported in [6]. It is based on the computation of maximum flows in a sequence of acyclic networks with integer capacities. Networks in this sequence are built by fixing, in an iterative way, a tentative maximum number of tasks  $B$  for each machine. It can be enforced by encoding graph capacities.  $B$  is clearly defining a tentative makespan. We prove that, by fixing  $B$ , flows are feasible if and only if a feasible schedule of makespan  $B$  exists.

Once an optimal makespan value  $B^*$  is found, we prove that a feasible schedule of makespan  $B^*$  can be found accordingly, by solving an edge coloring problem on a bipartite hyper graph. We point out that a similar approach is used to determine the optimal schedule of classical open-shop scheduling problems with unit processing times and parallel machines [3, p. 161]. The main modification of our approach is the presence of parallel edges in the bipartite graph on which the edge coloring is computed.

Finally, we investigate on ways to improve the lower bounds given by the setup-free relaxation, by re-introducing setups and progressively imposing more restrictions on them. We present our ongoing research in this context, and also discuss some open research question, like which relaxations of setups make the problem polynomially solvable, and which produce nontrivial bounds.

## References

- [1] AD-COM PROJECT WEBSITE <https://ad-com.net/> (accessed Feb. 10th 2022).

- [2] M.M. AHMADIAN, M. KHATAMI, A. SALEHIPOUR AND T.C.E. CHENG (2021). *Four decades of research on the open-shop scheduling problem to minimize the makespan*, European Journal of Operational Research, 295, 399–426
- [3] P. BRUCKER (2007). *Scheduling algorithms*, Springer, Heidelberg.
- [4] R. DE FREITAS RODRIGUES, M.C. DOURADO AND J.L. SZWARCFITER (2014). *Scheduling problem with multi-purpose parallel machines*, Discrete Applied Mathematics, 164, 313–319
- [5] M.R. GAREY AND D.S. JOHNSON (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.
- [6] B. JURISCH (1992). *Scheduling jobs in shops with multi-purpose machines* Ph.D. Thesis

# Operating rooms scheduling with a shared resource: a red-blue knapsack modeling approach

Federico Della Croce \*    Andrea Grosso (speaker) †    Vincent T'kindt ‡

---

## 1 Introduction

A practical operating rooms scheduling problem, which arises at a public hospital located in Northern Italy is considered. It consists in deriving a daily schedule for a set of operating rooms where each operating room is associated with a surgical specialty and a set of surgical interventions to be performed for the considered day. We denote this problem as the Shared Resource Operating Rooms scheduling (SRORS) problem. Operating rooms scheduling has been widely investigated in the last 50 years. We mention here four main reference surveys on the topic published between 2010 and 2019, namely [1], [2], [3] and [4].

The SRORS problem can be formally expressed as follows. We have  $n$  surgical interventions (hereafter also denoted as jobs)  $1, \dots, n$  and  $m$  operating rooms  $OR_1, \dots, OR_m$ . Each job  $j$  belongs to a specific surgical specialty that is univocally assigned to a given operating room  $OR_i$ . Then, for each operating room  $OR_i$ , there are  $n_i$  jobs to schedule in the considered day, and we have  $n = \sum_{i=1}^m n_i$ . We denote by  $J_i$  the set of jobs schedulable in operating room  $OR_i$ . Each operating room  $OR_i$  is available for a common duration  $W$  for surgical interventions. Each job  $j$  may or may not require the use of a shared resource. We will denote as *red* jobs ( $j \in R$ ) those jobs requiring the shared resource and as *blue* jobs ( $j \in B$ ) the other jobs. Also, for each  $OR_i$ , we will denote as  $R_i$  ( $B_i$ ) the set of relevant red (blue) jobs. Each job  $j$  schedulable in  $OR_i$  has a nominal duration  $w_{i,j}$  and a profit  $p_{i,j}$  directly derived from the job priority. The goal is to maximize the total profit of the scheduled jobs. Let us also define as  $\text{SRORS}_{\min}$ , the minimization version of the problem where the goal is to minimize the total profit of the rejected jobs. To avoid unnecessary moves of the shared resource from one operating room to another, it is assumed that every day this resource can spend at most one single time interval in each operating room. Correspondingly, for each  $OR_i$ , w.l.o.g. a sequence  $\beta_i^1 \rho_i \beta_i^2$  holds, where  $\beta_i^1$  is a first subset of blue jobs,  $\rho_i$  is a subset of red

---

\*federico.dellacroce@polito.it. DIGEP Politecnico di Torino, Italy,

†grosso@di.unito.it. D.I. Università di Torino, Italy,

‡tkindt@univ-tours.fr. LIFAT University of Tours, France.

jobs and  $\beta_i^2$  is a second subset of blue jobs. Notice that all these subsets may be empty and that is irrelevant the order of the jobs within each jobs subset.

## 2 Problem complexity

Being a generalization of the 0/1 Knapsack Problem (see, for instance, [5]), the SRORS problem is *NP*-Hard, but it is of interest to determine whether it is in the weak or in the strong sense.

**Theorem 1** *We have the following results:*

1. *For the case when  $m = 2$ , the SRORS problem is weakly NP-hard,*
2. *For the case when  $m$  is fixed, the SRORS problem is weakly NP-hard,*
3. *For the case of an arbitrary  $m$ , the SRORS problem is strongly NP-hard and the  $SRORS_{\min}$  problem does not have a polynomial-time approximation algorithm with a bounded approximation ratio unless  $P = NP$ .*

Proof: (Sketch): Case 1 (resp. 2) is proved by showing the existence of a DP algorithm running in  $O(nW)$  time (resp.  $O(nW^2)$  time) to solve it.

Case 3 is proved by reduction from 3-Partition where the approximation result relates to a  $SRORS_{\min}$  problem where no job is rejected IFF there exists a solution of 3-Partition.

## 3 ILP formulation and solution approach

To solve to optimality the SORS problem, we provide an improved knapsack-based formulation as follows. For each job  $j \in J_i$ , we introduce two binary variables  $x_{i,j}^1$  and  $x_{i,j}^2$  if job  $j \in B_i$  (blue job), or a binary variable  $y_{i,j}$  if job  $j \in R_i$  (red job). For blue jobs,  $x_{i,j}^1 = 1$  ( $x_{i,j}^2 = 1$ ) if job  $j$  is selected and is placed in the first (second) subset  $\beta_i^1$  ( $\beta_i^2$ ), alternatively  $x_{i,j}^1 = 0$  ( $x_{i,j}^2 = 0$ ). Similarly, for red jobs,  $y_{i,j} = 1$  if job  $j$  is selected and is placed in subset  $\rho_i$ , alternatively  $y_{i,j} = 0$ .

Then, for each operating room  $OR_i$ , we introduce two continuous variables  $s_i \geq 0$  and  $e_i \geq 0$  denoting the start time and end time, respectively, of subset  $\rho_i$  of red jobs. Finally, for each pair  $O_i, O_k$  (with  $i < k$ ) of operating rooms, we introduce a binary variable  $z_{i,k}$  where  $z_{i,k} = 1$  if  $e_i \leq s_k$ , else  $z_{i,k} = 0$ .

The following ILP model holds.

$$\max \sum_{i \in [1, \dots, m], j \in B_i} p_{i,j}(x_{i,j}^1 + x_{i,j}^2) + \sum_{i \in [1, \dots, m], j \in R_i} p_{i,j}y_{i,j} \quad (1)$$

$$\sum_{j \in B_i} w_{i,j}x_{i,j}^1 \leq s_i, \quad \forall i \in [1, \dots, m] \quad (2)$$

$$\sum_{j \in R_i} w_{i,j} y_{i,j} \leq e_i - s_i, \quad \forall i \in [1, \dots, m] \quad (3)$$

$$\sum_{j \in B_i} w_{i,j} x_{i,j}^2 \leq W - e_i, \quad \forall i \in [1, \dots, m] \quad (4)$$

$$\sum_{i \in [1, \dots, m], j \in R_i} w_{i,j} y_{i,j} \leq W \quad (5)$$

$$s_i \leq e_i \quad \forall i \in [1, \dots, m] \quad (6)$$

$$x_{i,j}^1 + x_{i,j}^2 \leq 1 \quad \forall j \in B_i \quad \forall i \in [1, \dots, m] \quad (7)$$

$$e_i \leq s_k + W(1 - z_{i,k}) \quad \forall i, k \in [1, \dots, m], i \neq k \quad (8)$$

$$e_k \leq s_i + W z_{i,k} \quad \forall i, k \in [1, \dots, m], i \neq k \quad (9)$$

$$x_{i,j}^1 \in \{0, 1\} \quad \forall i \in [1, \dots, m], \quad \forall j \in B_i \quad (10)$$

$$x_{i,j}^2 \in \{0, 1\} \quad \forall i \in [1, \dots, m], \quad \forall j \in B_i \quad (11)$$

$$y_{i,j} \in \{0, 1\} \quad \forall i \in [1, \dots, m], \quad \forall j \in R_i \quad (12)$$

$$z_{i,k} \in \{0, 1\} \quad \forall i, k \in [1, \dots, m], i \neq k \quad (13)$$

The considered ILP model can be enhanced in several ways. A preprocessing step for eliminating all dominated jobs can be applied. Recalling that, given a job  $j$ , every job  $k$  with larger profit and smaller weight is preferable to  $j$ , we can show that a red job  $j \in R_i$  is dominated (hence  $y_{i,j} = 0$ ) whenever

$$\sum_{k \in R_i \cup B_i: k \neq j, w_{i,k} \leq w_{j,k}, p_{i,k} > p_{j,k}} w_{i,k} > W - w_{j,k} + 1$$

namely there exists a set of jobs in  $R_i \cup B_i$  that are preferable to  $j$  such that the sum of their durations exceeds  $W - w_{j,k}$ .

Similarly, we can show that a blue job  $j \in R_i$  is dominated (hence  $x_{i,j}^1 + x_{i,j}^2 = 0$ ) whenever

$$\sum_{k \in B_i: k \neq j, w_{i,k} \leq w_{j,k}, p_{i,k} > p_{j,k}} w_{i,k} > W - w_{j,k} + 1$$

We tested the above ILP formulation constructing realistic instances from the considered hospital data. Each job (intervention)  $j$  assigned to operating room  $OR_i$  was assigned a nominal duration  $w_{i,j}$  (in minutes) generated in the uniform interval  $[30, 240]$ . The nominal duration embeds also the setup time required for cleaning the operating room before the entrance of the following patient. Each job  $j$  assigned to operating room  $OR_i$  was randomly given a priority  $pr_j \in \{5, 2, 1\}$  representing the urgency of the intervention: correspondingly, a nominal profit  $p_{i,j} = pr_j w_{i,j}$  was generated inducing strongly correlated instances. Each job  $j$  was given a probability of requiring the use of the shared resource randomly selected in the interval  $[10\%, 30\%]$ . Instances were generated with 10 or 20 jobs for each operating room and 5 or 10 operating rooms. Larger instances do not seem to be realistic. The considered time horizon was either 360 minutes or 720 minutes and 10 instances were considered for each pair  $[\# \text{ jobs}, \# \text{ operating rooms}]$ .

CPLEX 20.1.0.0 applied to the proposed ILP formulation was capable of solving to optimality all instances within a timelimit of 30 seconds when the time horizon was 360 minutes. With the 720 minutes horizon, CPLEX failed to reach optimality within a timelimit of 60 seconds in 3 of the  $20 \times 10$  instances. Still, at the timelimit for the unsolved instances, the gap between upper and lower bounds was negligible. Hence, from a practical point of view, the proposed approach is viable. We also explored another approach exploiting a genuine combinatorial generation of all relevant schedules for each operating room  $OR_i$ : let  $\mathcal{S}_i$  be the set of schedules of the form  $\beta_i^1 \rho_i \beta_i^2$  for  $OR_i$ . By means of dynamic programming, it is possible to generate all sets  $\mathcal{S}_i$ . Being able to enumerate the optimal red/blue schedules for each machine suggests a column-generation approach. Many columns turn out to be dominated and can be discarded. Usually not many undominated columns are left (can be thousands, but manageable). Then via a simple master problem it is possible to compose the optimal solution. Experimental results will show the efficiency of this latter approach with respect to the first one.

## Acknowledgements

This work has been partially supported by "Ministero dell'Istruzione, dell'Università e della Ricerca" Award "TESUN-83486178370409 finanziamento dipartimenti di eccellenza CAP. 1694 TIT. 232 ART. 6".

## References

- [1] Cardoen B., Demeulemeester E., Belien J. 2010, "Operating room planning and scheduling: a literature review", *European Journal of Operational Research*, Vol. 201 (3), pp. 921-932.
- [2] Guerriero F., Guido R. 2011, "Operational research in the management of the operating theatre: a survey", *Health Care Management Science*, Vol. 14 (1), pp. 89-114.
- [3] Samudra M., Van Riet C., Demeulemeester E., Cardoen B., Vansteenkiste N., Rademakers F.E. 2016, "Scheduling operating rooms: achievements, challenges and pitfalls", *Journal of Scheduling*, Vol. 19, pp.493-525.
- [4] Zhu S., Fan W., Yang S., Pei J., Pardalos P.M. 2019, "Operating room planning and surgical case scheduling: a review of literature", *Journal of Combinatorial Optimization*, Vol. 37, pp. 757-805.
- [5] Pferschy U., Pisinger D, Kellerer H. 2004, "Knapsack problems", *Springer-Verlag*.

# Using Benders' cuts to backtrack in sports timetabling

David Van Bulck (Speaker) \*

Dries Goossens †

---

## 1 Introduction

This abstract considers the construction of a double round-robin (2RR) sports timetable defining when and where teams meet. In a 2RR timetable, each team faces every other team twice: once as the home team and once as the away team (the game is played in the venue of the home team). The simplicity of this tournament structure notwithstanding, the planning of a practical 2RR timetable is challenging due to the large number and the diversity of constraints and objectives that typically need to be considered. A common approach is therefore to decompose the problem with the first-break-then-schedule (FBTS) approach which first decides when teams play home or away (the so-called home-away pattern or HAP set), after which it determines a compatible assignment of opponents. Despite the common use of FBTS to schedule real-life tournaments in practice (see e.g. [1]), research on how to backtrack between the different phases of the algorithms is scarce. This abstract shows how FBTS relates to the classic Benders' decomposition with integer subproblems. In particular, we show how Benders' feasibility and optimality cuts in combination with the well-known no-good and integer optimality cuts can be used to organize backtracking, resulting in an exact approach. Compared to the classic FBTS approach, our approach has the advantage to be able to cope with most real-life constraints and can even deal with situations where the objective is (partly) determined by the assignment of opponents.

## 2 Problem description and Benders' decomposition

The input of a typical sports timetabling problem consists of a set of  $n$  teams  $T$  that need to play a 2RR, and a set of rounds  $R$  (e.g. days or weekends) during which the games take place. A feasible timetable assigns each game of the tournament to a round in  $R$ , such that no team plays more than one game per round. In the remainder of this abstract, we assume that the number of teams is even and

---

\*[david.vanbulck@ugent.be](mailto:david.vanbulck@ugent.be). Faculty of Economics and Business Administration, Ghent University, Ghent, Belgium.

†[dries.goossens@ugent.be](mailto:dries.goossens@ugent.be). Faculty of Economics and Business Administration, Ghent University, Ghent, Belgium. [FlandersMake@UGent](mailto:FlandersMake@UGent) – core lab CVAMO, Ghent University, Ghent, Belgium.

that there are just enough rounds to schedule all games (i.e.  $|R| = 2(n - 1)$ , so-called compact timetables). We further assume that the objective is to minimize  $z = z_1 + z_2$ , where  $z_1$  only depends on the HAP set associated with the timetable (e.g. the minimization of consecutive home or consecutive away games) and where  $z_2$  (linearly) depends on the assignment of opponents (e.g. the minimization of costs related to the assignment of games). With only very few exceptions, the vast majority of sports timetabling problems previously presented in the literature adhere to this assumption.

One possible approach to tackle this problem is to solve the following monolithic integer programming (IP) formulation, where  $h_{i,r}$  indicates whether team  $i \in T$  plays home in round  $R$  and  $x_{i,j,r}$  indicates whether  $i$  plays at home against  $j \in T \setminus \{i\}$  in  $r$ .

$$\mathbf{min} \quad z_1 + z_2 \tag{1}$$

$$z_1 = \mathbf{a}^\top \mathbf{h} \tag{2}$$

$$z_2 = \mathbf{b}^\top \mathbf{x} \tag{3}$$

$$\sum_{r \in R} h_{i,r} = (n - 1) \quad \forall i \in T \tag{4}$$

$$\sum_{i \in T} h_{i,r} = n/2 \quad \forall r \in R \tag{5}$$

$$\sum_{j \in T \setminus \{i\}} x_{i,j,r} = h_{i,r} \quad \forall i \in T, \forall r \in R \tag{6}$$

$$\sum_{j \in T \setminus \{i\}} x_{j,i,r} = 1 - h_{i,r} \quad \forall i \in T, \forall r \in R \tag{7}$$

$$\sum_{r \in R} x_{i,j,r} = 1 \quad \forall i, j \in T : i \neq j \tag{8}$$

$$h_{i,r} \in \{0, 1\}, x_{i,j,r} \in \{0, 1\} \quad \forall i, j \in T : i \neq j, \forall r \in R \tag{9}$$

The objective is to minimize the sum of  $z_1$  and  $z_2$ . The relation between  $z_1$  and the HAP set is expressed by Constraints (2), where  $\mathbf{a}$  is an application dependent cost vector and  $\mathbf{h}$  is the vector of all  $h_{i,r}$  variables. Constraints (3) regulate the value of  $z_2$ , where  $\mathbf{b}$  is again an application dependent cost vector and  $\mathbf{x}$  is the vector of all  $x_{i,j,r}$  variables. Constraints (4) state that each team plays half of its games at home, and Constraints (5) state that exactly half of the teams in each time slot play home (clearly a necessary condition for HAP set feasibility). Constraints (6) and (7) require that all teams play according to their HAP, and Constraints (8) ensure that all games are scheduled. Finally, constraints (9) are the binary constraints.

Instead of solving the above formulation at once, we propose to apply Benders' decomposition by projecting out the complicating  $x_{i,j,r}$  variables. The master problem then corresponds to constructing a HAP set, whereas the sub problem is to find a compatible assignment of opponents or to prove that none exists. As the  $x_{i,j,r}$  variables need to be integral, we complement the traditional Benders' feasibility and optimality cuts with no-good and integer optimality cuts.

$n$	Literature		HAP					
	LB	Best	LB	Best	Time	BF	NG	Sol.
4	<b>17</b>	<b>17</b>	<b>17</b>	<b>17</b>	$\leq 1$	1	4	1
6	<b>43</b>	<b>43</b>	<b>43</b>	<b>43</b>	$\leq 1$	36	3	6
8	<b>80</b>	<b>80</b>	<b>80</b>	<b>80</b>	$\leq 1$	4	0	10
10	<b>124</b>	<b>124</b>	<b>124</b>	<b>124</b>	1373	39	0	20
12	<b>181</b>	<b>181</b>	<b>181</b>	<b>181</b>	4162	156	0	20
14	<b>252</b>	<b>252</b>	<b>252</b>	<b>252</b>	35	4	0	25
16	<b>327</b>	<b>327</b>	327	329	36000	40	0	37
18	414	417	414	418	36000	185	0	25
20	<b>520</b>	<b>520</b>	<b>520</b>	<b>520</b>	25938	229	0	27
22	<b>626</b>	<b>626</b>	626	629	36000	5	0	44
24	744	749	744	749	36000	17	0	41
26	884	896	884	886*	36000	35	0	34
28	<b>1021</b>	<b>1021</b>	1021	1025	36000	17	0	51
30	1170	1179	1170	1181	36000	12	2	59

Table 1: Computational results for the C-TTP. Proven optimal solutions are indicated in bold, and new better results are indicated with an asterisk.

### 3 Minimizing the sum of travel trips

As will be shown in this talk, the Benders’ decomposition approach outlined in the previous section is widely applicable. To show its effectiveness in this abstract, we only discuss the results for the constant-distance traveling tournament problem (C-TTP). The C-TTP is to construct a compact 2RR timetable that minimizes the total travel trips of the teams such that (i) no team plays more than 3 consecutive away games, (ii) no team plays more than 3 consecutive home games, and (iii) no game  $i$  at  $j$  is immediately followed by game  $j$  at  $i$ . Team  $i$  is said to have a travel trip whenever it travels from the venue of team  $j \in T$  to the venue of team  $k \in T \setminus \{j\}$ .

Over the past two decades, several research efforts have been made to construct high-quality solutions for the C-TTP (see e.g. [2]). Urrutia and Ribeiro [3] show that the total number of travel trips can be minimized by maximizing the total number of breaks (i.e. consecutive home or successive away games). In order to generate trip-minimal timetables, we apply the proposed Benders’ approach by first constructing a break-maximal HAP set and then finding a compatible assignment of opponents.

Computational results for the C-TTP are given in Table 1. The first three columns of the table give the number of teams, the best known lower bound, and the best known solution from the literature. Next, the table provides the best lower bound and solution found, the total time taken, the total number of Benders’ feasibility (‘BF’) and No-Good cuts (‘NG’), and the total number of solutions (i.e. feasible HAPs) found. The low number of no-good cuts illustrates the strength of the Benders’ feasibility cuts: for most infeasible HAP sets, infeasibility is already detected at the level of the LP-relaxation. The table also shows that we are able to

compute an optimal solution for all but three of the problem instances that have previously been solved to optimality in the literature. Together with the fact that we find a new better solution for the 26 teams problem instance, this is remarkable given that the TTP is a well researched problem.

## References

- [1] G. L. NEMHAUSER AND M. A. TRICK, *Scheduling a major college basketball conference*, Oper. Res., 46 (1998), pp. 1–8.
- [2] R. V. RASMUSSEN AND M. A. TRICK, *A Benders approach for the constrained minimum break problem*, Eur. J. Oper. Res., 177 (2007), pp. 198 – 213.
- [3] S. URRUTIA AND C. C. RIBEIRO, *Maximizing breaks and bounding solutions to the mirrored traveling tournament problem*, Discrete Appl. Math., 154 (2006), pp. 1932–1938.

# Minimizing Tardiness in a Scheduling Environment with Jobs' Dominance Hierarchy

Michal Sinai (Speaker) \*

Tami Tamir †

---

## 1 Introduction

In many scheduling environments, the jobs are not treated in a fair way. Naturally, some jobs have higher priority than others. Such scenarios are theoretically modelled in two ways: (i) jobs are associated with weights that reflect their importance. (ii) there are precedence constraints among jobs. We study a scheduling setting, motivated by real-life behaviour, in which the priority among jobs is defined in a different way. In our model, the users are arranged in a *dominance hierarchy*, and high ranking users are ready to accept only outcomes in which the service they receive cannot be improved by hurting the service provided to subordinate users.

Studies in behavioral science show that different organisms have different aggressiveness levels. Aggression is defined as a behavior intended to increase the social dominance of the organism relative to the dominance position of other organisms. Different levels of aggressiveness lead to a dominance hierarchy. Highly rank members of the society have better access to valuable resources such as mates, living space, and food.

Our model is inspired by such environments. Specifically, the jobs are partitioned into  $c$  hierarchical levels, and the only allowed schedules are those that are stable against an aggressive behaviour (bypassing of subordinate jobs) of high-ranking jobs. Thus, while the assignment is determined by a centralized authority, it must satisfy some stability constraints implied by the jobs' dominance hierarchy.

Let  $\mathcal{J}$  be a set of jobs. Every job  $J_j \in \mathcal{J}$  has a processing time  $p_j$ , as well as a due-date  $d_j$ , denoting the time in which it should be completed. The set  $\mathcal{J}$  consists of  $c$  sets;  $\mathcal{J} = \mathcal{J}_1 \cup \dots \cup \mathcal{J}_c$ , where  $\mathcal{J}_\ell$  is a set of jobs in the  $\ell$ -th hierarchy level. That is, the jobs of  $\mathcal{J}_1$  have the highest rank, and the jobs of  $\mathcal{J}_c$  are the most subordinate. A schedule  $\pi$  on a single machine determines a non-preemptive assignment of the jobs on the machine. For a schedule  $\pi$  and a job  $J_j$ , let  $S_j(\pi), C_j(\pi)$  denote the start time and the completion time of  $J_j$  in  $\pi$ . In our setting, all the jobs are available at time 0 (no release times), and no preemptions are allowed, therefore, for every job  $j$ ,  $C_j(\pi) - S_j(\pi) = p_j$ . Also, w.l.o.g., we only

---

\*michal.sinai@post.idc.ac.il. School of Computer Science, Reichman University, Israel.

†tami@idc.ac.il. School of Computer Science, Reichman University, Israel.

consider schedules with no intended idle. Clearly, idle segments can be removed by shifting some jobs to start earlier. By the above, a schedule  $\pi$  can be described by specifying an order of the jobs, and  $C_j(\pi) = \sum_{j': S_{j'}(\pi) \leq S_j(\pi)} p_{j'}$ .

For a given schedule  $\pi$ , let  $L_j(\pi) = C_j(\pi) - d_j$  denote the lateness of job  $J_j$  in  $\pi$ .  $T_j(\pi) = \max\{0, C_j(\pi) - d_j\}$  is the *tardiness* of  $J_j$ , and  $U_j(\pi) \in \{0, 1\}$  is a binary *lateness indicator* indicating whether  $J_j$  is *tardy*, that is,  $U_j(\pi) = 1$  if and only if  $C_j(\pi) > d_j$ .

A schedule  $\pi$  is considered *feasible* with respect to some objective function, if for every hierarchy level  $1 \leq \ell \leq c$ , and every job  $J_i \in \mathcal{J}_\ell$  it holds that the objective value of  $J_i$  cannot be improved by preserving or improving the objective value of each of the jobs in  $\cup_{1 \leq k \leq \ell} \mathcal{J}_k$ , and possibly hurting the objective value of less dominant jobs (in  $\cup_{\ell+1 \leq k \leq c} \mathcal{J}_k$ ). This general definition has a different practical meaning depending on the objective function, for example, a schedule  $\pi$  is *feasible with respect to tardiness* if for every tardy job  $J_i$  it holds that there is no schedule in which  $J_i$  has a reduced tardiness and no job from a higher or equal hierarchy level has an increased tardiness.

We analyze two objective functions. The first is minimizing the maximal tardiness, and the second is minimizing the number of tardy jobs. Both problems have simple greedy algorithms in an environment without a dominance hierarchy [4, 3]. Using the common three-fields notation for scheduling problems [1], we denote the corresponding problems in the presence of a dominance hierarchy by  $1|{\textit{hierarchy}}|T_{max}$  and  $1|{\textit{hierarchy}}|\sum U_j$ .

For the objective of minimizing the total completion time, in every feasible schedule, for all  $\ell_1 < \ell_2$ , all the jobs of  $\mathcal{J}_{\ell_1}$  will be processed before all the jobs of  $\mathcal{J}_{\ell_2}$ . Thus, an optimal solution is easily obtained by processing the jobs of each hierarchy level in shortest processing time order. Objective functions that depend on jobs' tardiness are more challenging since a job may have a high completion time and still perform perfectly as long as it is not tardy. Thus, the order of jobs in an optimal schedule does not necessarily agree with their ranks. This observation is crucial in understanding the model and the involved challenges.

The general problem we consider is finding a feasible schedule that optimizes the objective function, that is, minimizes the maximal tardiness of a job, or minimizes the number of tardy jobs. Another problem that we consider is a multi-criteria one. Specifically, the primary goal is to optimize the schedule for  $\mathcal{J}_1$ . Out of all feasible schedules achieving the best for  $\mathcal{J}_1$ , the goal is to optimize the schedule for the jobs in  $\mathcal{J}_2$ , and so on. We use the notation  $1|{\textit{hierarchy}}|(\gamma_1, \dots, \gamma_c)$  to denote the problem with the multi-criteria objective function  $\gamma$ . E.g., for  $c = 2$ , in the problem  $1|{\textit{hierarchy}}|(U_1, U_1)$ , the primary goal is to minimize the number of tardy dominant jobs, and among all the feasible schedules achieving this objective, minimize the number of tardy subordinate jobs.

The following example illustrates some of the challenges in scheduling jobs with different hierarchy levels, and the difference between the global objective function and the multi-criteria one. Consider the problem of minimizing the number of tardy jobs. That is,  $1|\sum U_j$ . Assume  $c = 2$ . Let  $\mathcal{A} = \{a_1, a_2, a_3\}$  be the set of

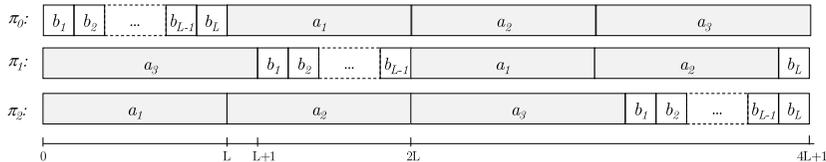


Figure 1:  $\pi_0$  is optimal for  $1||\sum U_j$  (no hierarchy),  $\pi_1$  is optimal for  $1|hierarchy|\sum U_j$ , and  $\pi_2$  is optimal for  $1|hierarchy|(U_A, U_B)$ .

dominant jobs, where  $p_1 = p_2 = L$  and  $p_3 = L+1$ , for some constant  $L > 2$ . The set  $\mathcal{B}$  of subordinate jobs includes  $L$  unit-length jobs. Note that  $n = |\mathcal{A}| + |\mathcal{B}| = L + 3$ . Assume further that all the jobs in the instance have the same due-date of  $2L$ . Without dominance hierarchy, an optimal schedule for  $\sum U_j$  is the schedule  $\pi_0$ , presented in the top of Fig. 1. There are  $L + 1$  non-tardy jobs. With hierarchy, an optimal schedule for  $\sum U_j$  is the schedule  $\pi_1$ . There are  $L$  non-tardy jobs: the longer dominant job,  $a_3$ , and  $L - 1$  subordinate jobs. The schedule  $\pi_1$  is feasible, even though  $a_1$  and  $a_2$  are late. None of these jobs can benefit from bypassing subordinate jobs, as their total processing time is less than  $L$ . The schedule  $\pi_2$  is optimal for the problem  $1|hierarchy|(U_A, U_B)$ . Only the two dominant jobs  $a_1$  and  $a_2$  are non-tardy, and all the other  $L + 1$  jobs are late.

## 2 Our Results

We present the following results for scheduling on a single machine with the presence of jobs' dominance hierarchy:

**1.** algorithms for testing the feasibility of a schedule with respect to jobs' tardiness and with respect to the lateness indicator. The two tests are different since the feasibility constraint for the later problem is weaker. Consider for example the schedule  $\pi_1$  in Fig. 1. Job  $a_1$  may reduce its tardiness from  $L$  to 1 if it bypass the jobs of  $\mathcal{B}$ . It will remain tardy in the resulting schedule, therefore,  $a_1$  cannot reduce its lateness indicator. Thus,  $\pi_1$  is feasible with respect to  $U_j$ , but not feasible with respect to  $T_j$ .

**2.** for the problem of minimizing the maximal tardiness of a job, we present an algorithms that produces an optimal schedule for both the global objective and the multi-criteria objective. The algorithm is based on an initial schedule of the jobs according to EDD order, followed by shifting and reassigning of jobs. The challenge is to make the minimal possible perturbation of the EDD order that will guarantee feasibility.

**3.** for the problem of minimizing the number of tardy jobs, we distinguish between the global objective and the multi-criteria objective. For a constant number of hierarchy levels we present an NP-hardness proof for the global objective, and an optimal algorithm for the multi-criteria objective. The algorithm is based on a dynamic programming solution similar to the one presented in [2] for  $1||\sum w_j U_j$  with a constant number of different job weights.

## References

- [1] 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. *Ann. Discrete Math*, 5:287–326, 1979.
- [2] D. Hermelin, S. Karhi, M. Pinedo, and D. Shabtay. New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research, Springer*, 298(1):271–287, 2012.
- [3] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Sci.*, 19:544–546, 1973.
- [4] J.M. Moore. An  $n$  job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Sci.*, 15:102–109, 1968.

# Replacement and Repair of Common Components in Systems Subject to Operations Planning

Gabrijela Obradović (Speaker) \*      Ann-Brith Strömberg †

Kristian Lundberg ‡

---

## 1 Motivation

While a system operates its components deteriorate and in order for the system to stay operational, the components have to be maintained regularly in relation to their usage in the system. When planning the maintenance for the system, the decisions to be made concern when each of its components should be maintained (i.e., repaired or serviced) and what kind of maintenance should then be performed, with respect to the operational schedule of the system. So-called preventive maintenance (PM) can often be planned well in advance, while corrective maintenance (CM) is done after a failure has occurred, which may come on very short notice. On the other hand, an unexpected but necessary CM action may provide an opportunity for PM actions to be rescheduled, starting from the system's current state. While both PM and CM are aimed at restoring the components in order to put the system back in an operational state, CM is often much more costly than PM, due to a longer system down-time and also due to possible damages to other components caused by the failure. In this research, we consider PM scheduling, while CM is implicitly included by an additional cost which increases with the time between PM occasions. The increasing cost reflects the increased risk of having to perform CM.

We consider a setting with one *system operator* and one *maintenance workshop*, which are typically two separate stakeholders, and a *contract* governing their joint activities. Components that are to be maintained are sent to a maintenance workshop, which needs to schedule and perform all maintenance activities while satisfying the contract, which may define conditions on delivery dates for and/or requirements on the availability of components for the system operator.

---

\*[gabobr@chalmers.se](mailto:gabobr@chalmers.se). Department of Mathematical Sciences, Chalmers University of Technology, Gothenburg, Sweden.

†[anstr@chalmers.se](mailto:anstr@chalmers.se). Department of Mathematical Sciences, Chalmers University of Technology, Gothenburg, Sweden.

‡[kristian.lundberg@saabgroup.com](mailto:kristian.lundberg@saabgroup.com). Saab AB, Linköping, Sweden.

The workshop’s ability to fulfill the contract is dependent on its capacity, in terms of the number of parallel repair lines; the investment costs for additional repair lines should thus be weighed against the cost of not being able to fulfill the contract at hand.

The original goal of this research was to investigate how different contracting forms between the stakeholders affect the efficiency of maintenance activities, the flow of components through the system—of—systems, as well as the availability of the systems over time. The first contract we modeled resembles the type of contract currently employed by our case industry, i.e., a component repair turn-around time based contract. Since the resulting mathematical model [1] appeared

to be computationally intractable we chose to challenge and compare this contract with a contract aimed at regulating the availability of repaired components. The corresponding mathematical model appeared to be substantially more tractable, and also the solutions—in terms of resulting numbers of repaired components on the stock—seem to be more robust in terms of ability to keep the systems running.

## 2 Problem description

A number of systems are operating to fulfill a common production demand; their operating schedules are assumed to be predefined, resulting in certain time-windows during which maintenance of the systems’ components may be performed. While the systems operate their components degrade, which lead to a requirement for maintenance (i.e., service, replacement, or repair of the components of the systems). At a maintenance occasion, one or several components are taken out of the system, sent to the maintenance workshop for repair, and returned back to the stock of repaired components, ready to be used again (by any of the systems). The components that are sent for repair are instantly replaced by components that are currently on the the stock of repaired components. Hence, there is a circulating flow of individual components, being used and degraded, replaced, repaired or serviced, and then put back in a system to be used again. This structure of the system—of—systems is illustrated in Figure 1.

We make a formal definition of the generalized PMSPIC—which models the replacement scheduling for the components of the systems considered—along with a mixed-binary linear optimization (MBLP) formulation. Then, the scheduling

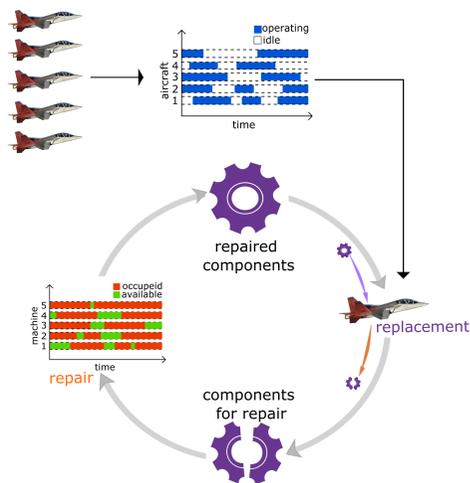


Figure 1: Illustration of the problem for an application with a system of aircraft.

of the maintenance workshop is modeled using mixed-integer linear optimization (MILP). These systems are then integrated through the dynamics of the stocks of components waiting to be maintained and those that have finished maintenance and are available to be used again by the systems. We analyze an availability contracting form between the system operator and the repair workshop by studying and comparing the Pareto fronts resulting from different parameter settings, regarding minimum allowed stock levels as well as investments in the repair capacity of the workshop.

### 3 Summary of Results

Figure 2 shows the computed points on the Pareto front in the bi-objective optimization problem for the workshop capacities  $L = 10$  and  $L = 3$ . The lower limit on the availability is in the interval  $[5, 10]$  while the total maintenance cost is in the interval  $[5542, 5828]$  for  $L = 10$  and in the interval  $[5631, 5856]$  for  $L = 3$ . We observe that for every increase by one in the availability, the increase in the maintenance cost becomes higher. That leads to longer maintenance interval lengths which increases the risk of component/system failure. To receive a high lower limit on components available, there has to be a loss on the system operator's side, which could be, for example, that maintenance intervals are longer which leads to higher maintenance costs. Another observation is that the difference between maintenance costs for  $L = 3$  and  $L = 10$  decreases as the availability of repaired components increases; this means that it is costly to obtain a higher availability, regardless of the capacity in the maintenance workshop.

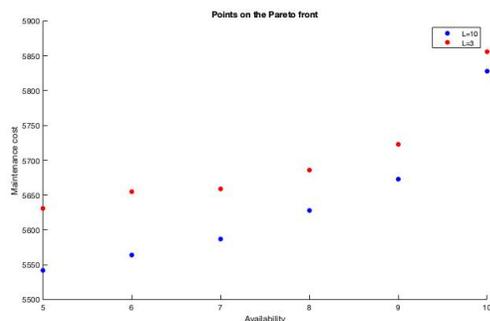


Figure 2: Availability vs. Maintenance cost. The computed points on the Pareto front for  $(I, J_i, K, T, \underline{b}^i) = (5, 15, 10, 40, 1)$ ,  $\epsilon = 1$ ,  $L \in \{3, 10\}$ .

Figure 3 shows the load of the maintenance workshop over time, for capacities  $L \in \{3, 5, 10\}$ . We observe that for  $L = 10$ , the number of active repair lines does not exceed 7, which implies that  $L \geq 7$  does not constrain the number of repair lines used at any time in an optimal solution. However, when reducing the capacity to  $L = 5$ , there are many time steps at which the workshop is working at full capacity, and that is even more expressed when  $L$  is reduced to 3. If some unexpected failures occur, or if some components have longer processing times, a planned utilisation of the full capacity of the workshop at multiple consecutive time steps might lead to later/postponed deliveries. A consequence of later deliveries is lower levels on the stock of repaired components, which may not satisfy the

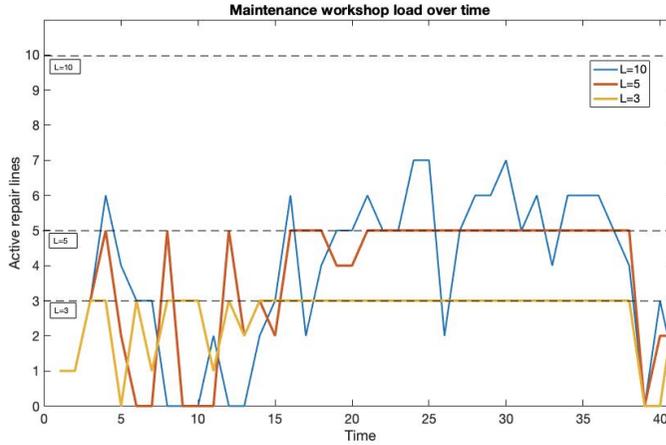


Figure 3: Load of the maintenance workshop over time for  $(I, J_i, K, T, \underline{b}^i) = (5, 15, 10, 40, 1)$ . For  $L = 10/5/3$ , the point on the Pareto front is given by: Availability=5 and Maintenance cost=5542/5546/5631.

lower limit on availability. This may lead to maintenance intervals having to be extended. Therefore, the loading of the parallel repair machines should not be at the level of its upper limit for too many time steps.

## 4 Conclusions

We present a brief overview of an integrated model of a system-of-systems composed by the maintenance scheduling for components, the maintenance workshop, the stock dynamics, and an availability contract governing joint activities of the two respective stakeholders. The solutions resulting from our modelling can be used to find a lower limit on an optimal joint performance of a collaboration between stakeholders governing a common system-of-systems regulated by an availability contract.

## References

- [1] G. OBRADOVIĆ (2021). *Mathematical Modeling, Optimization and Scheduling of Aircraft's Components Maintenance and of the Maintenance Workshop*. Chalmers University of Technology, Sweden. <https://research.chalmers.se/en/publication/524023>

# Parameterized Complexity of Single-machine Scheduling with Precedence, Release Dates and Deadlines

Claire Hanen \*    Maher Mallem (Speaker) †    Alix Munier Kordon ‡

---

## 1 Introduction

Many scheduling problems, even seemingly basic ones such as  $1|r_i|L_{\max}$ , have been proved strongly NP-hard [7] over the years. Upon reaching NP-hardness this quickly, it becomes difficult to establish anything deeper about these problems within the scope of classical complexity theory.

To answer this, parameterized complexity theory gives additional tools for a refined analysis of such hard scheduling problems. Given a parameter  $k$  and denoting  $n$  the input size, a problem is called *fixed-parameter tractable* (FPT) with respect to parameter  $k$  if it can be solved in time  $\text{poly}(n) \times f(k)$  with  $f$  an arbitrary function [3]. The idea is to identify  $k$  as the limiting property and give a polynomial time algorithm for all instances with a bounded value of  $k$ . When the studied problem is believed to not be FPT, the para-NP class is used as a parameterized version of NP: a problem is in para-NP with respect to parameter  $k$  if there is a non-deterministic algorithm that solves it in time  $\text{poly}(n) \times f(k)$  with  $f$  an arbitrary function. In order to prove that a problem is para-NP-hard with respect to  $k$ , it is enough to prove that the un-parameterized problem is NP-hard for some fixed value of the parameter [4]. For example the  $k$ -COLORING graph problem is para-NP-hard with respect to the number  $k$  of colors from knowing that the graph coloring problem is already NP-hard with  $k = 3$  colors.

Several parameters have been considered for the analysis of the parameterized complexity of scheduling problems, like the maximum processing time  $p_{\max}$  [8] or the width of the precedence graph [2]. In the presence of time windows, recent papers considered parameters based on the structure of time intervals. Bodlaender and van der Wegen [1] addressed the single-machine scheduling of a set of chain like precedence constraints with delays between successive tasks, and each chain has its own time window (release time and deadline). They defined the *thickness* as the maximum number of overlapping intervals of chains and proved that the problem with minimum delays given in unary is W[2]-hard with respect to the

---

\*[claire.hanen@lip6.fr](mailto:claire.hanen@lip6.fr). Sorbonne Université, CNRS, LIP6, F-75005 Paris, France and UPL, Université Paris Nanterre, F-92000 Nanterre, France

†[maher.mallem@lip6.fr](mailto:maher.mallem@lip6.fr). Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

‡[alix.munier@lip6.fr](mailto:alix.munier@lip6.fr). Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

thickness. Munier Kordon [9] developed an FPT algorithm for the decision problem  $P|prec, p_j = 1, r_j, d_j|*$ . A star in the third field denotes here a decision problem. The parameter was the *pathwidth*  $\mu$  of the interval graph induced by the time windows of the tasks, which equals the maximum number of overlapping time windows at any given time, minus one. An FPT algorithm with respect to the tuple of parameters  $(\mu, p_{\max})$  has been provided in [5] for a problem with typed tasks of arbitrary processing times. This shows that the pathwidth  $\mu$  is an interesting parameter to handle problems with time windows. In this work we consider several single-machine scheduling problems with release dates, deadlines and precedence relations with or without delays. We analyze the parameterized complexity of these problems with respect to the parameter  $\mu$ .

## 2 An FPT algorithm for $1|prec, r_j, d_j|*$

In this section we consider the case where there are no delays. Related problems are  $1|prec, r_i|C_{\max}$  which is polynomial [6] whereas  $1|r_i|L_{\max}$  is strongly NP-hard [7]. This implies the NP-hardness of  $1|prec, r_j, d_j|*$ , even without precedence constraints. It is also proved in [5] that  $P2|r_j, d_j|*$  is para-NP-complete with respect to the parameter  $\mu$ . We provide in this talk an FPT algorithm based only on the parameter  $\mu$  for the single-machine case. We consider the sorted list  $(u_\alpha)$  of end points of the jobs intervals  $(r_i, d_i)$ . The algorithm is based on a dynamic programming recurrence equation for  $M_\alpha(X)$ , the optimal makespan of a feasible schedule of a subset  $X$  of jobs starting before  $u_\alpha$ . The parameter  $\mu$  is used to enumerate the feasible subsets and the terms of the recurrence equation, leading to the following result:

**Theorem 1**  $1|prec, r_j, d_j|*$  and  $1|prec, r_j, d_j|C_{\max}$  are fixed-parameter tractable with respect to the parameter  $\mu$ . The complexity of the FPT algorithm is  $\mathcal{O}(n^2 \times 2^{2\mu})$ .

## 3 Hardness results for $1|chains(\ell_{i,j}), p_j = 1, r_j, d_j|*$

In this section, delays values  $\ell_{i,j}$  are added to the precedence relations. Two cases are considered: EXACT-DELAYS for which delays are equal to the given values  $\ell_{i,j}$  and MIN-DELAYS for which delays are required to be greater than or equal to the given values. In both cases we show that adding delays to the problem makes it para-NP complete with respect to parameter  $\mu$  even with chains as precedence relations and unit-time jobs.

**Theorem 2**  $1|chains(\ell_{i,j}), p_j = 1, r_j, d_j|*$  is strongly NP-hard with pathwidth  $\mu = 1$  in both EXACT-DELAYS and MIN-DELAYS cases.

This result complements the work done by Bodlaender and van der Wegen [1], which investigated the parameterized complexity of similar single-machine scheduling problems where time windows were given on chains instead of individual jobs

with respect to the thickness. We establish the para-NP hardness of our problems, a stronger result than W[2]-hardness. Moreover, our reductions are based on a trick of our own that do not require delays to be given in unary.

The EXACT-DELAYS case is proven using a reduction from 3-COLORING. We consider as many chains as there are nodes in the input graph. We dedicate one time segment of length three to each edge in the graph. The chain relative to a node  $i \in [1..n]$  must schedule a job in all the time segments where node  $i$  is part of the corresponding edge. Then two nodes sharing an edge have both a job to schedule in the same edge time segment of length three at some point. If we associate the position of jobs to color choices, then scheduling both jobs at different times corresponds to choosing different colors between nodes sharing an edge. Finally the exact delays are used to make the color choice consistent along each chain.

The MIN-DELAYS case is also reduced from 3-COLORING and is based on the same idea as the previous reduction. Around the beginning and the end of each chain, we add two gadgets that force all the delays along the edge time segments to be equal to their minimum value in any valid schedule. This allows us to keep the color choice consistent along each chain and make the reduction work the same way as in the EXACT-DELAYS case.

## 4 Conclusion

In this talk we provide a new FPT algorithm for a single-machine scheduling problem with release dates, deadlines and precedence relations with pathwidth  $\mu$  as the parameter. When precedence delays are added, we show the para-NP-completeness with respect to parameter  $\mu$  of a restrictive sub-case of the problem, which suggests that parameter  $\mu$  alone is not enough to handle delays efficiently. Future work would investigate the maximum value of a delay  $\ell_{\max}$  as a parameter. It would be interesting to see whether an FPT algorithm exists for problem  $1|prec(\ell_{i,j}), p_j = 1, r_j, d_j|C_{\max}$  with respect to the tuple of parameters  $(\mu, \ell_{\max})$ .

## References

- [1] Hans L Bodlaender and Marieke van der Wegen. Parameterized complexity of scheduling chains of jobs with delays. *arXiv preprint arXiv:2007.09023*, 2020.
- [2] Robert Brederick, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, René van Bevern, and Gerhard J Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. In *International conference on discrete optimization and operations research*, pages 105–120. Springer, 2016.
- [3] R. Downey and M. Fellows. *Parameterized complexity*. Springer, 1999.
- [4] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer, 1998.

- [5] Claire Hanen and Alix Munier-Kordon. Fixed-parameter tractability of scheduling dependent typed tasks subject to release times and deadlines. *Submitted*, 2021.
- [6] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Sci.*, 19:544–546, 1973.
- [7] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Ann. of Discrete Math.*, 1:343–362, 1977.
- [8] Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1):533–562, 2015.
- [9] Alix Munier Kordon. A fixed-parameter algorithm for scheduling unit dependent tasks on parallel machines with time windows. *Discrete Applied Mathematics*, 290:1–6, 2021.

# Joint replenishment meets scheduling

Tamás Kis (Speaker) \*      Péter Györgyi †      Tímea Tamási ‡  
József Békési §

---

## 1 Introduction

We consider a combination of two classic problems of operations research: the joint replenishment problem (JRP) and single machine scheduling with release dates. In this problem, each job requires some resources and it can be processed on the machine after the required resources are replenished. One has to decide both about the replenishments and the schedule of the machine. The first subproblem is a well-known variant of the joint replenishment problem, while the second one belongs to the class of single machine scheduling problems with release dates, where the release dates are determined by the replenishment times of the required resources.

In the joint replenishment problem, the goal is to fulfill a set of demands (jobs) emerging over the time horizon. A demand can be fulfilled by ordering its required items (resources) not sooner than the arrival time of the demand. Orders of different demands can be combined, and the cost of simultaneously ordering a subset of item types incurs a joint ordering cost and an additional item ordering cost for each item type in the order. None of these costs depends on the number of units ordered. One of the main variants of this problem is the so-called *JRP-W*, where the objective is to minimize the sum of the ordering costs and the cost incurred by delaying the fulfillment of the jobs. Our problem is an extension of this variant: each demand (job) has to be processed on a single machine after the required items (resources) are replenished. The objective is to minimize the total ordering cost plus a scheduling criterion. We provide several complexity results for the offline problem, and competitive analysis for online variants with min-sum and min-max criteria, respectively.

Formally, we have a set  $\mathcal{J}$  of  $n$  jobs that have to be scheduled on a single machine. Each job  $j$  has a processing time  $p_j > 0$ , a release date  $r_j \geq 0$ , and

---

\*[kis.tamas@sztaki.hu](mailto:kis.tamas@sztaki.hu). Institute for Computer Science and Control, Eötvös Loránd Research Network, Kende Str. 13-17., 1111 Budapest, Hungary.

†[gyorgyi.peter@sztaki.hu](mailto:gyorgyi.peter@sztaki.hu). Institute for Computer Science and Control, Eötvös Loránd Research Network.

‡[tamasi.timea@sztaki.hu](mailto:tamasi.timea@sztaki.hu). Institute for Computer Science and Control, Eötvös Loránd Research Network and Department of Operations Research, Eötvös Loránd University.

§[bekesi@inf.u-szeged.hu](mailto:bekesi@inf.u-szeged.hu). Department of Computer Algorithms and Artificial Intelligence, Faculty of Science and Informatics, University of Szeged, Hungary.

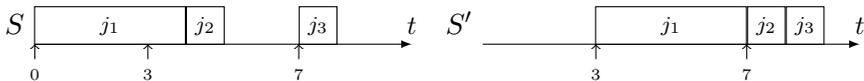


Figure 1: Two feasible solutions. The arrows below the time axis indicate the replenishment time points.

possibly a weight  $w_j > 0$  (in case of min-sum type objective functions). In addition, there is a set of resources  $\mathcal{R} = \{R_1, \dots, R_s\}$ , and each job  $j \in \mathcal{J}$  requires a non-empty subset  $R(j)$  of  $\mathcal{R}$ . A job  $j$  can only be started if all the resources in  $R(j)$  are replenished after  $r_j$ . Each time some resource  $R_i$  is replenished, a fixed cost  $K_i$  is incurred on top of a fixed cost  $K_0$ , which must be paid each time any replenishment occurs. These costs are independent of the replenished amount.

A solution of the problem is a pair  $(S, \mathcal{Q})$ , where  $S$  is a schedule specifying a starting time for each job  $j \in \mathcal{J}$ , and  $\mathcal{Q} = \{(\mathcal{R}_1, t_1), \dots, (\mathcal{R}_q, t_q)\}$  is a replenishment structure, which specifies time moments  $t_\ell$  along with subsets of resources  $\mathcal{R}_\ell \subseteq \mathcal{R}$  such that  $t_1 < \dots < t_q$ . We say that job  $j$  is *ready to be started at time moment  $t$  with respect to replenishment structure  $\mathcal{Q}$* , if each resource  $R \in R(j)$  is replenished at some time moment in  $[r_j, t]$ , i.e.,  $R(j) \subseteq \cup_{t_\ell \in [r_j, t]} \mathcal{R}_\ell$ . The solution is *feasible* if (i) the jobs do not overlap in time, i.e.,  $S_j + p_j \leq S_k$  or  $S_k + p_k \leq S_j$  for each  $j \neq k$ , and (ii) each job  $j \in \mathcal{J}$  is ready to be started at  $S_j$  w.r.t.  $\mathcal{Q}$ .

The *cost of a solution* is the sum of the scheduling cost  $c_S$ , and the replenishment cost  $c_{\mathcal{Q}}$ . The former can be any optimization criteria know in scheduling theory (e.g., the total weighted completion time  $\sum w_j C_j$ , the total flow time  $\sum F_j$ , or the maximum flow time  $F_{\max}$ ). The replenishment cost is calculated as follows:  $c_{\mathcal{Q}} := \sum_{\ell=1}^{|\mathcal{Q}|} (K_0 + \sum_{R_i \in \mathcal{R}_\ell} K_i)$ .

**Example 1** Suppose there are 3 jobs,  $p_1 = 4$ ,  $p_2 = p_3 = 1$ ,  $r_1 = 0$ ,  $r_2 = 3$ , and  $r_3 = 7$ , and the objective is to minimize  $\sum C_j$ . The replenishment costs are  $K_0$  and  $K_1$ , and we deliberately do not specify numerical values for these parameters. If there are 3 replenishments from a single resource  $R$ , i.e.,  $\mathcal{Q} = ((\{R\}, 0), (\{R\}, 3), (\{R\}, 7))$ , and the starting times of the jobs are  $S_1 = 0, S_2 = 4$ , and  $S_3 = 7$ , then  $(S, \mathcal{Q})$  is a feasible solution with total replenishment cost of  $3(K_0 + K_1)$  and total completion time 17 (Figure 1 left).

However, if there are only two replenishments in  $\mathcal{Q}'$  at  $t_1 = 3$  and  $t_2 = 7$ , then we have to start the jobs later, e.g., if  $S'_1 = 3, S'_2 = 7$ , and  $S'_3 = 8$ , then  $(S', \mathcal{Q}')$  is feasible. Observe that in the second solution we have saved the cost of a replenishment ( $K_0 + K_1$ ), however, the total completion time of the jobs has increased from 17 to 24 (see Figure 1 right).

As to whether  $(S, \mathcal{Q})$  is better than  $(S', \mathcal{Q}')$  or not depends on the value of  $K_0 + K_1$ .

Table 1: Results of the paper.

Problem	Result
$1 jrp, s = 1, r_j \sum C_j + c_Q$	NP-hard
$1 jrp, p_j = 1, r_j \sum C_j + c_Q$	NP-hard
$1 jrp, s = 2, r_j F_{\max} + c_Q$	NP-hard
$1 jrp, s = \text{const}, p_j = 1, r_j \sum w_j C_j + c_Q$	polynomial alg.
$1 jrp, s = \text{const}, p_j = p, r_j \sum C_j + c_Q$	polynomial alg.
$1 jrp, s = 1, r_j F_{\max} + c_Q$	polynomial alg.
$1 jrp, s = \text{const}, p_j = p, r_j F_{\max} + c_Q$	polynomial alg.
$1 jrp, s = 1, p_j = 1, r_j \sum C_j + c_Q$	2-competitive alg.
$1 jrp, s = 1, p_j = 1, r_j \sum C_j + c_Q$	no $\left(\frac{3}{2} - \varepsilon\right)$ -competitive alg.
$1 jrp, s = 1, p_j = 1, r_j \sum w_j C_j + c_Q$	no $\left(\frac{\sqrt{5}+1}{2} - \varepsilon\right)$ -competitive alg.
$1 jrp, s = 1, p_j = 1, r_j \sum F_j + c_Q$	2-competitive alg.
$1 jrp, s = 1, p_j = 1, r_j \sum F_j + c_Q$	no $\left(\frac{3}{2} - \varepsilon\right)$ -competitive alg.
$1 jrp, s = 1, p_j = 1, \text{regular } r_j F_{\max} + c_Q$	$\sqrt{2}$ -competitive alg.
$1 jrp, s = 1, p_j = 1, \text{regular } r_j F_{\max} + c_Q$	no $\left(\frac{4}{3} - \varepsilon\right)$ -competitive alg.
$1 jrp, s = 1, p_j = 1, r_j F_{\max} + c_Q$	no $\left(\frac{\sqrt{5}+1}{2} - \varepsilon\right)$ -competitive alg.

\*  $jrp$  indicates the joint replenishment extension,  $s = 1$  limits the number of resources to 1, and  $c_Q$  is the total replenishment cost.

## 2 Results

The main results of the paper fall in 3 categories: (i) NP-hardness proofs, (ii) polynomial time algorithms, and (iii) competitive analysis of online variants of the problem, see Table 1 for an overview. We provide an almost complete complexity classification for the offline problems with both of the  $\sum w_j C_j$  and  $F_{\max}$  objectives. Notice that the former results imply analogous ones for the  $\sum w_j F_j$  criterion. While most of our polynomial time algorithms work only with unit-time jobs, a notable exception is the case with a single resource and the  $F_{\max}$  objective, where the job processing times are arbitrary positive integer numbers. We have devised online algorithms for some special cases of the problem for both min-sum and min-max criteria. In all variants for which we present an online algorithm with constant competitive ratio, we have to assume unit-time jobs. While we have a 2-competitive algorithm with unit time jobs for min-sum criteria, for the online problem with the  $F_{\max}$  objective we also have to assume that the input is regular, i.e., in every time unit a new job arrives, but in this case the competitive ratio is  $\sqrt{2}$ . The technical details can be found in [1].

## Acknowledgment

This work has been supported by the National Research, Development and Innovation Office grants no. TKP2021-NKTA-01, and ED 18-2-2018-0006. The research of Péter Györgyi was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

## References

- [1] P. GYÖRGYI, T. KIS, T. TAMÁSI, J. BÉKÉSI (2021). *Joint replenishment meets scheduling*. <https://arxiv.org/abs/2104.09178>

# The High-Dimensional Cow-Path Problem

Antonios Antoniadis\*    Ruben Hoeksma\*    Sándor Kisfaludi-Bak†  
Kevin Schewior (Speaker)‡

---

## 1 Introduction

The  $d$ -dimensional *Hyperplane Search Problem* (informally cow-path problem) asks for a curve (or *search strategy*)  $\zeta : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^d$  with  $\zeta(0) = 0$  that minimizes the competitive ratio: We say that  $\zeta$  is  $c$ -*competitive* if there exists  $\alpha \in \mathbb{R}_{\geq 0}$  such that

$$\text{length}(\zeta|_{[0,t^*]}) \leq c \cdot \text{dist}(0, H) + \alpha$$

for all  $H \in \mathcal{H}^{d-1}$ , where  $\mathcal{H}^{d-1}$  is the set of all hyperplanes in  $\mathbb{R}^d$  and  $t^* = \inf\{t : \zeta(t) \in H\}$ . The infimum over all such  $c$  is the *competitive ratio* of  $\zeta$ .

When  $d = 1$ , it is well-known that the best-possible competitive ratio is 9 [?], achieved by visiting the points  $(-2)^0, (-2)^1, \dots$  sequentially. Moreover, the competitive ratio can be improved to about 4.591 using randomization [?]. Surprisingly, in addition to the case of  $d = 1$ , results only seem to be known for  $d = 2$  and for offline versions [?, ?], leaving a gap in the online-algorithms literature. For  $d = 2$ , it is conjectured that a logarithmic spiral that achieves a competitive ratio of about 13.811 is optimal [?, ?] (for a related problem, such a spiral is known to be optimal [?]). The present abstract addresses the aforementioned gap by providing the first asymptotic results for  $d \rightarrow \infty$ .

For the remainder of the abstract, we will consider the essentially equivalent but arguably cleaner *Sphere Inspection Problem*: The goal is to find a  $d$ -dimensional minimum-length closed curve,  $\gamma$ , that *inspects the unit sphere* in  $\mathbb{R}^d$ , i.e.,  $\gamma$  sees every point  $p$  on the unit sphere  $S^{d-1}$ . Here, we say that an object  $O \subseteq \mathbb{R}^d$  *sees* a point  $p'$  on the surface of the unit sphere if there is a point  $p \in O$  such that the line segment  $pp'$  intersects the unit *ball* exclusively at  $p'$ . Note that the curve is not required to start in the origin any more. Such a minimum-length curve exists for any dimension [?]. While this problem is trivial for  $d \in \{1, 2\}$ , it has only been

---

\*Department of Applied Mathematics, University of Twente, Enschede, The Netherlands.  
fa.antoniadis,r.p.hoeksma@utwente.nl

†Department of Computer Science, Aalto University, Espoo, Finland.  
sander.kisfaludi-bak@aalto.fi

‡Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark. kevs@sdu.dk

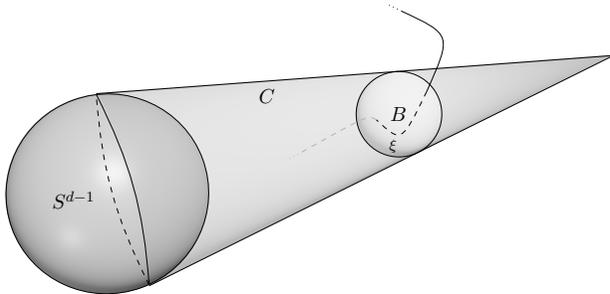


Figure 1: The apex of the cone  $C$  sees all points on the unit sphere that  $B$  does.

shown recently that the best-possible length for  $d = 3$  is  $4\pi$  [?]. No results for higher dimensions are known.

It is not difficult to see that both problems are equivalent with respect to the asymptotic behavior of the achievable competitive ratio and tour length, respectively: A curve that inspects the sphere can be turned into a search strategy by a standard doubling technique. Conversely, a certain prefix of a search strategy can be viewed as half of a closed tour that inspects the sphere; scaling can take care of the additive term. A full proof as well as full proofs of the results in the following sections are given in [?].

## 2 Lower Bound

In this section, we sketch a proof of the fact that any curve for the sphere inspection problem has length at least  $2d$ . Towards this, let  $\gamma$  be a curve in  $\mathbb{R}^d$  that inspects the unit sphere. We cut  $\gamma$  into a minimum number of contiguous portions of length at most  $\delta$  for some fixed  $\delta < 2$ . Let  $\xi_1, \dots, \xi_n$  be the resulting tour portions, where  $n = \lceil |\gamma|/\delta \rceil$ . Choose a portion  $\xi$ , and let  $x$  be its midpoint. Clearly  $\xi$  is contained in the ball  $B$  that has center  $x$  and radius  $\delta/2$ . Further define  $C$  to be the cone that is the intersection of all halfspaces that contain both  $B$  and  $S^{d-1}$  and whose defining hyperplanes are tangent to both  $B$  and  $S^{d-1}$ . Note that the set of points on the sphere that can be seen by the curve  $\xi$  can also be seen from the apex of  $C$ , as visualized by Figure 1. This holds since the radius of  $B$  is  $\delta/2 < 1$ . Note that a single point  $p \in \mathbb{R}^d$  can see some subset of an *open* hemisphere  $H$  of the unit sphere. Let  $H_1, \dots, H_n$  denote a set of open hemispheres such that  $H_i$  covers the portion of the sphere seen by  $\xi_i$ . Since  $\gamma$  inspects the sphere, we have that  $(H_i)_{i=1}^n$  covers the sphere. The claim now follows from the fact that the number of open hemispheres required to cover  $S^{d-1}$  is  $d + 1$ .

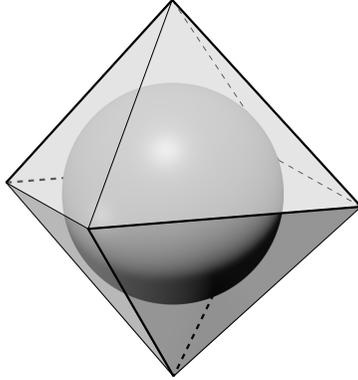


Figure 2: The cross polytope that has vertices at distance  $\sqrt{d}$  from the origin contains the unit sphere, and thus, a tour of its vertices (thick) inspects the unit sphere.

### 3 Upper Bound

In this section we sketch a proof of the fact that there is a curve for the sphere inspection problem of length  $O(d^{3/2})$ . Define  $\bar{C}^d := \{x \in \mathbb{R}^d : \|x\|_1 \leq \sqrt{d}\}$  to be the  $d$ -dimensional cross-polytope scaled up by a factor of  $\sqrt{d}$ , as shown in Figure 2. It is not difficult to see that the unit sphere is (just) included in  $\bar{C}^d$  and that this implies that the vertices of  $\bar{C}^d$  see the unit sphere. Hence, a closed tour that includes these vertices inspects the sphere. Since the graph of  $\bar{C}^d$  is Hamiltonian and the distance between any two adjacent vertices is  $\sqrt{2d}$ , there is such a tour of length  $2d \cdot \sqrt{2d} \in O(d^{3/2})$ .

### 4 Conclusion

We narrowed down the optimal competitive ratio for the  $d$ -Dimensional Hyperplane Search Problem to  $\Omega(d) \cap O(d^{3/2})$ . The obvious open problem is closing this gap. Applications and variants of this fundamental problem are very conceivable.

**Acknowledgement.** This work was supported in part by the Independent Research Fund Denmark, Natural Sciences, grant DFF-0135-00018B.

### References

- [1] Antonios Antoniadis, Krzysztof Fleszar, Ruben Hoeksma, and Kevin Schewior. A PTAS for Euclidean TSP with hyperplane neighborhoods. *ACM Trans. Algorithms*, 16(3):38:1–38:16, 2020.

- [2] Antonios Antoniadis, Ruben Hoeksma, Sándor Kisfaludi-Bak, and Kevin Schewior. Online search for a hyperplane in high-dimensional Euclidean space. *Information Processing Letters*. Forthcoming.
- [3] Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.
- [4] Anatole Beck and D. J. Newman. Yet more on the linear search problem. *Israel Journal of Mathematics*, 8(4):419–429, 1970.
- [5] Adrian Dumitrescu and Csaba D. Tóth. The traveling salesman problem for lines, balls, and planes. *ACM Trans. Algorithms*, 12(3):43:1–43:29, 2016.
- [6] Steven R Finch and Li-Yan Zhu. Searching for a shoreline. *arXiv preprint math/0501123*, 2005.
- [7] Mohammad Ghomi and James Wenk. Shortest closed curve to inspect a sphere. *Journal für die reine und angewandte Mathematik (Crelle's Journal)*, 2021(781):57–84, 2021.
- [8] Elmar Langetepe. On the optimality of spiral search. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–12, 2010.

# Probabilistic Real-Time Scheduling

Georg von der Brüggen (Speaker) \*      Sergey Bozhko †  
Kuan-Hsun Chen ‡      Jian-Jia Chen §      Björn B. Brandenburg ¶

---

The proposed talk will introduce the topic of probabilistic real-time scheduling analysis and give an overview over our recent results published in RTSS 2021 [1, 3], DATE 2019 [6], ECRS 2018 [2], and SIES 2017 [5].

## 1 Introduction

In a classical, deterministic scheduling analysis for hard real-time systems, recurrent tasks are usually described by a tuple  $\tau_i = (C_i, D_i, T_i)$ , where  $C_i$  is the task's worst-case execution time (WCET),  $D_i$  its relative deadline, and  $T_i$  its minimum inter-arrival time. The schedulability is determined under the assumption that all tasks must fulfill their timing requirements under all circumstances. Hence, it is assumed that all jobs execute according to their WCET, which can be significantly larger than the average-case execution time in practice. However, providing schedulability guarantees under this pessimistic assumptions not only risks to significantly underutilize the processor in the average case, as it considers exceptional execution conditions — even certain safety-critical systems can tolerate occasional deadline misses as long as their probability can be quantified. As a result, there is increasing interest in probabilistic schedulability and timing analysis in the real-time systems community (a survey was provided in 2019 [7]).

The deadline misses are usually quantified by either the deadline miss rate (which describes the percentage of deadline misses in the long run) or the worst-case deadline failure probability (WCDFP) (which is an upper bound on the probability of observing the first deadline miss of a task under analysis in a given busy window). While the deadline miss rate may seem as the more intuitive metric, we focus on the WCDFP. It allows to lower-bound the mean time to (temporal) failure of a system and directly corresponds to the deadline miss rate if jobs that miss their deadline are directly aborted. Furthermore, it is an important subroutine in the only known analytical approach to determine the deadline miss rate if tardy jobs are not aborted, which was provided by Chen et al. [4] in 2018.

---

\* [georg.von-der-brueggen@tu-dortmund.de](mailto:georg.von-der-brueggen@tu-dortmund.de). TU Dortmund University, Germany

† [sbozhko@mpi-sws.org](mailto:sbozhko@mpi-sws.org). Max Planck Institute for Software Systems (MPI-SWS), Germany

‡ [kuan-hsun.chen@utwente.nl](mailto:kuan-hsun.chen@utwente.nl). University of Twente, The Netherlands

§ [jian-jian.chen@tu-dortmund.de](mailto:jian-jian.chen@tu-dortmund.de). TU Dortmund University, Germany

¶ [bbb@mpi-sws.org](mailto:bbb@mpi-sws.org). Max Planck Institute for Software Systems (MPI-SWS), Germany

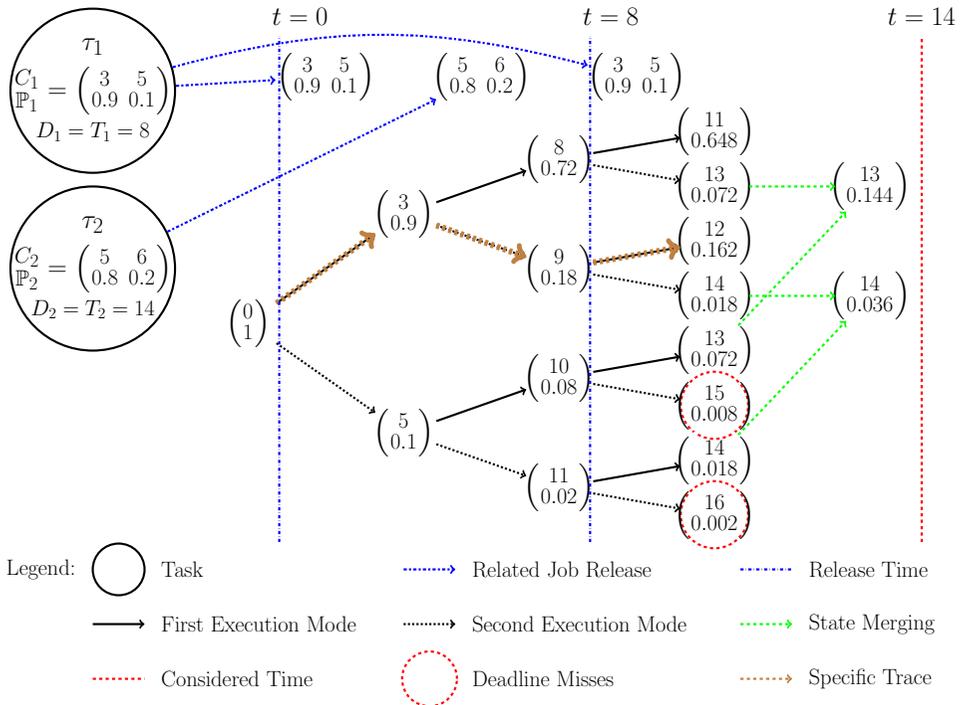


Figure 1: A convolution example for two task under rate-monotonic scheduling.

## 2 Probabilistic Analysis: Basics and Problems

For a probabilistic scheduling analysis, the execution time is not a deterministic value. Instead, it is either modelled as a probabilistic distribution or as a set of pairs of possible modes, each with ET and related probability, e.g.,  $C_i = \begin{pmatrix} 3 & 5 \\ 0.9 & 0.1 \end{pmatrix}$  means  $\tau_i$  has ET 3 with probability 0.9 and ET 5 with probability 0.1.

If the release pattern of jobs is known, the probability for jobs to miss their deadline under a given scheduling algorithm can be determined using job-level convolution (JLC) as shown in Figure 1 for static-priority scheduling. That is, we start with an initial state with ET 0 and probability 1 and jobs are convolved one by one with the current states, in each step adding up the ETs while multiplying the probabilities. By this process, the probability that the job of  $\tau_2$  meets its deadline at  $t = 8$  or at  $t = 14$  can easily be determined, as JLC considers all possible combinations of job costs. However, Figure 1 also shows one of the main problems we face in probabilistic analysis: the number of states is exponential in the number of jobs for a JLC. Therefore, it can only directly be applied if the number of jobs and the number of possible release patterns we have to consider are small. Maxim and Cucu-Grosjean showed the worst-case scenario for static-priority scheduling at RTSS 2013 [8] and the worst-case pattern for EDF scheduling was established in RTSS 2021 [3]. However, both scenarios require more efficient calculation methods than JLC to handle task sets with practically relevant size.

### 3 Efficient Approximation of Miss Probabilities

JLC can be sped up reducing the number of states by re-sampling [8], that is, if the number of states exceeds a certain threshold states are collapsed until the set is small enough. This procedure, however, reduces the precision of the calculation in a way that depends on the concrete re-sampling scheme in a non-trivial manner.

Instead of considering all possible combinations of job costs at the same time, the Monte-Carlo Response Time Analysis [1] analyzes individual job traces. That is, in each iteration, one specific trace (e.g., the one indicated with brown arrows in Figure 1) is analyzed by, for each job, taking one of the ETs considering the related probabilities. This procedure is performed multiple times, counting the number of iterations and of deadline misses. Afterwards, the WCDFP can be determined within a given interval based on the number of iterations and of deadline misses.

The task-level convolution [2] (TLC) does not consider jobs in order of arrival, but each relevant interval individually (e.g., the intervals [0,8] and [0,14] in Figure 1). TLC can calculate the probability for each interval more efficiently, utilizing the observation that, if a task releases multiple jobs in a given interval, only the number of jobs in a specific mode matters but not the concrete order.

Lastly, analytic bounds also estimate the probability for individual intervals. However, they do not consider modes for individual jobs, but estimate that the probability that the workload in a given interval is larger than the interval length directly, using Chernoff Bounds [5, 6] or Hoeffding's or Bernstein inequalities [2].

### References

- [1] S. BOZHKO, G. VON DER BRÜGGEN AND B. B. BRANDENBURG *Monte Carlo Response-Time Analysis* (RTSS 2021, Best Paper)
- [2] G. VON DER BRÜGGEN, N. PIATKOWSKI, K.-H. CHEN, J.-J. CHEN AND K. MORIK *Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems* (ECRTS 2018)
- [3] G. VON DER BRÜGGEN, N. PIATKOWSKI, K.-H. CHEN, J.-J. CHEN, K. MORIK AND B. B. BRANDENBURG *Efficiently Approximating the Worst-Case Deadline Failure Probability under EDF* (RTSS 2021)
- [4] K.-H. CHEN, G. VON DER BRÜGGEN AND J.-J. CHEN *Analysis of Deadline Miss Rates for Uniprocessor Fixed-Priority Scheduling* (RTCSA 2018, Outstanding Paper)
- [5] K.-H. CHEN AND J.-J. CHEN *Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors* (SIES 2017)
- [6] K.-H. CHEN, N. UETER, G. VON DER BRÜGGEN AND J.-J. CHEN *Efficient Computation of Deadline-Miss Probability and Potential Pitfalls* (DATE 2019)
- [7] R. I. DAVIS AND L. CUCU-GROSJEAN *A Survey of Probabilistic Schedulability Analysis Techniques for Real-Time Systems* (LITES 2019)
- [8] DORIN MAXIM AND LILIANA CUCU-GROSJEAN *Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters* (RTSS 2013)

# Restricted Adaptivity in Stochastic Scheduling

Guillaume Sagnol \*    Daniel Schmidt genannt Waldschmidt (Speaker) †

---

## 1 Introduction

Load balancing problems are one of the most fundamental problems in the field of scheduling, with applications in various sectors such as manufacturing, construction, communication or operating systems. The common challenge is the search for an efficient allocation of scarce resources to a number of tasks. While many variants of the problem are already hard to solve, in addition one may have to face uncertainty regarding the duration of the tasks; one way to model this is to use stochastic information learned from available data.

In contrast to the solution concept of a schedule in deterministic problems, we are concerned with *non-anticipatory policies* in stochastic scheduling problems. Such a policy has the ability to react to the information observed so far. While this adaptivity can be very powerful, there are situations where assigning resources to jobs prior to their execution is a highly desired feature, e.g. for the scheduling of healthcare services. This is especially true for the daily planning of elective surgery units in hospitals, where a sequence of patients is typically set in advance for each operating room. In this work, we present and analyze semi-adaptive policies, which allow one to control the level of adaptivity of the policy.

## 2 Problem Formulation

We consider the stochastic scheduling problem of minimizing the expected makespan on  $m$  parallel identical machines – denoted by  $P || \mathbb{E}[C_{\max}]$  using the three-field notation [2]. The input consists of a set of  $n$  jobs  $\mathcal{J}$  and a set of  $m$  parallel identical machines  $\mathcal{M}$ . Each job  $j \in \mathcal{J}$  is associated with a non-negative random variable  $P_j$  representing the processing time of the job. The processing times are assumed to be (mutually) independent and to have finite expectation. The task is to find a non-anticipatory policy minimizing the expected makespan  $\mathbb{E}[C_{\max}] := \mathbb{E}[\max_{j \in \mathcal{J}} C_j]$ , where  $C_j$  denotes the (random) completion time of job  $j$  under the considered policy. Roughly speaking, a non-anticipatory policy may,

---

\*sagnol@math.tu-berlin.de. Department of Mathematics, TU Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany.

†dschmidt@math.tu-berlin.de. Department of Mathematics, TU Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany.

at any point in time  $t$ , decide to start a job on an idle machine or to wait until a later decision time. However, it may not anticipate any future information of the realizations. A subclass of policies are constituted by the fixed assignment policies that have to irrevocably assign jobs to machines beforehand and starts the jobs on each machine without idle time.

While any (adaptive) list scheduling policy achieves an approximation ratio of 2 [1], any (non-adaptive) fixed assignment policy has performance guarantee  $\Omega\left(\frac{\log m}{\log \log m}\right)$ .

### 3 Our Contribution

Although the performance of fixed assignment policies are worse, there are applications in which non-adaptive policies are desired. We introduce the two classes of  $\delta$ -delay and  $\tau$ -shift policies whose degree of adaptivity can be controlled by a parameter. These semi-adaptive policies interpolate between the two extremes of non-adaptive fixed assignment policies and adaptive policies.

**Definition 1 ( $\delta$ -delay and  $\tau$ -shift policies)** *A  $\delta$ -delay policy for  $\delta > 0$  is a non-anticipatory policy which starts with a fixed assignment of all jobs to the machines and which may, at any point in time  $t$ , reassign not-started jobs to other machines with a delay of  $\delta$ : the reassigned jobs are not allowed to start before time  $t + \delta$ .*

*A  $\tau$ -shift policy for  $\tau > 0$  is a non-anticipatory policy which starts with a fixed assignment of all jobs to the machines and which may reassign jobs to other machines, but only at times that are an integer multiple of  $\tau$ .*



Figure 1: Snippets of the execution of a  $\delta$ -delay policy.



Figure 2: Snippets of the execution of a  $\tau$ -shift policy.

Observe that for  $\delta, \tau \rightarrow 0$  we recover the class of non-anticipatory policies in general, whereas for  $\delta, \tau \rightarrow \infty$  we recover the class of fixed assignment policies. Our main result is the design of an approximative policy that is a  $\delta$ -delay and  $\tau$ -shift policy.

**Theorem 2** *There exists a  $\delta$ -delay and  $\tau$ -shift policy  $\Pi_{\delta,\tau}$  with  $\delta, \tau \in \mathcal{O}(OPT)$  that is an  $\mathcal{O}(\log \log m)$ -approximation. Moreover, there is a matching lower bound.*

In other words, an exponential improvement on the performance of any fixed assignment policy can be achieved when allowing a small degree of adaptivity.

The policy we analyze can in fact be seen as a generalization of the list policy Longest Expected Processing Times First (LEPT), which waits for predefined periods  $\tau$  before reassigning in a LEPT-fashion the non-yet started jobs, taking the delay of  $\delta$  into account. The main idea of the proof for the upper bound is to show that at the beginning of each period, there is a constant fraction of available machines with high probability. As a result, after  $\mathcal{O}(\log \log m)$  reassigning steps, the load of the remaining jobs is small with high probability. The lower bound results from the following instance with  $n = Nm$  jobs over  $m$  machines for a large integer  $N$ : Each job has processing time  $P_j \sim \text{Bernoulli}(\frac{1}{N})$ , i.e.  $P_j = 1$  with probability  $\frac{1}{N}$ , and  $P_j = 0$  otherwise.

For more details we refer to the preprint version [3] and the conference proceedings of ESA 2021 [4].

## References

- [1] RONALD L. GRAHAM. *Bounds for certain multiprocessing anomalies*. In Bell System Technical Journal, volume 45, pages 1563–1581. Wiley Online Library, 1966.
- [2] RONALD L. GRAHAM, EUGENE L. LAWLER, JAN KAREL LENSTRA AND ALEXANDER H.G. RINNOOY KAN. *Optimization and approximation in deterministic sequencing and scheduling: a survey*. In Annals of Discrete Mathematics, volume 5, pages 287–326. Elsevier, 1979.
- [3] G. SAGNOL AND D. SCHMIDT GENANNT WALDSCHMIDT. *Restricted Adaptivity in Stochastic Scheduling*. In arXiv:2106.15393, 2021.
- [4] G. SAGNOL AND D. SCHMIDT GENANNT WALDSCHMIDT. *Restricted Adaptivity in Stochastic Scheduling*. In Proceedings of 29th Annual European Symposium on Algorithms, volume 204, pages 79:1–79:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

# Data transfer scheduling for deep space exploration

Emmanuel Hebrard (Speaker)\*    Christian Artigues\*    Pierre Lopez\*

Arnaud Lusson\*    Steve Chien†    Adrien Maillard†

Gregg Rabideau†

---

## 1 The overlapping memory dumping problem

Scientific instruments for deep space exploration spacecrafts become more and more sophisticated and consequently produce more and more data that must be sent to Earth. Data management is highly critical as the onboard memory is limited and communication with Earth is often a bottleneck. In this paper we consider the same case as in [2] in the context of the Rosetta/Philae mission, where the data production plan is known and the problem consists in planning memory dumps. Data is produced into several memory buffers and the goal is to avoid data loss, which occurs when data are produced into a full buffer. More precisely, we want to maximize the minimum margin, where the margin is the percentage of a buffer's capacity left free. This objective aims at designing robust plans where unexpected changes of the dump or fill rates can be absorbed by the margins. Data can only be dumped when the spacecraft is visible from Earth. This is materialized by consecutive disjoint downlink windows. Data dumping is a semi-automatized process. For each downlink window a priority has to be assigned to each buffer, then, the transfers follow this priority ordering. We therefore consider the problem of computing a priority assignment that maximizes the minimum margin.

In the overlapping Memory Dumping Problem (oMDP), we are given  $m$  downlink windows, where  $[s_j, e_j]$  stands for the time interval in which the downlink  $j$  is available, and  $\delta_j$  for the dump rate for transfers of downlink  $j$ . Moreover, there are  $n$  memory buffers, where  $C_i$  stands for the capacity of buffer  $i$ ;  $r_i(j)$  for the maximum handover (residual) usage of buffer  $i$  at the end of downlink window  $j$ ; and  $f_i : \mathbb{R} \mapsto \mathbb{R}^+$  for the piece-wise constant fill rate function of buffer  $i$  over time. Let  $\nu$  be the number of inflection points over all fill rate functions.

The transfers can be controlled by setting a priority function over the buffers defining a *ranking*. At every step, one of the buffers of highest priority among those who have a packet in memory is selected via round-robin to transfer a packet.

---

\*{hebrard,artigues,lopez}@laas.fr. LAAS-CNRS, Université de Toulouse, CNRS, France

†{steve.a.chien,adrien.maillard,gregg.r.rabideau}@jpl.nasa.gov. Jet Propulsion Laboratory, California Institute of Technology, CA, USA

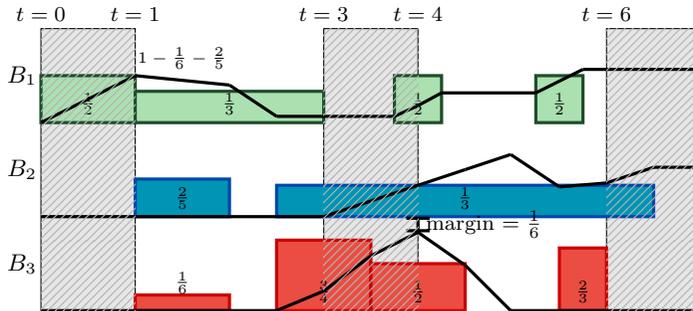


Figure 1: Example with 3 buffers. White areas indicate visibility.

Let  $U_i : \mathbb{R} \mapsto \mathbb{R}$  be the quantity of data on buffer  $i$  over time and  $g_i : \mathbb{R} \mapsto \mathbb{R}$  be the transfer rate out of buffer  $i$  over time.

In between downlink window, all transfer rates are null and the memory usage growth is the fill function, for every buffer  $i$ . In particular, for any time  $t$  preceding the start of the first downlink  $s_1$ , we have  $g_i(t) = 0$  and  $U_i(t) = \int_0^t f_i(x) dx$ .

During a downlink window with dump rate  $\delta$ , however, the effective transfer rate at time  $t$  depends on the priority function  $P$ , and on the usages and fill rates.

We consider the decision problem of finding a priority ranking for every downlink, such that the peak usage of any buffer (given its fill rate functions, and the data transfer system described above) is less than its capacity, i.e., without *data loss*. The objective function is actually to maximize the minimum margin,  $\min\{M(i) \mid i \in B\}$ , where  $M(i)$  is defined as one minus the ratio between the peak usage and the capacity of buffer  $i$ . However, it can be achieved by dichotomic search using an algorithm for the decision problem above.

**Example 1.** Figure 1 shows a plan with two downlinks, both with dump rate 1, and 3 buffers all of capacity 1. Fill rates are figured by colored rectangles, e.g.,  $f_1(t) = \frac{1}{2}$  for  $t \in [0, 1]$ . In the first downlink, all buffers have equal priority  $P_1(1) = P_1(2) = P_1(3) = 1$ , in the second, buffer 3 has the highest priority then buffer 2 then buffer 1. The black curves stand for the resulting buffer usage over time  $U_i(t)$ . The minimum margin is  $\frac{1}{6}$ .

## 2 Complexity and algorithms

The software used for the real mission (DALLOC) is described in [2] and relies on a heuristic called DOWNLINKCOUNT. In a nutshell, this method assigns priorities based on when each buffer would exceed a given target margin. More precisely, it counts how many downlink windows would occur before exceeding the target if no data could be dumped. At each downlink window  $j \in [1, m]$ , the transfers are

computed starting from the current usage, and with a dump rate equal to 0. Then a priority assignment is extracted as defined above, and window  $j$  is simulated using the same procedure but with dump rate  $\delta_j$ . It is important to notice that the choice of target margin can change the priority assignment. Therefore, in DALLOC, this heuristic is used within a binary search. The resulting algorithm is denoted ITERATEDLEVELING.

As the complexity of the problem was not established, we first consider the decision problem PRIORITYALLOC: “Is there a priority assignment for each downlink window such that there is no data loss?”.

First, we show that the problem is in NP because, given a priority assignment, simulating the transfers and therefore verifying a certificate, can be done in polynomial time, using similar arguments as in [1]. This is not trivial to see that not every packet of data need to be tracked individually, and the number of packets can be exponential in the size of the instance. Moreover, to compute the usage over time of a given buffer, we only need to know the set of buffers of strictly higher priority and the set of buffers of equal priority (Property 1). In other words, the exact priority assignment within the set with higher priority is irrelevant. We define a procedure (denoted Algorithm 1 from now on) to compute the memory usages (and so the minimum margin) in  $O((\nu + m)(\log(\nu + m) + n^2 \log n))$  time, given a priority assignment  $P$  on each downlink window. Then we use a reduction from partition to show that PRIORITYALLOC is NP-complete.

Next we consider the single-window problem ( $m = 1$ ). Thanks to Property 1 and Algorithm 1, we can compute the margin  $M_{\Gamma \prec \Omega}(i, j)$  of buffer  $i$  in window  $j$  when the buffers in the set  $\Gamma$  have a strictly higher priority than  $i$ , and the buffers in the set  $\Omega$  have the same priority as  $i$ . We propose a method (Algorithm 2) to solve PRIORITYALLOC on a single downlink window with  $O(n^2)$  calls to Algorithm 1.

Finally, for the general case, we propose a heuristic (REPAIRDESCENT) which iteratively computes a seed priority assignment using the heuristic DOWNLINKCOUNT as greedy “descent” and then “repairs” it to achieve a strictly higher target margin.

The repair procedure runs the current priority assignment using Algorithm 1 until reaching downlink  $j$  at which the target margin is exceeded. Then, it calls Algorithm 2 to check whether there exists a priority assignment  $P_j$  that achieves the expected margin. If there is such an assignment, it is run by Algorithm 1 and we advance to downlink window  $j + 1$ . If the last window is reached, an improving solution has been found. Otherwise, if downlink  $j$  does not have a priority assignment without data loss, Algorithm 2 returns a set  $B$  of buffers guaranteed to exceed their target margin given the current usage at the end of downlink  $j - 1$ , even if they are given (globally) the highest priority. Therefore, the only way to avoid data loss in this downlink is to reduce the residual load for at least one of these buffers at the end the previous downlink. In order to avoid branching on the possible ways of reducing this load, we instead add one randomly chosen handover constraint. We pick a random buffer  $i$  in the set  $B$ , and we set its maximum handover value  $r_i(j - 1)$  to the current usage  $U_i(j - 1)$  to which we subtract the

gap we need to achieve the expected margin: the margin we obtain when buffer  $i$  is given highest priority with all other buffers in  $B$  (that is,  $M_{\emptyset \rightarrow B}(i, j)$ ) minus the expected margin  $obj$ . Then the previous window is solved again with this new constraint: it “backtracks” by decreasing the current downlink window  $j$ . If the current downlink window is the first ( $j = 1$ ), then we stop.

### 3 Experimental Evaluation

We used the same data set as in [2], constituted of four scenarios (MTP1, MTP2, MTP3 and MTP4), corresponding to the whole activity sequence of Rosetta divided in four quarters. The whole sequence has over 40,000 data production events for 16 memory buffers and 324 downlink windows over 3 months. Table 1 give the margin value as well as average CPU times for: the results of DOWNLINKCOUNT published results [2]; our implementation using Algorithm 1 of the same heuristic; ITERATEDLEVELINGre-implemented using our version of DOWNLINKCOUNT and finally REPAIRDESCENT. We first observe that our reimplementation is much faster than the original one. Moreover, REPAIRDESCENT finds the most robust download plan for the hardest real scenario (MTP4). As ITERATEDLEVELING, it matches an upper bound on MTP1, MTP2, and MTP3.

	Margin				Avg. CPU
	MTP1	MTP2	MTP3	MTP4	
Upper Bound	46.4	72.5	54.8	53.4	
DALLOC’s DownlinkCount	40.4	46.5	17.8	29.8	14.6
Our implementation of DownlinkCount	46.4	68.1	17.8	30.8	0.018
ITERATEDLEVELING	46.4	72.5	54.8	48.5	0.116
REPAIRDESCENT	46.4	72.5	54.8	<b>52.8</b>	0.088

Table 1: Comparison with the state of the art on real scenarios.

Finally, on a much larger randomly generated data set, REPAIRDESCENT is consistently faster than ITERATEDLEVELING to find solutions of similar quality and is able to significantly improve the minimum margin when given more CPU time.

## References

- [1] G. SIMONIN, C. ARTIGUES, E. HEBRARD AND P. LOPEZ (2015). Scheduling scientific experiments for comet exploration. *Constraints: An International Journal* 20(1), 77–99.
- [2] G. RABIDEAU, S. CHIEN, M. GALER, F. NESPOLI AND M. COSTA SITJÀ (2017). Managing Spacecraft Memory Buffers with Concurrent Data Collection and Downlink. *Journal of Aerospace Information Systems* 14 (12), 637–651.

# Better Algorithms for Online Bin Stretching via Computer Search

Matej Lieskovský \*

---

ONLINE BIN STRETCHING, introduced by Azar and Regev [1] is an assignment problem similar to ONLINE BIN PACKING. A sequence of items of size between 0 and 1 arrive and each must be assigned a bin before processing the next item. We are assured that the items fit into  $m$  bins of size 1. We must also use at most  $m$  bins but we get to increase their size to what is known as the *stretching factor*  $\alpha$ .

Kellerer et al. [5] showed an  $\alpha = 4/3 < 1.334$  algorithm for two bins and proved it to be optimal. Azar and Regev [1] showed an algorithm with  $\alpha = 13/8 = 1.625$  for any  $m$  and  $\frac{5m-1}{3m+1}$  for  $3 \leq m \leq 21$ . Better results have been found since then with best known results being by Böhm et al. [2] achieving  $\alpha = 11/8 = 1.375$  for  $m = 3$  and  $\alpha = 3/2 = 1.5$  for all  $m$ .

A lower bound of  $4/3$  was first shown for  $m \in \{2, 3\}$  by Kellerer et al. [5] and then generalised to any  $m \geq 2$  by Azar and Regev [1]. Gabay et al. [4] introduced the idea of using computer search to find new lower bounds for small values of  $m$ . This approach was then improved by Böhm and Simon [3] who proved a lower bound of  $56/41 > 1.365$  for  $m = 3$  and  $19/14 > 1.357$  for  $4 \leq m \leq 8$ .

We modify the computer search approach so that it can be used for finding new algorithms for small values of  $m$ . This yields new algorithms, improving  $\alpha$  for  $m = 4$ ,  $m = 5$  and  $m = 6$  significantly.

Number of bins	4	5	6
Previous upper bound	$\frac{19}{13} < 1.462$	$\frac{3}{2} = 1.5$	$\frac{3}{2} = 1.5$
Our upper bound	$\frac{7}{5} = 1.4$	$\frac{27}{19} < 1.422$	$\frac{19}{13} < 1.462$
Granularity (see Section 1)	25	19	13
Time needed (hours)	880	1259	27

The computer search was done using a server with an Intel Xeon E5-2630 v3 CPU and 126 GB of RAM. The time complexity grows rapidly with increasing  $m$  and granularity, but we do not believe that we have reached any fundamental limit of this method. We are working on improving the algorithm to enable further search.

---

\*ml@iuuk.mff.cuni.cz. Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Prague, Czechia. Partially supported by GA ČR project 19-27871X.

Granularity settings resulting in run times exceeding 14 days have been tested with no success. For three bins, we found an upper bound of  $76/55 < 1.382$  using granularity 55 with a run time of 413 hours. This is worse than the previous upper bound of  $11/8 = 1.375$  by Böhm et al. [2] For  $m = 7$  we did not find an upper bound better than  $3/2 = 1.5$ , mostly due to granularity being limited to at most 11.

## 1 Defining Rounded Game

We wish to view ONLINE BIN STRETCHING as a game between two players, ALGORITHM and ADVERSARY. In every round, ADVERSARY generates an item which ALGORITHM must place into one of the  $m$  bins. The ALGORITHM is victorious when the items generated by the ADVERSARY can be proven to no longer fit into  $m$  offline bins of size 1 or when ADVERSARY resigns. The ADVERSARY is victorious when any online bin exceeds size  $\alpha$ . If a winning strategy exists for the ALGORITHM, there exists an algorithm for ONLINE BIN STRETCHING with  $m$  bins that requires a stretching factor of no more than  $\alpha$ . We shall call this game the REAL GAME.

The main obstacles to implementing a search of REAL GAME are the following:

1. ADVERSARY has an infinite selection of item sizes to pick from.
2. By sending arbitrarily small items, ADVERSARY can make REAL GAME last arbitrarily many rounds.
3. Proving that the items generated by ADVERSARY so far do not fit into the offline bins can be computationally intensive.

In order to avoid these problems, we analyse a simplified version of REAL GAME that we shall call ROUNDED GAME and then prove that a winning strategy for ALGORITHM in ROUNDED GAME translates to an algorithm for ONLINE BIN STRETCHING. We do this by defining ROUNDED GAME in a manner that corresponds to REAL GAME where ALGORITHM is restricted in its decision-making process and ADVERSARY is permitted to cheat to a limited extent.

ROUNDED GAME is parametrized by three values—the number of bins  $m$ , the granularity level  $k$  and the target bin size  $s$ —and corresponds to REAL GAME with  $m$  bins and stretching factor  $s/k$ . We scale the instance by  $k$  and work with integer values from now on. This makes the size of the offline bins  $k$  and the size of the online bins  $s$ .

We classify the items by their size into  $k$  classes from 0 to  $k - 1$ , where an item of class  $c$  has size in the range  $(c, c + 1]$ . We also similarly define fill levels of bins from 0 to  $s - 1$  with the sole difference being that bins containing volume 0 are considered to have a fill level 0 while items of size 0 are disregarded entirely. Observe that inserting a class  $c$  item will always increase the fill level of the bin by either  $c$  or  $c + 1$ . We call the latter case an overflow. ADVERSARY generates an item by specifying its class and its overflows. In ROUNDED GAME, we do not insist on the overflows being consistent. For example, item  $A$  can overflow on bin 1 and not on bin 2, be placed in bin 3, and the following item  $B$  can overflow on

bin 2 and not on bin 1.

We need to restrict `ROUNDED GAME` to a finite number of states. Consider a state defined by the pair  $(\mathcal{L}, \mathcal{H})$  where  $\mathcal{L}$  is the list of fill levels of the individual bins and  $\mathcal{H}$  is the multiset of non-zero classes of items that have been seen so far. We show that this leads to a finite number of game states that can be further reduced by more analysis.

We explore the resulting `ROUNDED GAME` by a `MINIMAX` algorithm. We prevent `ADVERSARY` from exceeding the maximum possible volume and, whenever `ADVERSARY` is about to win, use an ILP solver to check if the items can be packed into the offline bins. In order to attain higher granularity, we implement multiple optimizations, including smart caching based on us defining a partial order on game states. These optimizations are described in greater detail in the technical report. [6] This is how we achieve the results in the table.

## References

- [1] Y. Azar and O. Regev. On-line bin-stretching. *Theoretical Computer Science*, 268(1):17–41, 2001.
- [2] M. Böhm, J. Sgall, R. van Stee, and P. Veselý. A two-phase algorithm for bin stretching with stretching factor 1.5. *Journal of Combinatorial Optimization*, 34:810–828, 2017.
- [3] M. Böhm and B. Simon. Discovering and certifying lower bounds for the online bin stretching problem. arXiv:2001.01125, 2020.
- [4] M. Gabay, N. Brauner, and V. Kotov. Improved lower bounds for the online bin stretching problem. *4OR*, 15:183–199, 2017.
- [5] H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997.
- [6] M. Lieskovský. Better algorithms for online bin stretching via computer search. arXiv:2201.12393, 2022.

# A Multi-Phase Algorithm for Bin Stretching with Stretching Factor below 1.5

Martin Böhm <sup>\*</sup>      Matej Lieskovský <sup>†</sup>      Sören Schmitt (Speaker) <sup>‡</sup>  
Jiří Sgall <sup>§</sup>      Rob van Stee <sup>¶</sup>

---

ONLINE BIN STRETCHING is a semi-online variant of the well-studied assignment problem BIN PACKING. In 1998, Azar and Regev initiated the study of the following algorithmic problem. An (online) algorithm is presented with an input of items with sizes between 0 and 1, which are revealed one by one. Before the next item is revealed the algorithm must decide in which bin the item should be packed. In advance, the algorithm has access to the information that the complete input can be packed in  $m$  bins of size 1. In general, we are allowed to use at most  $m$  bins of capacity  $\alpha \geq 1$ . Our goal is to minimize the *stretching factor*  $\alpha$ .

Online bin stretching can also be viewed as semi-online makespan scheduling on  $m$  identical machines, where the optimal (offline) makespan is known in advance.

For any value  $m \geq 2$  a general lower bound of  $4/3$  was shown by Azar and Regev [1]. Böhm and Simon presented improved lower bounds of  $56/41 > 1.365$  for  $m = 3$  and  $19/14 > 1.357$  for  $4 \leq m \leq 8$  by computer search [3]. The current best algorithms are  $11/8$ -competitive for  $m = 3$  and  $3/2$ -competitive for all  $m$  by a subset of the authors of this paper [2]. For  $m = 2$  a simple algorithm achieves optimal  $4/3$ -competitiveness. In a separate MAPSP submission, Lieskovský presents improved upper bounds of  $31/22 < 1.409$  for  $m = 4$ ,  $23/16 < 1.438$  for  $m = 5$  and  $19/13 < 1.462$  for  $m = 6$  by a modified computer search approach [4].

Our work considers the case where the number of bins  $m$  is large and we are allowed resource augmentation. This means that we can use not only bins of size  $\alpha$ , but also a small constant number of additional bins. This allows for a much more readable algorithm and restricts the number of special cases.

Under these conditions, we present a multi-phase algorithm that is the first to break the crucial  $3/2$ -competitiveness, and point out some challenges that had to be overcome to achieve this.

---

<sup>\*</sup>boehm@cs.uni.wroc.pl. Combinatorial Optimization Group, Institute of Computer Science, University of Wrocław.

<sup>†</sup>ml@iuuk.mff.cuni.cz. Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Prague, Czechia. Partially supported by GA ČR project 19-27871X.

<sup>‡</sup>soeren.schmitt@uni-siegen.de. Department of Mathematics, University of Siegen, Germany.

<sup>§</sup>sgall@iuuk.mff.cuni.cz. Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Prague, Czechia. Partially supported by GA ČR project 19-27871X.

<sup>¶</sup>rob.vanstee@uni-siegen.de. Department of Mathematics, University of Siegen, Germany.

type	max size
top	10
big	8
large	$(15 - \varepsilon)/2$
half	$(15 + \varepsilon)/3$
3-stackable	$(15 - \varepsilon)/3$
small <sup>+</sup>	4
quarter	$10/3$
small <sup>-</sup>	2

Table 1: Item types and sizes

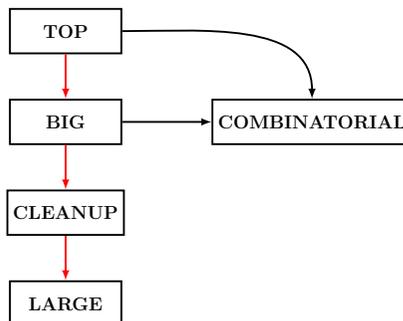


Figure 1: Overview of phases

For readability and for convenience we scale the offline bin size to 10 and the online bin size to  $15 - \varepsilon$ . Hence a positive  $\varepsilon$  implies a stretching factor of  $\alpha = (15 - \varepsilon)/10 < 3/2$ . Our notation for the different item types and their sizes can be seen in the table. The size of a top item is in the interval  $(8, 10]$ .

The decrease of the online bin size from 15 to  $15 - \varepsilon$  causes many additional difficulties. For instance, half items exist, so items that can be packed two per offline bin but online we gain no advantage because a half item may block a top item and three half items do not fit in an online bin. This appears to necessitate the additional item types that we introduce and the consideration of many new cases.

**Algorithm** Our algorithm uses multiple phases to deal with the different *threats* it faces while processing the input. The main threats are blocks of many items of one type arriving all at once, for instance  $m$  top items. These threats remain until enough volume has been packed. For example, before we have packed a volume of  $2m$  (resp.  $(5 + \varepsilon)m/2$ ), there is still the threat that  $m$  top (resp. big) items will arrive in the future.

Because top and big items are of size greater than  $(15 - \varepsilon)/2$ , these items need to be packed one per bin. Thus, if the adversary starts the input with items that can be combined with top items in offline bins, we cannot allow the algorithm to pack any bin above the threshold of  $5 - \varepsilon$ . This observation suggests that a good approach would be to first try to use the additional space of  $5 - \varepsilon$  that the algorithm has in each bin. However, any algorithm that is less than  $3/2$ -competitive must also deal with the threat of  $2m$  items of size between  $(15 - \varepsilon)/3$  and  $(15 + \varepsilon)/3$  (half items) without being able to pack these items more efficiently than the offline algorithm. This is one reason why it is so challenging to achieve a competitive ratio less than  $3/2$ . Instead of 5, we use  $(15 + \varepsilon)/3$  as upper bound for half items. This way no large item can be combined with an half item in an offline bin and we can argue that whenever two half items arrive the threat of large items decreases by one. Hence, in the beginning we pack small items by FIRST FIT in slots of size  $5 - 5\varepsilon/3$  instead of  $5 - \varepsilon$ .

The main line of our algorithm (shown in red in the figure) is concerned with

the case where many small items arrive at the beginning. In the so-called *top phase* we pack them to at most  $5 - 5\epsilon/3$  per bin, making sure that  $m$  top items still fit. In the *big phase* we make sure that no small item is packed above  $7 - \epsilon$  in any bin. Further, the number of bins containing total size more than  $5 - \epsilon$  (but no top item) plus the number of top items in the input is at most  $m$  (including any top items that may still arrive in the future). Then, in the *large phase* we start filling up these bins, making sure to pack at least  $(15 - \epsilon)/2$  additional per bin on average. This ensures that the threat of big items never becomes a problem. Between the big and large phase, we transition into the intermediate *cleanup* phase that takes care of special items and bins.

We plan to end the top phase if it is true that whenever we pack additional 8 volume, the possible number of future top items is reduced by one. This happens, for example, if the input starts with a lot of small items with a total volume of  $2m$ . After the top phase the threat of top items decreases gradually and we allow to fill some bins over the threshold  $5 - 5\epsilon/3$ . In the big phase we gradually *enlarge* some of the slots dedicated to small items to  $7 - \epsilon$ , so that still any big item fits on top of these slots and any input with  $m$  big items can be packed. Note that at this point it is no longer possible for  $2m$  half items to arrive. We end the big phase once we have filled approximately  $3m/4$  of the bins (the actual condition is that a total size of  $(5 + \epsilon)m/2$  must have arrived).

If early in the input for instance some quarter items arrive, we also have the guarantee that the input will not contain  $m$  top items. However, it is not yet the case that the number of top items that may still arrive decreases whenever we pack an additional volume of 8. The condition for ending the top phase is purely based on combinatorics and the condition for ending the big phase is purely size-based.

While in the top or big phase it must still be possible for us to pack  $m$  large items. But these items are the largest items that we can pack more efficiently than the adversary by pairing them in previously empty bins. Hence, to deal with this threat we instead of never packing any bin over the threshold  $(15 - \epsilon)/2$  (before we have packed  $(5 - \epsilon)m$  volume), we rather strive to keep some of the bins empty. This is the reason why we keep half of the bins empty in the top phase and roughly  $m/4$  bins empty while in the big phase.

If the input does not start with a lot of small items we sometimes need to (or can) adapt our packing methods and transition to different phases. For example, if the input starts with  $k$  half items, we pack them twice per bin. By a combinatorial argument we know that in the future at most  $m - k/2$  top items can arrive, but we still do not have that whenever we pack additional 8 volume, we reduce the threat of future top items by one. We then transition into the so-called *combinatorial phase*. In this phase we combine any future items that are smaller than 3-stackable items with the pairs of half items. If this indeed happens, we may end up with a lot of bins that are filled to a very high level. We give a bound on the number and fill level of those bins, after which we can start using a very simple method to pack the entire input. If these bounds are met we are in what we call the *First Fit case*. On the other hand, if no or not enough such items arrive, we pack the

remaining input in the combinatorial phase.

To analyze our algorithm we use different kinds of volume and combinatorial arguments. We combine those arguments by defining and referring to *timed* value to decide when to transition between different phases. For an overview, refer to the figure.

We show that a very small amount of resource augmentation is enough to break the barrier of  $3/2$  for this problem. Future work is to reduce or eliminate the amount of resource augmentation required. We conclude our results in the following theorem.

**Theorem 1** *There exists a positive  $\varepsilon$  for which our algorithm is  $(15 - \varepsilon)/10$ -competitive, if it has access to a small amount of resource augmentation.*

## References

- [1] Yossi Azar and Oded Regev. On-line bin-stretching. *Theor. Comput. Sci.*, 268(1):17–41, 2001.
- [2] Martin Böhm, Jiří Sgall, Rob van Stee, and Pavel Veselý. A two-phase algorithm for bin stretching with stretching factor 1.5. *J. Comb. Optim.*, 34(3):810–828, 2017.
- [3] Martin Böhm and Bertrand Simon. Discovering and certifying lower bounds for the online bin stretching problem. *CoRR*, abs/2001.01125, 2020.
- [4] Matej Lieskovský. Better algorithms for online bin stretching via computer search. *CoRR*, abs/2201.12393, 2022.

# Resource Optimization for Program Committee Members: A Subreview Article \*

Michael A. Bender <sup>†</sup>    Samuel McCauley <sup>‡</sup>    Bertrand Simon <sup>§</sup>  
Shikha Singh (Speaker) <sup>‡</sup>    Frédéric Vivien <sup>¶</sup>

---

## 1 Introduction

Serving on the program committee (PC) for a flagship theory conference is rewarding for the PC members, but it is intense and time-consuming work. Each submitted manuscript is typically assigned to three or four PC members. As a result, when you serve on a PC, you are given somewhere between 10 and 60 manuscripts to review, depending on the committee. Reviews need to be completed rapidly—on the order of one month. For many of these papers, you have only a casual familiarity, meaning that doing a high-quality review will consume large amounts of time and may be fraught with uncertainty.

Consequently, most PC members rely on subreviewers to provide outside reviews. The subreviewer mechanism works as follows. When you serve on a PC, you decide which manuscripts require outside assistance. Then you scrape the Internet for subreviewers, sending request emails to friends, calling in favors, and searching out new domain experts.

Many of these subreview requests are denied. It is not uncommon that you need to send 3-8 review requests before you find a subreviewer that is not reading the paper for another PC member, already overburdened with reviews from the same conference, also serving on the PC, advising or being advised by one of the paper authors, having a baby, married to one of the paper authors, or just taking a well needed break from responding to email. Thus, even the administrative process of collecting subreviews is work consuming and worth optimizing.

One resource optimization, familiar to seasoned PC members, is the race to send out review requests the moment that the submissions have been meted out to the PC. The longer you wait to ask for a review from someone, the less likely

---

\*This paper appeared in the Proceedings of Conference on Fun with Algorithms 2016.

<sup>†</sup>Stony Brook University, Stony Brook, NY 11794-4400 USA.  
Email: [bender@cs.stonybrook.edu](mailto:bender@cs.stonybrook.edu)

<sup>‡</sup>Williams College, Williamstown MA 01267. Email: [{sam, shikha}@cs.williams.edu](mailto:{sam, shikha}@cs.williams.edu)

<sup>§</sup>Universität Bremen, FB3: Mathematik/Informatik, 28359 Bremen, Germany.  
Email: [bsimon@uni-bremen.de](mailto:bsimon@uni-bremen.de)

<sup>¶</sup>Univ Lyon, LIP, ENS Lyon, Lyon 69007 France. Email: [frederic.vivien@inria.fr](mailto:frederic.vivien@inria.fr)

it is that that person will comply. It is common that you miss out on collecting another subreviewer by just a few minutes—someone else’s request showed up in the mail queue first.

Another optimization is the use of parallelism: send requests to several potential subreviewers in parallel to increase the probability of a positive answer. But every request takes time, and if multiple people say “yes,” this leads to redundant work for the community and the PC member.

This paper studies this class of resource-allocation problem, which overburdened PC members naturally face while discharging their PC duties. Specifically, this paper studies the randomized resource-allocation problem of how to find subreviewers for the minimum expected cost. More generally, this paper explores how to delegate work to others, when delegation has a cost, a nontrivial probability of failing, and the delegator must take this into account in their decisions.

## 2 Scheduling Model

There are  $N$  papers to review, denoted  $1, \dots, N$ . Reviewing paper  $j$  yourself costs  $C_j$ . This cost is paper specific, because some submissions are easier for you to judge yourself, and others are harder. Each attempt to recruit a new subreviewer costs 1. The unit cost models the overhead of searching for email addresses, sending review requests, review reminders, thank you notes, requests for alternative reviewers, and all the other delegation overhead. Without loss of generality,  $C_j > 1$ ; otherwise, it is not cost-effective to recruit subreviewers.

There is a (round-dependent and paper specific) probability of  $p_{i,j}$  ( $0 \leq p_{i,j} \leq 1$ ) that at round  $i$  the candidate subreviewer declines to review paper  $j$ . Probability  $p_{i,j}$  is monotonically increasing in  $i$ , since as time goes on, any candidate subreviewer is increasingly likely to be reviewing the paper for someone else or already has all of his/her time accounted for.

The reviewing process proceeds in  $k + 1$  rounds. In the first round, you email out a bolus of subreview requests for papers  $1, 2, \dots, N$ . Some of these papers get successfully adopted by subreviewers. For those that do not, you send a second bolus of review requests in the second round, and so forth. In the last round  $k + 1$ , you need to review those stray papers that did not find a subreviewer owner in the first  $k$  rounds.

The objective is to minimize the total expected cost to review all  $N$  papers. Because we are dealing with expected values, we can analyze the subreviewer-finding strategy for each submission in isolation. Henceforth, we focus on the case of a single paper, and the multiple-paper case follows by linearity of expectation. We simplify notation by dropping the subscripts, writing  $C$  and  $p_i$ .

Modeling the process by a series of rounds faithfully captures how many PC members (including the authors on this paper) act when they serve on PCs. Round 1 opens with a flurry of energy and enthusiasm. It ends with a quiescent period filled with hope, despair, gratefulness, and occasional sleep. Round 2, 3, and so forth proceed similarly, as the PC members work up the gumption to

renew soliciting reviews, after getting more declines from potential subreviewers.

Although we describe this problem using the colorful language of beating the bushes for subreviewers, this model similarly captures a natural parallel scheduling problem. There are  $N$  tasks (the papers). The objective is to schedule all tasks while minimizing the expected cost. A task  $j$  can be executed locally (you write the review) or remotely (you assign the review to a subreviewer). A local execution is expensive. A remote execution is cheaper, but even launching a job has a cost and fails with a certain probability.

### 3 Results

We first explore the case where the rejection probabilities are known. We give a closed-form optimal solution for multiple rounds when the number of requests in each round is not required to be an integer. This closed-form solution gives intuition about how requests should be distributed over rounds, and is an important building block for our further results. Using the closed form solution, we construct an optimal algorithm for sending integral requests; the running time is linear in the number of rounds.

We then consider the bounded-reviewers case, where a submission may have only a bounded number  $R$  of knowledgeable experts. Once all  $R$  experts decline to review the paper, all resources are exhausted and the beleaguered committee member has no choice but to review the submission without help. We give an optimal algorithm if the probabilities remain constant, an approximation algorithm when they are monotonically increasing, and a pseudo-polynomial algorithm when papers have to share a budget  $R$ .

We also study the scheduling problem for the case where the probability that a subreviewer rejects the subreview request is constant but unknown. We give a two round strategy which is a  $(4\sqrt{C}/\ln C + 2)$ -approximation for any  $C \geq 2$ . We generalize this to a  $k$ -round strategy that is a  $k(C^{1/k} + 1)$ -approximation.

### 4 Related Work

In many papers in scheduling and distributed computing, processors randomly choose which task to execute from a pool of tasks. Because multiple processors may choose the same task, there is some probability of wasted work. This is the case in the so-called do-all or write-all literature (among a very large literature, see for instance [1, 6, 10]), as well as other contexts where processors randomly choose tasks to execute and may fail to find one that was not already chosen [2, 9].

The subreviewer scheduling problem is related to the problem of executing tasks on failure-prone platforms, and particularly to the work on replication for fault-tolerance [4, 5]. There is an important difference, however. Most work on fault-tolerance for failure-prone HPC platforms assume that failures are rare [8]; typically the mean time between failures of a component is expressed in tens of

years. This means that it is almost always better in practice only to duplicate failed tasks and not be proactive by replicating a priori. Furthermore, the low failure rate allows for first-order approximations [3, 7], which would be invalid in the current context of high rejection rates.

## References

- [1] Dan Alistarh, Michael A. Bender, Seth Gilbert, and Rachi Guerraoui. How to allocate tasks asynchronously. In *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 331–340, 2012.
- [2] Michael A. Bender and Cynthia A. Phillips. Scheduling DAGs on asynchronous processors. In *Proc. of the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 35–45, 2007.
- [3] George Bosilca, Aurélien Bouteiller, Élisabeth Brunet, Franck Cappello, Jack Dongarra, Amina Guermouche, Thomas Hérault, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. Unified Model for Assessing Checkpointing Protocols at Extreme-Scale. *Concurrency and Computation: Practice and Experience*, 26(17):2727–2810, 2013.
- [4] Henri Casanova, Fanny Dufossé, Yves Robert, and Frédéric Vivien. Mapping applications on volatile resources. *International Journal of High Performance Computing Applications*, 29(1):19, 2015.
- [5] Henri Casanova, Dounia Zaidouni, and Frédéric Vivien. Using replication for resilience on exascale systems. In Thomas Hérault and Yves Robert, editors, *Fault-Tolerance Techniques for High-Performance Computing*, page 50. Springer, 2015.
- [6] Bogdan S. Chlebus and Dariusz R. Kowalski. Cooperative asynchronous update of shared memory. In *Proc. of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 733–739, 2005.
- [7] John T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312, 2004.
- [8] Thomas Hérault and Yves Robert. *Fault-Tolerance Techniques for High-Performance Computing*. Springer, 2015.
- [9] Zvi M. Kedem, Krishna V. Palem, and Paul G. Spirakis. Efficient robust parallel computations. In *Proc. of the 22rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 138–148, 1990.
- [10] Dariusz R. Kowalski and Alexander A. Shvartsman. Writing-all deterministically and optimally using a nontrivial number of asynchronous processors. *ACM Transactions on Algorithms*, 4:33:1–33:22, 2008.

# Orthogonal schedules in Round Robin competitions

Roel Lambers (Speaker) \*      Jop Briët ‡      Viresh Patel †  
Frits Spieksma ‡      Mehmet Akif Yıldız ‡

---

## 1 Introduction

Scheduling major national or international competitions is a big logistic challenge. Typically, there are several stakeholders involved - the organizing association, the venues used, the cities where matches are played, the broadcasters that air the games and of course the teams and players that have to play.

Many competitions use a *Round Robin*-format, where every pair of teams meet each other a fixed amount of times, for example once to arrive at a Single Round Robin (SRR) or twice, to arrive at a Double Round Robin (DRR). In this work our main focus lies on the SRR competitions.

To schedule competitions while taking into account the wishes of all the stakeholders, a widely used approach is so-called First Break, Then Schedule (FBTS). When using FBTS, first the *Home-Away-Pattern* (HAP) is settled for all the teams, and as a second step the individual matches are scheduled.

Not all constraints depend solely on teams playing Home or Away, so as a scheduler, one wants to use HAPs that are *flexible* enough to adhere to the different needs. However, a priori, it is not clear how flexible the combined set of HAPs, *the HAP-set*, is.

Clearly, a match between two teams can only be played in rounds where the Home/Away assignment differs between the two. However, that is not a sufficient condition, as matches can be fixed or excluded from rounds, while the HAP-set at first glance shows more possibilities.

To give some insights in the flexibility, measures were introduced by [1], where it was shown that the commonly used HAP-sets have a low *width* - see section 2 for its definition. In particular, it is shown that single break HAP-sets have width 1, which means that in any feasible schedule, some matches are fixed to a particular round.

In this note we present results on the existence of HAP-sets that perform better with respect to the width, and construct HAPs that even have the highest possible

---

\*[r.lambers@tue.nl](mailto:r.lambers@tue.nl) Department of Mathematics & Computer Science. Eindhoven University of Technology, Eindhoven, Netherlands

†University of Amsterdam, Amsterdam, Netherlands

‡Eindhoven University of Technology, Eindhoven, Netherlands

flexibility.

## 2 Terminology

There is a set of teams  $\mathcal{T}$  with  $|\mathcal{T}| = N = 2n$ , that form a SRR over  $N - 1$  rounds. The HAP-set of the teams is denoted with  $\mathcal{H} = \{H(t) : t \in \mathcal{T}\}$ , where  $H(t) = (H_r(t))_{r \leq N-1}$ , a binary vector with 0 Home and 1 Away.

$S$  is given by  $S = \cup_r S_r$  where each  $S_r$  is a matching on the  $2n$  teams, in such a way that for every pair of distinct teams  $t, t' \in \mathcal{T}$ ,  $\exists r$  such that  $(t, t') \in S_r$  - every team meets every other team once. As notation,  $S(t, t') = r$  means that in basic schedule  $S$ , the match between  $t, t'$  is scheduled in round  $r$ . A schedule  $S$  is compatible with HAP-set  $\mathcal{H}$  if for every match  $(t, t') \in S_r$ ,  $H_r(t) \neq H_r(t')$ , for every round  $r$ .

Given a HAP-set  $\mathcal{H}$ , we denote  $\mathcal{S}(\mathcal{H}) = \{S : S \text{ compatible with } \mathcal{H}\}$  the set of all schedules compatible with  $\mathcal{H}$ . If  $\mathcal{S}(\mathcal{H}) \neq \emptyset$  we say the HAP-set is *feasible*.

**Definition 1** We define two schedules  $S, S'$  are **orthogonal**, if for every round  $r$  and every match  $(t, t') \in S_r$ ,  $(t, t') \notin S'_r$ . We denote this as  $S \perp S'$ .

**Definition 2** ([1]) The **width** of a HAP-set  $\mathcal{H}$  is the cardinality  $|\mathcal{S}|$  of the largest set of schedules  $\mathcal{S}$  compatible with  $\mathcal{H}$ , such that any two schedules  $S, S' \in \mathcal{S}$  are orthogonal.

If two schedules are orthogonal, every match is played in different rounds in both the schedules.

Another interesting property for a pair of schedules, is whether a pair of matches is never played together in the same round in both schedules. To capture this property we define match-pair disjointness.

**Definition 3** Two schedules  $S, S'$  are **match-pair disjoint**, if for every pair of matches  $(t, u), (t', u') \in S_r$ , there does not exist a  $r'$  such that  $(t, u), (t', u') \in S'_{r'}$ .

Thus, if two schedules are match-pair disjoint, it follows that for any pair of matches there is a schedule where they are not in the same round.

## 3 Our results

We state the following results regarding the width of HAP-sets and match pair disjoint schedules:

### Theorem 4

- Any  $\mathcal{H}$  on  $2n$  teams has  $\text{width}(\mathcal{H}) \leq n$ , with strict inequality if  $n$  is not a power of 2.
- When  $n = 2^\ell$ , there exists  $\mathcal{H}$  with  $\text{width}(\mathcal{H}) = n$ .

- There exists  $\mathcal{H}$  on  $2n$  teams with  $\text{width}(\mathcal{H}) = 2$  for every  $n \geq 2$ .
- When  $N = 8$ , there exists a  $\mathcal{H}$  compatible with two match pair disjoint schedules  $S, S'$ .

All results stem from constructive arguments. We close this abstract by giving two schedules for  $N=8$  that are match-pair disjoint and are compatible with the same HAP-set.

The two schedules  $S, S'$  are shown in Table 1. The matches per round are denoted as (Home,Away), and the HAP-set they are both compatible with, can be extracted from either of the two schedules and is given in Table 2.

Round	1	2	3	4	5	6	7
S	0-1	0-2	0-3	0-4	0-5	0-6	0-7
	2-3	3-1	2-1	1-5	4-1	1-7	6-1
	5-4	4-6	4-7	6-2	2-7	2-4	2-5
	6-7	5-7	5-6	3-7	3-6	5-3	3-4
S'	0-4	0-1	0-7	0-2	0-6	0-3	0-5
	2-7	3-6	2-3	1-7	4-5	2-6	6-7
	5-3	4-7	4-6	6-5	2-1	1-4	2-4
	6-2	5-2	5-1	3-4	3-7	5-7	3-1

Table 1: Two match-disjoint schedules

0	H	H	H	H	H	H	H
1	A	A	A	H	A	H	A
2	H	A	H	A	H	H	H
3	A	H	A	H	H	A	H
4	A	H	H	A	H	A	A
5	H	H	H	A	A	H	A
6	H	A	A	H	A	A	H
7	A	A	A	A	A	A	A

Table 2: The HAP-set compatible with both match-disjoint schedules

## References

- [1] ROEL LAMBERS, DRIES GOOSSENS, FRITS C.R. SPIEKMA (2022). *The Flexibility of Home Away Pattern Sets*. Journal of Scheduling, **Springer**

## Faster Matchings via Learned Duals

Michael Dinitz <sup>\*</sup>    Sungjin Im <sup>†</sup>    Thomas Lavastida <sup>‡</sup>  
Benjamin Moseley (Speaker) <sup>§</sup>    Sergei Vassilvitskii <sup>¶</sup>

---

An emerging line of research asks how algorithms can be augmented with machine-learned predictors to circumvent worst case lower bounds when the predictions are good, and approximately match them otherwise. Naturally, a rich area of applications of this paradigm has been in online algorithms, where the additional information revealed by the predictions reduces the uncertainty about the future and can lead to better choices, and thus better competitive ratios. For instance, see the work by [4].

However, the power of predictions is not limited to improving online algorithms. Indeed, the aim of the empirical paper that jump-started this area by [3] was to improve running times for basic indexing problems. The main goal and contribution of this work is to show that at least in one important setting (weighted bipartite matching), we can give formal justification for using machine learned predictions to improve running times: there are predictions which can provably be learned, and if these predictions are “good” then we have running times that outperform standard methods both in theory and empirically.

How can predictions help with running time? One intuitive approach, which has been used extensively in practice, is through the use of “warm-start” heuristics, where instead of starting with a blank slate, the algorithm begins with some starting state (which we call a warm-start “solution” or “seed”) which hopefully allows for faster completion. While it is a common technique, there is a dearth of analysis understanding what constitutes a good warm-start, when such initializations are helpful, and how they can best be leveraged.

Thus we have a natural goal: put warm-start heuristics on firm theoretical footing by interpreting the warm-start solution as learned predictions. In this set up we are given a number of instances of the problem (the training set), and we can use them to compute a warm-start solution that will (hopefully) allow us to more quickly compute the optimal solution on future, test-time, instances. There are three challenges that we must address:

---

<sup>\*</sup>mdinitz@cs.jhu.edu. Johns Hopkins University

<sup>†</sup>sim3@ucmerced.edu. UC Merced

<sup>‡</sup>tlavasti@andrew.cmu.edu. Carnegie Mellon University

<sup>§</sup>mosleeyb@andrew.cmu.edu. Carnegie Mellon University

<sup>¶</sup>sergeiv@google.com. Google

- (i) **Feasibility.** The learned prediction (warm-start solution) might not even be *feasible* for the specific instance we care about! For example, the learned solution may be matching an edge that does not exist in the graph at testing time.
- (ii) **Optimization.** If the warm-start solution is feasible and near-optimal then we want the algorithm to take advantage of it. In other words, we would like our running time to be a function of the quality of the learned solution.
- (iii) **Learnability.** It is easy to design predictions that are enormously helpful but which cannot actually be learned (e.g., the “prediction” is the optimal solution). We need to ensure that a typical solution learned from a few instances of the problem generalizes well to new examples, and thus offers potential speedups.

If we can overcome these three challenges, we will have an *end-to-end* framework for speeding up algorithms via learned predictions: use the solution to challenge (iii) to learn the predictions from historical data, use the solution to challenge (i) to quickly turn the prediction into something feasible for the particular problem instance while preserving near-optimality, and then use this as a warm-start seed in the solution to challenge (ii).

## Our Contributions

We focus on one of the fundamental primitives of combinatorial optimization: computing bipartite matchings. For the bipartite minimum-weight perfect matching (MWPM) problem, as well as its extension to  $b$ -matching, we show that the above three challenges can be solved.

A key conceptual question is finding a specification of the seed, and an algorithm to use it that satisfies the desiderata above. We have discussed warm-start “solutions”, so it is tempting to think that a good seed is a partial solution: a set of matched edges that can then be expanded to a optimal matching. After all, this is the structure we maintain in most classical matching algorithms. Moreover, any such solution is feasible (one can simply set non-existing edges to have very high weight), eschewing the need for the feasibility step. At the same time, as has been observed previously in the context of online matchings [1], this *primal* solution is brittle, and a minor modification in the instance (e.g. an addition of a single edge) can completely change the set of optimal edges.

Instead, following the work of [1], we look at the *dual* problem; that is, the dual to the natural linear program. We quantify the “quality” of a prediction  $\hat{y}$  by its  $\ell_1$ -distance from the true optimal dual  $y^*$ , i.e., by  $\|\hat{y} - y^*\|_1$ . The smaller quantities correspond to better predictions. Since the dual is a packing problem we must contend with feasibility: we give a simple linear time algorithm that converts the prediction  $\hat{y}$  into a feasible dual while increasing the  $\ell_1$  distance by a factor of at most 3.

Next, we run the Hungarian method starting with the resulting feasible dual. Here, we show that the running time is in proportional to the  $\ell_1$  distance of the feasible dual to the optimal dual. Finally, we show via a pseudo-dimension argument that not many samples are needed before the empirically optimal seed is a good approximation of the true optimum, and that this empirical optimum can be computed efficiently. For the learning argument, we assume that matching instances are drawn from a fixed but unknown distribution  $\mathcal{D}$ .

Putting it all together gives us our main result.

**Theorem 1 (Informal)** *There are three algorithms (feasibility, optimization, learning) with the following guarantees.*

- *Given a (possibly infeasible) dual  $\hat{y}$  from the learning algorithm, there exists an  $O(m+n)$  time algorithm that takes a problem instance  $c$ , and outputs a feasible dual  $\hat{y}'(c)$  such that  $\|\hat{y}'(c) - y^*(c)\|_1 \leq 3\|\hat{y} - y^*(c)\|_1$ .*
- *The optimization algorithm takes as input feasible dual  $\hat{y}'(c)$  and outputs a minimum weight perfect matching, and runs in time  $\tilde{O}(m\sqrt{n} \cdot \min\{\|\hat{y}'(c) - y^*(c)\|_1, \sqrt{n}\})$ .*
- *After  $\tilde{O}(C^2n^3)$  samples from an unknown distribution  $\mathcal{D}$  over problem instances, the learning algorithm produces duals  $\hat{y}$  so that  $\mathbb{E}_{c \sim \mathcal{D}} [\|\hat{y} - y^*(c)\|_1]$  is approximately minimum among all possible choices of  $\hat{y}$ , where  $C$  is the maximum edge cost and  $y^*(c)$  is an optimal dual for instance  $c$ .*

*Combining these gives a single algorithm that, with access to  $\tilde{O}(C^2n^3)$  problem instance samples from  $\mathcal{D}$ , has expected running time on future instances from  $\mathcal{D}$  of only  $\tilde{O}(m\sqrt{n} \min\{\alpha, \sqrt{n}\})$ , where  $\alpha = \min_y \mathbb{E}_{c \sim \mathcal{D}} [\|y - y^*(c)\|_1]$ .*

We emphasize that the Hungarian method with  $\tilde{O}(mn)$  running time is the standard algorithm in practice. Although there are other theoretically faster exact algorithms for bipartite minimum-weight perfect matching [2] that run in  $O(m\sqrt{n} \log(nC))$ , they are relatively complex (using various scaling techniques). Very recent breakthroughs give algorithms of run time  $\tilde{O}((m+n)^{1.5} \log^2(C))$  for the minimum-weight perfect matching problem and several interesting extensions [5]. However, the algorithms are highly complicated and their practical performance is yet to be demonstrated. In fact, we could not find any implementation of the above algorithms, except for the Hungarian method, of which multiple implementations are readily available.

Note that our result shows that we can speed up the Hungarian method as long as the  $\ell_1$ -norm error of the learned dual, i.e.,  $\|\hat{y} - y^*(c)\|_1$  is  $o(\sqrt{n})$ . Further, as the projection step that converts the learned dual into a feasible dual takes only linear time, the overhead of our method is essentially negligible. Therefore, even if the prediction is of poor quality, our method has worst-case running time that is *never* worse than that of the Hungarian algorithm. Even our learning algorithm is simple, consisting of a straightforward empirical risk minimization algorithm (the analysis is more complex and involves bounding the “pseudo-dimension” of the loss functions).

## References

- [1] Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings 10th ACM Conference on Electronic Commerce (EC-2009), Stanford, California, USA, July 6–10, 2009*, pages 71–78, 2009.
- [2] Ran Duan and Hsin - Hao Su. A scaling algorithm for maximum weight matching in bipartite graphs. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1413–1424. SIAM, 2012.
- [3] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504. ACM, 2018.
- [4] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3302–3311, 2018.
- [5] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020.

# The Power of Amortized Recourse on Online Graph Problems

Hsiang-Hsuan Liu \*      Jonathan Toole-Charignon (Speaker) †

---

## 1 Introduction

For a *graph sum problem*  $Q$ , given a graph  $G = (V, E)$ , an algorithm has to assign each element in  $\mathcal{X} = V \cup E$  a non-negative value such that the assignment satisfies certain properties associated with  $Q$ . Formally, the assignment is defined as  $\mathcal{A} : \mathcal{X} \rightarrow \mathbb{R}$  such that  $\mathcal{A}(\mathcal{X})$  satisfies a set of properties  $\mathcal{P}_Q$ . The *cost* of a feasible assignment  $\mathcal{A}$  is defined as a function  $cost(\mathcal{X}, \mathcal{A}(\mathcal{X})) = \sum_{x \in \mathcal{X}} \mathcal{A}(x)$ , which should be minimized or maximized as appropriate for the objective of the corresponding problem. In an *online model*, the elements in  $\mathcal{X}$  are revealed one-by-one rather than as a single complete input, and an *online algorithm* must make irrevocable decisions on each revealed element without knowledge of future inputs. In this work, the performance of an online algorithm is measured by the *competitive ratio*.

For many online graph problems\*, there is no  $O(1)$ -competitive algorithm in the standard model. This model is pessimistic, in that altering real-world decisions may be possible (albeit expensive). In this work, we study the model that improves online algorithms' performance by allowing *recourse* (that is, altering decisions). Specifically, we aim to find the trade-off between the performance improvement and the amount of amortized recourse, which is defined as the total number of altered decisions divided by the number of elements whose value can be non-zero.

Beyond the practical motivation of relaxing irrevocability of online algorithms' decisions, amortized recourse also provides insight on how a given online problem is affected by uncertainty. In particular, it captures how rapidly the structure of the offline optimal solution can change: the fewer elements required to do so, the larger the amortized recourse. Furthermore, the impact of uncertainty is directly correlated with this idea: the faster the optimal can change, the more impact uncertainty on future inputs will have. For example, to attain a constant competitive ratio, one needs exactly  $O(\log n)$  recourse per edge for min-cost bipartite matching [2, 8],

---

\*[h.h.liu@uu.nl](mailto:h.h.liu@uu.nl). Department of Information and computing sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands.

†[j.c.f.toole-charignon@uu.nl](mailto:j.c.f.toole-charignon@uu.nl). Department of Information and computing sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands.

\*Online graph problems have a lot of applications in decision-making problems such as scheduling, resource assignment, and network design problems.

while one only needs a constant amount of recourse per element for maximum independent set and minimum vertex cover [5].

There has been extensive work on online algorithms equipped with recourse for a variety of different problems. For amortized recourse, people have studied online bipartite matching [3], interval graph coloring [4], minimum spanning tree [9], traveling salesperson [9], Steiner tree [7], and online facility location [6]. Worst-case recourse has also been considered on a variety of graph problems [1, 5].

## 2 Our contribution

In this work, we define a family of general *monotone* graph problems and show that for every problem in this family and any  $t > 1$ , there is a  $t\alpha$ -competitive online algorithm,  $\text{TaS}_t$ , that incurs at most polynomial of  $t$  amortized recourse. The algorithm  $\text{TaS}_t$  uses an “incremental”  $\alpha$ -approximation algorithm as a yardstick and greedily assigns the newly-revealed element a feasible value if this assignment remains  $t\alpha$ -competitive. Otherwise,  $\text{TaS}_t$  *switches* its assignment to the yardstick’s one and incurs recourse. Notice that by the monotonicity of the problem, the optimal solution is incremental. Therefore,  $\text{TaS}_t$  can be  $t$ -competitive for any monotone problem with arbitrary  $t > 1$ . In addition, if we apply a polynomial-time approximation algorithm as reference, then  $\text{TaS}_t$  also runs in polynomial time.

We further refine this trade-off between competitive ratio and amortized recourse for three classical graph problems. For `INDEPENDENTSET`, we refine the analysis of our general algorithm and show that the actual amount of amortized recourse is smaller. For `MAXIMUMMATCHING`, we use an existing algorithm with limited greed to reduce the amount of amortized recourse. For `VERTEXCOVER`, we introduce a polynomial-time algorithm that further limits greed to show that constant amortized recourse is needed for achieving competitive ratio strictly smaller than 2 for any given instance. Our results are summarized in Table 1.

**Definition 1.** *In a monotone sum problem, for any two subsets of elements,  $S_1 \subseteq S_2 \subseteq \mathcal{X}$ ,  $\text{OPT}(S_1) \leq \text{OPT}(S_2)$ , where  $\text{OPT}(S)$  is the value of the optimal assignment for  $S^\dagger$ . Similarly, an algorithm  $\text{ALG}$  is incremental if for any two instances  $\mathcal{X}_1 \subseteq \mathcal{X}_2$ ,  $\text{ALG}(\mathcal{X}_1) \leq \text{ALG}(\mathcal{X}_2)$ .*

**Theorem 2.** *Using an incremental  $\alpha$ -approximation algorithm as the yardstick,  $\text{TaS}_t$  is  $(t \cdot \alpha)$ -competitive and incurs at most  $\frac{w_{\max} \cdot (t+1)}{\min\{1, w_{\min}\} \cdot (t-1)}$  amortized recourse for any monotone-sum graph problem where  $w_{\max}$  and  $w_{\min}$  are the maximum and minimum non-zero values that can be assigned to an element<sup>‡</sup>.*

---

<sup>†</sup>Independent set, vertex cover, and maximum matching are monotone sum problems. Dominating set and matching with delays are sum problems that are not monotone. Coloring is monotone, but not a sum problem.

<sup>‡</sup>For `INDEPENDENTSET`, `MAXIMUMMATCHING`, and `VERTEXCOVER` problems, the  $\text{TaS}_t$  algorithm attains competitive ratio  $t > 1$  while incurring at most  $\frac{t+1}{t-1}$  amortized recourse.

	(Competitive ratio, worst case recourse)	(Competitive ratio, amortized recourse)
Monotone sum problems		$(t\alpha, \frac{w_{\max} \cdot (t+1)}{\min\{w_{\min}, 1\} \cdot (t-1)})$
Maximum Independent Set	(2.598, 2) [5]	$(t, \frac{t}{t-1})$ (2.598, 1.626)
Maximum Matching	$(k, O(\frac{\log k}{k}) + 1)$ [1] (1.5, 2) [5]	$(t, \frac{(2-t^*)}{(t^*-1)(3-t^*)} + \frac{t^*-1}{3-t^*})$ ((1.5, 1) with $t^* = 1.5$ )
Minimum Vertex Cover	(2, 1) [5]	$(2 - \frac{2}{\text{OPT}}, \frac{10}{3})$

Table 1: Summary of our results. Note that  $t > 1$  can be any real number, and  $\alpha \geq 1$  is the incremental yardstick algorithm’s approximation ratio. For Maximum Matching,  $t^*$  is the largest number s.t.  $t^* \leq t$  and  $t^* = 1 + \frac{1}{j}$  for some integer  $j$ .

**IndependentSet.** For the maximum independent set problem in the vertex-arrival model, we show that the amortized recourse incurred by  $\text{TaS}_t$  is in fact smaller than the general bound  $\frac{t+1}{t-1}$  by a more sophisticated analysis.

**Lemma 3. (Instance reduction)** *For any instance  $(G, \sigma)$  of the maximum independent set problem, there exists an instance  $(G', \sigma')$  for which any newly revealed vertex is either accepted by  $\text{TaS}_t$  or is part of the optimal offline solution when  $\text{TaS}_t$  incurs its next switch, but not both, such that the amortized recourse for  $(G', \sigma')$  is at least that for  $(G, \sigma)$ .*

Using Lemma 3, we can bound above the amortized recourse incurred by  $\text{TaS}_t$  against any reduced instance, and thus against any instance.

**Theorem 4.** *For the maximum independent set problem, given a target competitive ratio  $t > 1$ ,  $\text{TaS}_t$  is  $t$ -competitive while incurring at most  $\frac{t}{t-1}$  amortized recourse. Furthermore, for any  $1 < t \leq 2$ ,  $\varepsilon > 0$ , and  $t$ -competitive deterministic online algorithm, there exists an instance for which the algorithm incurs at least  $\frac{1}{t-1} - \varepsilon$  amortized recourse.*

**MaximumMatching.** The  $\text{TaS}_t$  algorithm is greedy in a two-fold manner. First, the algorithm assigns the value greedily once an element arrives. Second, the algorithm switches its solution to the optimal one completely when the current solution fails to be  $t$ -competitive. The complete switching costs a lot of recourse. However, it may be possible to reduce the amount of amortized recourse while maintaining  $t$ -competitiveness by switching the solution *partially* into the optimal one. For the maximum matching problem, we show that the **L-Greedy** algorithm by Angelopoulos et al. [1] incurs less amortized recourse.

**Theorem 5.** *For the maximum cardinality matching problem in the edge-arrival model, we consider algorithms with target competitive ratio  $1 < t < 2^{\S}$ . The*

<sup>§</sup>Since it was shown that to achieve 1.5-competitiveness, every vertex incurs at most 2 recourse.

*L-Greedy* algorithm is  $t$ -competitive and incurs at most  $\frac{(2-t^*)}{(t^*-1)(3-t^*)} + \frac{t^*-1}{3-t^*}$  amortized recourse<sup>¶</sup>. Furthermore, no deterministic  $t$ -competitive online algorithm can incur amortized recourse less than  $\frac{(2-t^*)}{(t^*-1)(3-t^*)}$  in the worst case.

**VertexCover.** For VERTEXCOVER, we introduce a polynomial-time algorithm Duo-Halve that further limits greed to show that  $(2 - \frac{2}{OPT})$ -competitiveness, where OPT is the optimal vertex cover size, can be achieved with at most  $\frac{10}{3}$  amortized recourse by a potential function argument.

**Duo-Halve Algorithm (DH).** During the process, the algorithm maintains a maximal matching  $M(\mathcal{X})$  on the current input graph  $\mathcal{X}$  to construct a solution DH( $\mathcal{X}$ ). The algorithm only selects vertices that are saturated by the matching and rejects as many vertices as possible from the two latest matched edges.

**Theorem 6.** *The DH algorithm is  $(2 - \frac{2}{OPT})$ -competitive and incurs at most  $\frac{10}{3}$  amortized recourse, where OPT is the minimum vertex cover size. Furthermore, DH runs in  $O(n^3)$  time, where  $n$  is the number of vertices in the final graph.*

## References

- [1] S. ANGELOPOULOS, C. DÜRR, S. JIN. *Online Maximum Matching with Recourse*. J. Comb. Optim. 40(4): 974-1007 (2020)
- [2] A. ANTONIADIS, C. FISCHER, A. TÖNNIS *A collection of lower bounds for online matching on the line*. LATIN 2018: 52–65
- [3] A. BERNSTEIN, J. HOLM, E. ROTENBERG. *Online Bipartite Matching with Amortized  $O(\log^2 n)$  Replacements*. J. ACM 66(5): 37:1-37:23 (2019)
- [4] B. BOSEK, Y. DISSER, A. E. FELDMANN, J. PAWLEWICZ, A. ZYCH-PAWLEWICZ. *Recoloring Interval Graphs with Limited Recourse Budget*. SWAT 2020: 17:1-17:23
- [5] J. BOYAR, L. M. FAVRHOLDT, M. KOTRBCÍK, K. S. LARSEN. *Relaxing the Irrevocability Requirement for Online Graph Algorithms*. Algorithmica (2022)
- [6] M. CYGAN, A. CZUMAJ, M. MUCHA, P. SANKOWSKI. *Online Facility Location with Deletions*. ESA 2018: 21:1-21:15
- [7] A. GU, A. GUPTA, A. KUMAR. *The Power of Deferral: Maintaining a Constant-Competitive Steiner Tree Online*. SIAM J. Comput. 45(1): 1-28 (2016)
- [8] N. MEGOW, L. NÖLKE. *Online Minimum Cost Matching with Recourse on the Line*. APPROX-RANDOM 2020: 37:1-37:16
- [9] N. MEGOW, M. SKUTELLA, J. VERSCHAE, A. WIESE. *The Power of Recourse for Online MST and TSP*. SIAM J. Comput. 45(3): 859-880 (2016)

---

<sup>¶</sup> $t^*$  is the largest number such that  $t^* \leq t$  and  $t^* = 1 + \frac{1}{j}$  for some integer  $j$ .

# On Hop-Constrained Steiner Trees in Tree-Like Metrics\*

Martin Böhm<sup>†</sup>    Ruben Hoeksma (Speaker)<sup>‡</sup>    Nicole Megow<sup>§</sup>  
Lukas Nölke<sup>§</sup>    Bertrand Simon<sup>¶</sup>

---

## 1 Introduction

The *minimum-cost Steiner tree* problem is a fundamental network design problem: Given a set of terminals in a finite metric space, the task is to find a tree that spans all terminals and has minimum overall cost. While present in the original list of Karp’s 21 NP-complete problems [7], research on this problem goes back as far as the 1930s [6], and its origins can be traced back even further, as it is named after the 19th century Swiss mathematician Jakob Steiner. When the set of terminals to be interconnected is equal to the ground set of the metric space, we speak of the equally fundamental *minimum-cost spanning tree problem*, which is solvable in polynomial time. Both problems have had major applications in the 20th and 21st century, particularly in the design of transportation and communication networks.

When considering the efficiency and reliability of a network, it is a common and natural requirement that vertices are not just connected, but rather connected with a path that consists of only few edges. In the literature on network design, this requirement is known as *bounded hop distance*, where *hop* refers to an edge and *hop distance* to the number of edges on a path. A restriction on hop distances aims at reducing transmission delays and packet loss, avoiding the flooding of a network when routing, and increasing reliability of networks by limiting the amplifying effect of link failures.

Adding hop constraints makes network design problems substantially harder. The minimum-cost spanning tree problem, for example, is well-known to be polynomial-time solvable, whereas its hop-constrained variants do admit constant lower bounds on the approximation ratio [4, 1] in certain metrics. Network design problems without hop constraints often become easier when the underlying

---

\*An earlier version of this paper was published in the proceedings of MFCS 2020 [2].

<sup>†</sup>`boehm@cs.uni.wroc.pl`. Institute of Computer Science, University of Wrocław, Poland.

<sup>‡</sup>`r.p.hoeksma@utwente.nl`. Department of Applied Mathematics, University of Twente, Enschede, The Netherlands.

<sup>§</sup>`nicole.megow@uni-bremen.de`, `noelke@uni-bremen.de`. Department of Mathematics and Computer Science, University of Bremen, Germany.

<sup>¶</sup>`bertrand.simon@cc.in2p3.fr`. IN2P3 Computing Center, CNRS, Villeurbanne, France.

metric can be represented as a tree or is somewhat close to a tree, such as graphs with bounded treewidth. In this paper, we investigate whether in presence of hop constraints this is still true.

Formally, we define the problem as follows. We are given a finite metric space  $(V, d)$  with a set  $V$  of  $n$  points as well as a distance function  $d : V \times V \rightarrow \mathbb{Q}_+$ , a set of terminals  $\mathcal{X} \subseteq V$ , a root  $r \in \mathcal{X}$ , and an integer  $k \geq 1$ . A  $k$ -hop Steiner tree is said to be a tree  $\check{S} = (V_{\check{S}}, E_{\check{S}})$  rooted at  $r$  that spans all points in  $\mathcal{X}$  and has a depth of at most  $k$ . That is,  $\mathcal{X} \subseteq V_{\check{S}} \subseteq V$  and for  $v \in V_{\check{S}}$ , the number of edges in the  $r$ - $v$  path in  $\check{S}$  is at most  $k$ . The *cost* of a Steiner tree refers to the sum of its edge costs,  $\sum_{\{u,v\} \in E_{\check{S}}} d(u, v)$ , with edge costs given by  $d$ . We consider the *minimum-cost  $k$ -hop Steiner tree problem* ( $k$ -hop MŠT problem<sup>1</sup>) that asks for a  $k$ -hop Steiner tree of minimum cost.

Special cases of the  $k$ -hop MŠT problem include the minimum-cost  $k$ -hop spanning tree ( $k$ -hop MST) problem (when  $\mathcal{X} = V$ ), the Steiner Tree problem (when  $k \geq n$ ) and the Uncapacitated Facility Location problem (when  $k = 2$ ).

## 2 Our contribution

In this paper, we consider the  $k$ -hop MŠT problem in tree-like metrics. That is, we consider metrics which are represented by graphs from tree-like graph classes using the natural correspondence between metric spaces and weighted complete graphs via the shortest path metric. We say that a weighted graph  $G = (V, E)$  with a weight function  $d : E \rightarrow \mathbb{R}^+$  *induces* a metric  $(V, d)$ , if for any two vertices  $u, v \in V$  the length of the shortest  $u$ - $v$  path in  $G$  is equal to  $d(u, v)$ . We call a metric a *tree* (resp. *path*) *metric* if there is a tree (resp. path) that induces it, and we call it a *metric with treewidth  $\omega$*  if it is induced by a graph with treewidth  $\omega$ .

We develop dynamic programming algorithms for path metrics, tree metrics and metrics with treewidth  $\omega$  to show the following results.

**Theorem 1** *On path metrics,  $k$ -hop MŠT can be solved exactly in time  $O(kn^5)$ .*

**Theorem 2** *On tree metrics,  $k$ -hop MŠT can be solved exactly in time  $n^{O(k)}$ .*

**Theorem 3** *On metrics with treewidth  $\omega$ ,  $k$ -hop MŠT can be solved exactly in time  $n^{O(\omega k)}$ .*

We further extend the dynamic programming algorithm of Theorem 3 to graphs with bounded highway dimension [3] and graphs with bounded doubling dimension [5]. In both cases we only achieve a  $(1 + \epsilon)$ -approximate solution and we need to relax the hop constraint by allowing one more hop than the optimal solution.

---

<sup>1</sup>For brevity and as an homage to the work of Jarník and Kössler [6, 8], we use the Czech letter Š to distinguish Steiner trees from spanning trees in MŠT resp. MST. The pronunciation of Š is ⟨sh⟩, the same as the German pronunciation of the letter S in Steiner.

**Theorem 4** For a metric induced by a graph with bounded highway dimension (resp. bounded doubling dimension) and a constant  $k$ , let  $OPT_k$  be the cost of a  $k$ -hop MST. A  $(k+1)$ -hop Steiner tree of cost at most  $(1+\varepsilon)OPT_k$ , for  $\varepsilon > 0$ , can be computed in quasi-polynomial time.

Finally, supplementing our positive results, we show that in general metrics, we cannot hope for better than  $\log n$ -approximations in polynomial time, even when we relax the hop-constraint.

**Theorem 5** For any two constants  $c$  and  $\ell$ , and given a weighted graph  $G$  as input, it is NP-hard to find a  $(k+\ell)$ -hop Steiner tree in  $G$  of cost  $(1-c) \cdot \log n \cdot OPT$ , where  $OPT$  is the minimal cost of a  $k$ -hop Steiner tree in  $G$ . This remains true even when all edges in the graph  $G$  have weight equal to 1.

## References

- [1] L. ALFANDARI AND V. T. PASCHOS, *Approximating minimum spanning tree of depth 2*, International Transactions in Operational Research, 6 (1999), pp. 607–622.
- [2] M. BÖHM, R. HOEKSMAN, N. MEGOW, L. NÖLKE, AND B. SIMON, *Computing a Minimum-Cost  $k$ -Hop Steiner Tree in Tree-Like Metrics*, in MFCS, vol. 170 of LIPIcs, 2020, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 18:1–18:15.
- [3] A. E. FELDMANN, W. S. FUNG, J. KÖNEMANN, AND I. POST, *A  $(1+\varepsilon)$ -embedding of low highway dimension graphs into bounded treewidth graphs*, SIAM Journal on Computing, 47 (2018), pp. 1667–1704.
- [4] S. GUHA AND S. KHULLER, *Greedy strikes back: Improved facility location algorithms*, Journal of Algorithms, 31 (1999), pp. 228 – 248.
- [5] A. GUPTA, R. KRAUTHGAMER, AND J. R. LEE, *Bounded geometries, fractals, and low-distortion embeddings*, in FOCS, IEEE Computer Society, 2003, pp. 534–543.
- [6] V. JARNÍK AND M. KÖSSLER, *O minimálních grafech, obsahujících  $n$  daných bodů*, Časopis pro pěstování matematiky a fyziky, 63 (1934), pp. 223–235.
- [7] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, The IBM Research Symposia Series, Plenum Press, New York, 1972, pp. 85–103.
- [8] B. KORTE AND J. NEŠETRIL, *Vojtěch Jarník’s work in combinatorial optimization*, Discrete Mathematics, 235 (2001), pp. 1–17.

# On the Extended TSP Problem\*

Julián Mestre (Speaker) <sup>†</sup>      Sergey Pupyrev <sup>‡</sup>

Seeun William Umboh <sup>§</sup>

---

## 1 Introduction

Profile-guided binary optimization (PGO) is an effective technique in modern compilers to improve performance by optimizing how binary code is laid out in memory. At a very high level, the idea is to collect information about typical executions of an application and then use this information to re-order how code blocks are laid out in the binary to minimize instruction cache misses, which in turn translates into running time performance gains. Newell and Pupyrev [3] recently introduced an optimization problem, which they call the Extended TSP (EXT-TSP) problem that aims at maximizing the number of block transitions that do not incur a cache miss.

The input to the EXT-TSP problem is a weighted directed graph  $G = (V, E)$ , which in the context of PGO corresponds to the control flow representation of the code we are trying to optimize: Every node  $u \in V$  corresponds to a basic block of code (for the purposes of this paper we can think of each of these blocks as a single instruction that takes a fixed amount of memory to encode); every edge  $(u, v) \in E$  represents the possibility of an execution jumping from  $u$  to  $v$ , and the weight  $w(u, v)$  captures how many times the profiler recorded said jump during the data collection phase. Our ultimate goal is to find a linear ordering of the nodes, each of which represents a possible code layout of the binary; we let this linear ordering be encoded by a one-to-one function  $d : V \rightarrow \{1, \dots, |V|\}$ . Finally, each edge  $(u, v)$  contributes  $f(|d_u - d_v|) \cdot w(u, v)$  to the objective, where  $|d_u - d_v|$  is the distance between the edge endpoints in the linear ordering, and  $f(\cdot)$  is a non-increasing discount function such that  $f(1) = 1$  and  $f(i) = 0$  for  $i > k$ , where  $k = O(1)$  is part of the problem definition.

Newell and Pupyrev [3] designed and evaluated heuristics for EXT-TSP leading to significantly faster binaries. Their implementation is available in the open

---

\*An extended version of this paper was published in the Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC 2021) [2].

<sup>†</sup>[julian.mestre@sydney.edu.au](mailto:julian.mestre@sydney.edu.au). Meta Inc and University of Sydney.

<sup>‡</sup>[spupyrev@fb.com](mailto:spupyrev@fb.com). Meta Inc.

<sup>§</sup>[william.umbogh@sydney.edu.au](mailto:william.umbogh@sydney.edu.au) University of Sydney.

source project Binary Optimization and Layout Tool (BOLT) [1, 4, 3]. In their experiments, they found that setting  $k$  to be a small constant<sup>1</sup> and

$$f(|d_u - d_v|) = \begin{cases} 1 - \frac{|d_u - d_v|}{k} & \text{for } 1 < |d_u - d_v| < k, \\ 0 & \text{otherwise} \end{cases}$$

yields the best results. The high level intuition is that the discount factor is a proxy for the probability that taking the jump causes a cache miss. Thus, the EXT-TSP objective aims at maximizing the number of jumps that do not cause a cache miss.

In this paper we initiate the theoretical study of EXT-TSP by providing a variety of hardness and algorithmic results for solving the problem both in the approximate and the exact sense in both general and restricted graph classes.

## 2 Our results

We show that EXT-TSP is APX-hard to approximate in general. We give a polynomial time  $(k+1)$ -approximation algorithm and a  $n^{O(\frac{k}{\epsilon})}$  time  $(2+\epsilon)$ -approximation for general graphs. We also give a  $n^{O(\frac{k}{\epsilon})}$  time  $(1+\epsilon)$ -approximation for some sparse graphs classes such as planar or treewidth-bounded graphs.

Interestingly, the problem remains challenging even on very simple graph classes; indeed, there is no exact  $n^{o(k)}$  time algorithm for trees unless the ETH fail. Finally, we complement this negative result with an exact  $n^{O(k)}$  time algorithm for trees.

## 3 Open problems

We regard our work as the first step in the theoretical study of EXT-TSP. As such, there are some interesting questions that remain unanswered:

1. Is there a polynomial-time  $O(1)$ -approximation, independent of  $k$ ?
2. Is there an exact  $\mathcal{O}(f(k, t)n^{O(k)})$  time algorithm where  $t$  is the treewidth of the instance?

## References

- [1] Facebook Inc. Binary optimization and layout tool, 2020. URL: <https://github.com/facebookincubator/BOLT>.
- [2] Julián Mestre, Sergey Pupyrev, and Seeun William Umboh. On the extended TSP problem. In *Prof. of the 32nd International Symposium on Algorithms and Computation*, volume 212 of *LIPICs*, pages 42:1–42:14, 2021.

---

<sup>1</sup>To be more specific,  $k$  is the number of blocks that can fit into 1024 bytes of memory.

- [3] Andy Newell and Sergey Pupyrev. Improved basic block reordering. *IEEE Transactions in Computers*, 69(12):1784–1794, 2020.
- [4] Maksim Panchenko, Rafael Auler, Bill Nell, and Guilherme Ottoni. Bolt: A practical binary optimizer for data centers and beyond. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, page 2–14, 2019.

# Arc-retraversing Solutions to the Traveling Salesman Problem with Drones

Nicola Morandi (Speaker) \*    Roel Leus †    Hande Yaman ‡

---

## 1 Introduction

Since the pioneering work by Murray and Chu [4] in 2015, the scientific literature about applications of drones to routing and parcel delivery has grown at a remarkable pace. At the time of writing, searching the words “truck drone routing” by Google Scholar produced 8 620 results, 40% of which from just 2020. An impressive number of surveys on the topic already appeared, like, among other interesting ones, those by Macrina et al. [3] and Otto et al. [6]. The interest in drones’ applications also comes from institutions and companies. For example, the European Commission forecasts more than 100 000 people employed and an economic impact of over 10 billion Euros per year in the European drone sector by 2035 [2]. At the same time, the e-commerce multinational Amazon obtained the approval from the relevant US authority for its Prime Air service “beyond visual line of sight” [5].

In this presentation, we examine a specific optimization problem named Traveling Salesman Problem with Drones (TSP-mD), in which a single truck and multiple drones cooperate to visit all the customers in a given network in the minimum amount of time. In the TSP-mD, the drones are allowed to serve multiple customers per sortie (i.e., drone flight), and to return to the launching location itself for landing. The truck can revisit customers, and traverse any arc multiple times; in the latter case we refer to the resulting solution as *arc-retraversing*. Although their practical implementation does not require any further technology, we observe that no previous studies in the related literature included arc-retraversing solutions in their problem definitions. On the one hand, we prove that it is instead necessary to include these solutions in the solution space of the TSP-mD, by showing instances for which all the optimal solutions are arc-retraversing. On the other hand, we provide conditions under which at least one optimal solution is not arc-retraversing. Finally, we establish bounds on the completion time in a number of different problem settings.

---

\*[nicola.morandi@kuleuven.be](mailto:nicola.morandi@kuleuven.be). Research Centre for Operations Research and Statistics (ORSTAT), KU Leuven, Naamsestraat 69, 3000 Leuven, Belgium.

†[roel.leus@kuleuven.be](mailto:roel.leus@kuleuven.be). ORSTAT, KU Leuven, Naamsestraat 69, 3000 Leuven, Belgium.

‡[hande.yaman@kuleuven.be](mailto:hande.yaman@kuleuven.be). ORSTAT, KU Leuven, Naamsestraat 69, 3000 Leuven, Belgium.

## 2 Problem statement

The TSP-mD can be defined as follows. Let  $N$  be the set of nodes including the customer locations and the unique depot (denoted by 0), and  $A$  be the set of arcs  $(i, j)$  for any distinct nodes  $i, j \in N$ . We denote the length of an arc  $a \in A$  by  $\ell_a$ . The resulting graph  $G = (N, A)$  is complete and directed. A non-capacitated truck and multiple identical drones cooperate to serve all the customers in  $N$ ; the truck and the drones do not necessarily travel at the same speed, but the ratio of their speeds is constant. We denote by  $N^{dr} \subseteq N$  and  $N^{tr} \subseteq N$  the subsets of the locations that can be served by the drones and by the truck, respectively. The depot 0 belongs to  $N^{tr} \subseteq N$ . Notice that if a node  $i \in N \setminus N^{tr}$ , then the truck cannot traverse any of the arcs that are incident to  $i$ .

The truck travels along a route that starts and ends at 0, and serves all the nodes contained therein. The drones can be independently launched onto sorties, and must be retrieved by the truck along its route. A sortie can contain multiple customers. The energy consumed by any sortie must not exceed the maximum amount allowed by the battery, and every time a drone lands, its battery is swapped with a fully-charged one in a negligible amount of time. For this reason, the total length of the arcs traversed by a drone during a sortie must not be higher than a given value  $B > 0$ .

In our problem setting, both the truck and the drones are allowed to wait for each other at customer locations. The whole operation is complete when all the customers are served, and both the truck and the drones have returned to the depot. We minimize the completion time.

As in Agatz et al. [1] and closely-related variants, we allow the truck to visit customers multiple times. Furthermore, we allow the truck to traverse arcs multiple times, and refer to the corresponding solutions as *arc-retraversing*. Notice that the truck can traverse an edge  $\{i, j\}$  twice in the two opposite senses without retraversing the same arc  $(i, j)$ , because each sense of traversal corresponds to a different arc between  $(i, j)$  and  $(j, i)$ . Finally, we prove that there exist instances where all the optimal solutions are arc-retraversing, and consequently that it is necessary to incorporate them in our problem statement.

## 3 Main results

In this talk, we prove the necessity of arc-retraversing solutions; the corresponding result is stated as follows.

**Proposition 1** *There exists an instance such that  $N = N^{dr} = N^{tr}$  and all the optimal solutions are arc-retraversing.*

At the same time, under two different sets of conditions, we prove that it is sufficient to consider solutions that are not arc-retraversing. The corresponding results are stated as follows.

**Proposition 2** *At least one optimal solution is not arc-retraversing if only a single drone is available and sorties contain at most one customer.*

**Proposition 3** *At least one optimal solution is not arc-retraversing if only a single drone is available,  $N^{tr} = N$ , and the truck and the drones travel at the same speed.*

As a complement to the proofs of the aforementioned propositions, we also develop a MIP model capable of solving middle-sized instances with a time limit of two hours. Finally, we establish bounds on the objective function in a number of different problem settings.

## References

- [1] Niels Agatz, Paul Bouman, and Marie Schmidt. Optimization Approaches for the Traveling Salesman Problem with Drone. *Transportation Science*, 52(4):965–981, 2018.
- [2] European Council. Drones: reform of EU aviation safety. <https://www.consilium.europa.eu/en/policies/drones/>, 2019. Last accessed: 05-October-2021.
- [3] Giusy Macrina, Luigi Di Puglia Pugliese, Francesca Guerriero, and Gilbert Laporte. Drone-aided routing: A literature review. *Transportation Research Part C: Emerging Technologies*, 120:102762, 2020.
- [4] Chase C Murray and Amanda G Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.
- [5] CNBC news. Amazon wins FAA approval for Prime Air drone delivery fleet. <https://www.cnbc.com/2020/08/31/amazon-prime-now-drone-delivery-fleet-gets-faa-approval.html>, 2020. Last accessed: 05-October-2021.
- [6] Alena Otto, Niels Agatz, James Campbell, Bruce Golden, and Erwin Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, 72(4):411–458, 2018.

# Solving a real-life assembly problem with lobster precedence and workforce constraint

Mauro Dell’Amico \*

Dario Bezzi (Speaker) †

---

## 1 Introduction

We consider a real-life production problem of off-road vehicles, which involves both machines and workforce constraints. We present a mathematical model and compare its performance with some heuristics.

The production of the vehicles is an assembly process of several components on a basic chassis. The shop floor is organized in workcenters, and each workcenter may consist of a few identical machines. Every machine in a workcenter is functionally identical to every other machine in the same workcenter. For uniformity with the literature we speak of “machines”, but in some workcenters, the actual machine is a specific tool or even a predetermined section of the floor where the worker must perform the *assembly operation*. Every operation is assigned to a given workcenter, but the choice of the machine to be used for that operation is left to the scheduler. Each machine can execute at most one operation at a time, and preemption is not allowed.

The assembly of a specific model of the vehicle is made by moving the chassis to some workcenters, accordingly to a model dependent sequence. In each workcenter it is performed a specific assembly operation. Each of these operations require a few components to be mounted on the chassis, and each component is build by means of a single operation to be executed on a dedicated workcenter (different from the workcenters used for the assembly operations). Let us call *first level component* the elements that must be mounted on the chassis, and, when no ambiguity arise, the corresponding operation. These components are usually made using some *second level component*, which are produced in other workcenters.

Each operation (assembly operation or first/second level component) requires one human worker from a finite worker’s set with cardinality smaller than the number of available machines. This constraint turns our production scheduling into a dual-resource constrained problem: machines and workers.

---

\*[mauro.dellamico@unimore.it](mailto:mauro.dellamico@unimore.it). Dipartimento di Scienze e Metodi dell’Ingegneria, Università degli Studi di Modena e Reggio Emilia.

†[dario.bezzi@unimore.it](mailto:dario.bezzi@unimore.it). Dipartimento di Scienze e Metodi dell’Ingegneria, Università degli Studi di Modena e Reggio Emilia.

The problem is a generalization of the Flexible Job Shop Problem with dual Resource Constraints and partial path flexibility. In the job shop each job consists of a sequence of operations, while in our case we have to deal with the production of the first and second level components. We address the interested reader to two recent surveys [1] and [5]. The problem also generalizes the scheduling of tasks with caterpillar precedence graph on identical machines (see [3]), but in our case we have lobster graphs instead of caterpillars (see below) and the operations must be executed on a subset of the machines. The problem has also relations with the project scheduling problem with resource constraints (see, e.g., [2]).

## 2 Formal description

Let  $W$  denote the set of workcenters, and let  $M^w$  be the set of identical machines available in workcenter  $w \in W$ . Let  $M = \bigcup_{w \in W} M^w$  denote the set of all machines and  $T$  the planning horizon. As usual in scheduling, we can represent the production constraints through an acyclic digraph  $G = (V, A)$ , where each vertex  $v \in V$  corresponds to an operation. Each arc  $(i, j)$  in the arc set  $A$  represents a precedence constraint: operation  $i$  must be finished before operation  $j$  can start. Each operation  $v \in V$  has an associated processing time  $p_v$ , and must be executed on exactly one machine in a given workcenter  $w(v) \in W$ . Preemption is not allowed and each machine can handle one operation at a time. Finally, it is given a set of  $R^{max}$  human resources with same skills, and each operation can be executed only if are contemporarily available a machine and a human resource. All operations are available at time 0, and must be scheduled in time interval  $T$ . The objective function requires to minimize the makespan.

This formulation holds for any digraph  $G$ , but the actual precedence constraints of the manufacturing problem instances form a very specific kind of digraph, called a *lobster* (a tree having the property that the removal of leaf nodes leaves a caterpillar graph).

### 2.1 Mathematical Model

Our model resembles the time-indexed model proposed in [4] for the project scheduling problem with resource constraint.

In order to simplify the writing we need to introduce, for each  $w \in W$ , the set  $V_w \subset V$  of the operations associated to workcenter  $w$  (i.e., such that  $w(v) = w \forall v \in V_w$ ). We will use binary variables  $x$  where

$$x_{vt} = \begin{cases} 1 & \text{if operation } v \text{ starts at time } t \\ 0 & \text{otherwise} \end{cases} \quad v \in V, t \in T$$

$$\min \max_{v \in V} \left( \sum_{t \in T} tx_{vt} + p_v \right) \quad (1)$$

$$\text{s.t. } \sum_{t \in T} x_{vt} = 1 \quad \forall v \in V \quad (2)$$

$$\sum_{v \in V_w} \sum_{h=t-p_v+1}^t x_{vh} \leq |M^w| \quad \forall t \in T, w \in W \quad (3)$$

$$\sum_{t \in T} tx_{it} + p_i \leq \sum_{t \in T} tx_{jt} \quad \forall (i, j) \in A \quad (4)$$

$$\sum_{v \in V} \sum_{h=t-p_v+1}^t x_{vh} \leq R^{max} \quad \forall t \in T \quad (5)$$

$$x_{vt} \in \{0, 1\} \quad \forall v \in V, t \in T \quad (6)$$

We minimize the makespan (1). Constraints (2) impose that each operation is executed once, constraints (3) ensure that no more than  $|M^w|$  machines are used in parallel in each workcenter  $w \in W$ . Constraints (4) impose the precedence constraints for each arc  $(i, j) \in A$ . Finally, constraint (5) impose that no more than  $R^{max}$  human resources can be used in parallel.

### 3 Real-life problem and solution approach

The real-life problem consists of scheduling the production of 20-30 vehicles on a set of 48 machines. Each operation can take several hours, so the entire production requires a couple of weeks to be terminated. The shop floor is divided in  $k+1$  areas. Each of the first  $k$  areas contains two workcenters: one dedicated to the assembly operations and one to the first level operations which produce components for the assembly workcenter of the same area. The last area is dedicated to produce the second level components for all the first level operations. In our case we have  $k = 6$ , four machines in each assembly workcenter and up to six in each of the workcenters dedicated to the first level operations. In the last area there are up to ten machines.

Each model of vehicle is described by a subgraph containing a main path describing the assembly operations on the  $k$  assembly workstation, plus some 2-arc paths describing the first and second level operations (see Figure 1 for a pictorial representation).

We implemented some list scheduler approach and Local Search improvement methods and we will present comparison of the solutions obtained with the mathematical model and with manual solutions by the company.

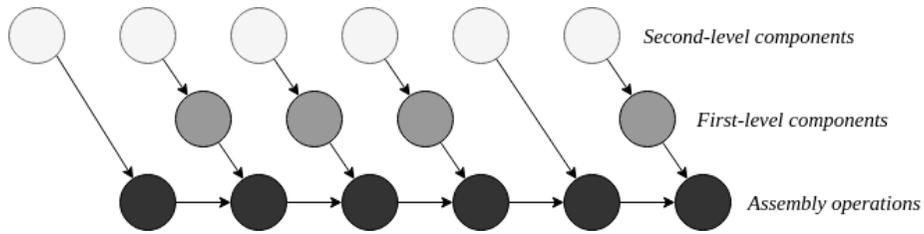


Figure 1: Example of the precedence graph for a single model (lobster graph)

## References

- [1] M. DHIFLAOUIA AND H.E. NOURI AND O.B. DRISSB (2018). *Dual-Resource Constraints in Classical and Flexible Job Shop Problems: A State-of-the-Art Review*, *Procedia Computer Science* 126, 1507–1515.
- [2] S. HARTMANN AND D. BRISKORN, *An updated survey of variants and extensions of the resource-constrained project scheduling problem*, *European Journal of Operational Research*, 297, 1-14.
- [3] G. NICOSIA AND A. PACIFICI (2013). *Scheduling tasks with comb precedence constraints on dedicated machines*, 7th IFAC Conference on Manufacturing Modelling, Management.
- [4] A.A.B. PRITSKER AND L.J. WATTERS AND P.M. WOLFE (1969), *Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach*, *Management Science*, 1,93-108
- [5] H. XIONG AND S. SHI AND D. REN AND J. HU (2022). *A survey of job shop scheduling problem: the types and models*, *Computers & Operations Research* 142, 105731.

# A 12/7-approximation algorithm for the discrete Bamboo Garden Trimming problem

Martijn van Ee \*

---

## 1 Introduction

In this abstract, we discuss a recent paper [4] on the discrete Bamboo Garden Trimming problem (BGT). In this problem, we are given a garden with  $n$  bamboos, where bamboo  $i$  grows with a rate of  $h_i$  per day. We assume w.l.o.g. that  $h_1 \geq h_2 \geq \dots \geq h_n$ . Initially, all bamboos have height zero. At the end of each day, one can cut down one bamboo to height zero. The goal in BGT is to make a perpetual schedule of cuts (trimming schedule) such that the height of the tallest bamboo ever is minimized. BGT has, among others, applications in scheduling maintenance of machines. Gasieniec et al. [6] gave a 2-approximation for BGT. Recently, Della Croce [3] improved this to a guarantee of approximately 1.888. In this paper, we will extend the approach of [6] in order to obtain a 12/7-approximation algorithm. This result is obtained by reducing BGT to the Pinwheel Scheduling problem (PS).

In the Pinwheel Scheduling problem [7], we are given  $n$  jobs with integral periods  $p_1 \leq \dots \leq p_n$ . On each day, we can schedule at most one job. The goal is to design a perpetual schedule such that each job  $i$  is scheduled at least once in any period of  $p_i$  consecutive days, or to conclude that no such schedule exists. Holte et al. [7] showed that the problem is contained in PSPACE, but no completeness result is known. An important issue of PS, and other related problems like BGT, is the representation of a solution. Explicitly writing down the schedule may take exponential time and space. In this paper, we only consider periodic schedules that can be represented by numbers  $o_i$  and  $t_i$  for each  $i$ . In a periodic schedule, job  $i$  is scheduled for the first time on day  $o_i$ , and then scheduled every  $t_i$  days later. Our algorithm will be able to compute the  $o_i$ 's and  $t_i$ 's in polynomial time.

An interesting aspect of a PS-instance is the density. The density of instance  $A = \{p_1, \dots, p_n\}$  is defined as  $\rho(A) = \sum_i 1/p_i$ . Clearly, if  $\rho(A) > 1$ , instance  $A$  is not schedulable. Holte et al. [7] showed that if  $p_i$  divides  $p_j$  whenever  $i < j$ , and  $\rho(A) \leq 1$ , then  $A$  is schedulable. Consequently, if  $\rho(A) \leq 1/2$ ,  $A$  can be scheduled by rounding each period down to its nearest power of 2. Later, more elaborate ways of rounding the periods were considered [1, 2, 5]. In the first paper

---

\*M.v.Ee.01@mindex.nl. Faculty of Military Sciences, Netherlands Defence Academy, Het Nieuwe Diep 8, 1781 AC Den Helder, The Netherlands

by Chan and Chin [2], they designed an algorithm that was able to round any instance  $A$  with  $\rho(A) \leq 2/3$  to a schedulable instance. They also considered a simpler algorithm such that any instance with density at most  $7/12$  is rounded to a schedulable instance. We will use this algorithm for our main result. They also observed that instance  $A = \{2, 3, M\}$  cannot be scheduled for any value of  $M$ . Hence, there exists an instance with density above  $5/6$  that is not schedulable. They conjectured that any instance  $A$  with  $\rho(A) \leq 5/6$  is schedulable. In the second paper by Chan and Chin [1], they improved the achievable density bound to  $7/10$ . Finally, Fishburn and Lagarias [5] showed that any instance with density at most  $3/4$  can be scheduled.

## 2 Approximation algorithm

Our algorithm uses the same approach as the 2-approximation by [6]. Define  $H = \sum_i h_i$ . As observed in [6],  $H$  is a lower bound on the optimal value. Hence, if we find a solution with value  $\alpha H$ , we know that we are within a factor  $\alpha$  of the optimal value. Given an instance of BGT, we initially define periods:

$$p_i := \frac{12H}{7h_i}.$$

The period  $p_i$  can be interpreted as the (possibly fractional) number of days it takes for bamboo  $i$  to reach height  $12H/7$ . Since the  $p_i$ 's need not to be integral, we call  $A = \{p_1, \dots, p_n\}$  a pseudo-instance of the Pinwheel Scheduling problem. A feasible solution for a pseudo-instance of the Pinwheel Scheduling problem is a schedule assigning at most one job per day such that each job  $i$  is scheduled at least once in any period of  $\lfloor p_i \rfloor$  consecutive days. If we are able to find a feasible schedule for the created pseudo-instance, we know that no bamboo will ever exceed height  $12H/7$ .

However, if  $p_1 < 2$ , it follows (assuming  $n \neq 1$ ) that a maximum height of  $H$  is unachievable. Hence, we can increase the lower bound until  $p_1 = 2$  holds. Effectively, this means that we use the lower bound  $H' = \max\{7h_1/6, H\}$ , and create a pseudo-instance of the Pinwheel Scheduling problem by defining periods:

$$p'_i := \frac{12H'}{7h_i}.$$

The created pseudo-instance  $A'$  has density  $\rho(A') \leq 7/12$ , and all periods are at least 2. As noted in the introduction, Chan and Chin [2] designed an algorithm that is able to round any instance of PS with density at most  $7/12$  to a schedulable instance. As it turns out, this algorithm is even able to round any pseudo-instance of PS with density at most  $7/12$  and  $p_1 \geq 2$  to a schedulable instance of PS.

**Theorem 1** *The algorithm of Chan and Chin [2] obtains a schedulable PS-instance for any pseudo-instance  $A$  with  $\rho(A) \leq 7/12$  and  $p_1 \geq 2$ . Moreover, a representation of a feasible schedule for the obtained instance can be found in polynomial time.*

**Corollary 2** *There is a  $12/7$ -approximation algorithm for the discrete Bamboo Garden Trimming problem.*

Next, we will illustrate that the approximation guarantee of  $12/7$  is essentially the best we can hope for when our algorithm is based on a reduction to a pseudo-instance of PS. First, we give an example of a pseudo-instance of PS with density  $7/12 + \delta$ , with  $\delta > 0$ , that is not schedulable.

**Example 3** *Consider the pseudo-instance with  $p_1 = 3 - \epsilon$ ,  $p_2 = 4 - \epsilon$ , and  $p_3 = M$ , with  $\epsilon > 0$  and  $M$  a large number. This pseudo-instance has density*

$$\frac{7}{12} + \delta \text{ with } \delta = \frac{\epsilon}{9 - 3\epsilon} + \frac{\epsilon}{16 - 4\epsilon} + \frac{1}{M}.$$

*In order to obtain a feasible schedule for this pseudo-instance, we should round  $p_1$  to 2. Similarly, we should round  $p_2$  to an integer smaller than or equal to 3. In the most conservative rounding, we obtain the PS-instance  $\{2, 3, \lfloor M \rfloor\}$ , which is known to be non-schedulable [2]. Hence, our pseudo-instance is not schedulable.*

The example above shows that if we reduce instances of BGT to pseudo-instances of PS, and then only focus on schedulability of these pseudo-instances, we will not improve upon the guarantee of  $12/7$ . However, it might be possible that “bad” pseudo-instances like the one in Example 3 are not encountered after the reduction. Now, consider the BGT-instance with  $h_1 = 4$ ,  $h_2 = 3$ , and  $h_3 = \gamma$ , with  $\gamma$  small. If we use  $H'$  as our lower bound, and a magnifying factor of  $12/7 - \eta$ , with  $0 < \eta < 5/7$ , to make the reduction from BGT to a pseudo-instance of PS, we will encounter the pseudo-instance of Example 3. This shows that if we use  $H'$  as our lower bound, algorithms based on a reduction to a pseudo-instance of PS will not give guarantees better than  $12/7$ .

## References

- [1] Mee Yee Chan and Francis Y. L. Chin. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Transactions on Computers*, 41(6):755–768, 1992.
- [2] Mee Yee Chan and Francis Y. L. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9(5):425–462, 1993.
- [3] Federico Della Croce. An enhanced pinwheel algorithm for the bamboo garden trimming problem. *arXiv preprint arXiv:2003.12460*, 2020.
- [4] Martijn van Ee. A  $12/7$ -approximation algorithm for the discrete bamboo garden trimming problem. *Operations Research Letters*, 49(5):645–649, 2021.
- [5] Peter C. Fishburn and Jeffrey C. Lagarias. Pinwheel scheduling: Achievable densities. *Algorithmica*, 34(1):14–38, 2002.

- [6] Leszek Gasieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In *SOFSEM*, pages 229–240. Springer, 2017.
- [7] Robert Holte, Al Mok, Louis Rosier, Igor Tulchinsky, and Donald Varvel. The pinwheel: A real-time scheduling problem. In *Proceedings of the 22th Annual Hawaii International Conference on System Sciences*, volume 2, pages 693–702, 1989.

# On the bamboo garden trimming problem.

Felix Höhne (Speaker) \*

Rob van Stee †

---

In the discrete bamboo garden trimming problem (BGT), first introduced by Gasieniec et al. [1] we are given a set of  $n$  bamboo that grow at rates  $v_1, \dots, v_n$  per day. We assume that these growth rates are arranged such that  $v_1 \geq v_2 \geq \dots \geq v_n$ . Each day a robotic gardener cuts down one bamboo to height zero. The goal is to design a trimming schedule such that the height of the tallest bamboo that ever exists is minimized. Gasieniec et al. gave a 2-approximation for discrete BGT which was improved by van Ee [2] to a  $\frac{12}{7}$ -approximation.

Both results are obtained by reducing BGT to the pinwheel scheduling problem. In the pinwheel scheduling problem, we are given  $n$  jobs with (integer) periods  $p_1, \dots, p_n$ . Time is slotted, and only one job can be scheduled in each time slot. The goal is to design a schedule such that each job  $i$  is scheduled at least once in any period of  $p_i$  consecutive days, or to conclude that no such schedule exists. The density of a pinwheel problem is  $\sum_{i=1}^n \frac{1}{p_i}$ . Chan and Chin [3] conjectured that any pinwheel problem with density at most  $\frac{5}{6}$  can be scheduled. Fishburn and Lagarias [4] showed that any instance with density at most 0.75 can be scheduled and for the special case of  $p_1 = 2$  they give a guarantee of  $10/7$ .

Given an instance of BGT and a desired maximum height  $K$ , we may divide  $K$  by the growth rates and create a set of frequencies the bamboo need to be visited with to maintain a maximum height of  $K$ . Rounding these frequencies down to the nearest integer creates an instance of the pinwheel scheduling problem. A valid schedule for the pinwheel problem also maintains the desired height in BGT.

Gasieniec et al. also consider a continuous version of the BGT problem. In this version the bamboo are distributed in some metric space and the gardener needs to travel between the bamboo to cut them. Cutting is done instantly and the goal is to find a route that minimizes the maximum height of the bamboos. For this problem they provide a  $O(\log n)$ -approximation algorithm that works on any metric space.

**Our results for discrete BGT** We first take a look at discrete BGT for which we propose the following  $10/7$ -approximation algorithm that is also based on a slightly cleverer reduction to the pinwheel-problem.

---

\*felix.hoehne@uni-siegen.de. University of Siegen, Walter-Flex-Strae 3 57072 Siegen

†rob.vanstee@uni-siegen.de. University of Siegen, Walter-Flex-Strae 3 57072 Siegen

---

**Algorithm 1:**

---

Let  $H = \sum_{i=1}^n v_i$  and  $R = \frac{10}{7}$ . Using binary search in the interval  $[H, 2H]$  find the smallest  $K$  such that the following procedure returns a valid schedule and return this schedule.

1. Given  $K$  define  $p_i^* = \lfloor \frac{K}{v_i} \rfloor$  and  $p_i = \lfloor R p_i^* \rfloor$ .
  2. Solve the pinwheel problem  $(p_1, \dots, p_n)$ .
- 

$H$  is a lower bound on the optimal value and it is known that there is an algorithm that produces a schedule of height  $2H$ . It is a lower bound because as long as all bamboo have height at most  $H' < H$  the sum of all bamboo increases by at least  $H - H' > 0$  each step until it exceeds  $nH'$ . Then there must be a bamboo with height more than  $H'$  and since there are only finitely many heights for bamboos that are less than  $H$  at some point there must be a bamboo with height at least  $H$ . Thus the optimal value is somewhere in the interval  $[H, 2H]$ . We show the following theorem to establish the algorithm as a  $10/7$ -approximation.

**Theorem 1** *If there is a schedule that maintains maximum height  $K$  then the procedure in Algorithm 1 finds a schedule that maintains a maximum height of at most  $\frac{10}{7}K$ .*

Since there is a schedule that maintains optimal maximum height within the interval  $[H, 2H]$ , the algorithm, given this theorem, returns a schedule that maintains a maximum height of at most  $10/7$  times the optimal height. What remains is to show that the algorithm can indeed always solve the pinwheel-problem that is presented to it.

To this end we refer to the criteria developed by Fishburn and Lagarias that give bounds on the density required to ensure that the pinwheel problem can be solved. If the frequencies  $p_i$  are small enough, we can immediately argue that their density must be below  $3/4$ , which means the instance can be scheduled. It is also possible that there are a few frequencies that are too large. In these cases we use custom schedules for just the first few large frequencies that leave holes for the remaining frequencies. Afterwards the criteria can again be used to find a schedule that fits the remaining smaller frequencies into the holes.

For the ratio  $\frac{10}{7}$  these subproblems are easier than the original problem. We conjecture that a ratio arbitrarily close to  $\frac{4}{3}$  can be achieved using a more involved case analysis.

**Our results for continuous BGT** For the continuous BGT problem, Gasienic et al. give a  $O(\log n)$  approximation algorithm that works in any metric space. We show that the *deadline driven strategy* is a constant approximation on the star metric if there is a bamboo in every leaf (none in the center). This algorithm always cuts the bamboo with the earliest deadline provided that the height of this bamboo has reached a certain threshold. The deadline of a bamboo is the time it

reaches the height the algorithm wants to maintain. This algorithm has already been considered for discrete BGT and there it is a 2-approximation as shown by John Kuszmaul [5].

After finding a suitable lower bound on OPT and using it to determine deadline and threshold we get a  $(2 + \sqrt{3})$ -approximation on the star graph. It is also possible to extend the argumentation to a star that has multiple plants on each branch. Here our adjustments to the lower bound and the behaviour of the algorithm pay a price in the approximation ratio and lead to a  $(5 + \sqrt{21})$ -approximation for this problem.

We also consider the algorithm *Reduce\_fastest* and show that this algorithm is a 4-approximation on the star graph. This algorithm always cuts the fastest plant provided that the height of this bamboo has reached a certain threshold. For discrete BGT the algorithm is a 2.62-approximation as shown by Bilò et al. [6]

## References

- [1] LESZEK GASIENIEC, RALF KLASING, CHRISTOS LEVCOPOULOS, ANDRZEJ LINGAS, JIE MIN AND TOMASZ RADZIK , Bamboo Garden Trimming Problem (Perpetual Maintenance of Machines with Different Attendance Urgency Factors), *SOFSEM* 2017
- [2] MARTIJN VAN EE , A 12/7-approximation algorithm for the discrete Bamboo Garden Trimming problem, *CoRR* 2020
- [3] MEE YEE CHAN AND FRANCIS Y. L. CHIN , General Schedulers for the Pinwheel Problem Based on Double-Integer Reduction, *IEEE Trans. Computers* 1992
- [4] PETER C. FISHBURN AND J. C. LAGARIAS , A 12/7-approximation algorithm for the discrete Bamboo Garden Trimming problem, *Algorithmica* 2002
- [5] JOHN KUSZMAUL, Bamboo Trimming Revisited: Simple Algorithms Can Do Well Too, *CoRR* 2020
- [6] DAVIDE BILÒ, LUCIANO GUALÀ, STEFANO LEUCCI, GUIDO PROIETTI AND GIACOMO SCORNAVACCA, Cutting Bamboo down to Size, *FUN* 2021

# Bamboo Trimming Revisited: Simple Algorithms Can Do Well Too

John Kuszmaul (Speaker) \*

---

## 1 Introduction

The bamboo trimming problem considers  $n$  bamboo with growth rates  $h_1, h_2, \dots, h_n$  satisfying  $\sum_i h_i = 1$ . During a given unit of time, each bamboo grows by  $h_i$ , and then the bamboo-trimming algorithm gets to trim one of the bamboo back down to height zero. The goal is to minimize the height of the tallest bamboo, also known as the backlog. The bamboo trimming problem is closely related to many scheduling problems, and can be viewed as a variation of the widely-studied fixed-rate cup game, but with constant-factor resource augmentation.

Past work has given sophisticated pinwheel algorithms [4] that achieve the optimal backlog of 2 in the bamboo trimming problem. It remained an open question, however, whether there exists a *simple* algorithm with the same guarantee—recent work has devoted considerable theoretical and experimental effort to answering this question. Two algorithms, in particular, have appeared as natural candidates: the **Reduce-Max** algorithm (which always cuts the tallest bamboo) and the **Reduce-Fastest**( $x$ ) algorithm (which cuts the fastest-growing bamboo out of those that have at least some height  $x$ ). It is conjectured that **Reduce-Max** and **Reduce-Fastest**(1) both achieve backlog 2.

This paper improves the bounds for both **Reduce-Fastest** and **Reduce-Max**. Among other results, we show that the *exact optimal* backlog for **Reduce-Fastest**( $x$ ) is  $x + 1$  for all  $x \geq 2$  (this proves a conjecture of [3] in the case of  $x = 2$ ), and we show that **Reduce-Fastest**(1) *does not* achieve backlog 2 (this disproves a conjecture of [3]).

Finally, we show that there is a different algorithm, which we call the **Deadline-Driven** Strategy, that is both very simple and achieves the optimal backlog of 2. This resolves the question as to whether there exists a simple worst-case optimal algorithm for the bamboo trimming problem.

---

\*john.kuszmaul@yale.edu. Computer Science Department, Yale University,

## 2 Background

A classic scheduling problem is the so-called *cup game*, which is a two-player game that takes place on  $n$  cups. In each step of the game, the filler player distributes 1 unit of water among the cups arbitrarily; the emptier player then selects a single cup and removes up to 1 unit of water from that cup. The emptier’s goal is to minimize the backlog of the system, which is defined to be the amount of water in the fullest cup.

The cup game was first introduced in the late 1960s [6], and has been studied in many different forms. The game has found extensive applications in areas such as processor scheduling, network-switch buffer management, quality of service guarantees, and data-structure deamortization. See the full version of this paper as well as [5] for a more detailed discussion of the related work.

Perhaps the most natural emptying algorithm is the **Reduce-Max** algorithm, which always empties from the fullest cup. **Reduce-Max** achieves an asymptotically optimal backlog of  $O(\log n)$  [1]. In fact, in addition to being asymptotically optimal, **Reduce-Max** is known to be *exactly* optimal—no other algorithm can do better, even by an additive constant [1].

An important special case of the cup game is the setting where the filler’s behavior is the same on every step, also known as the fixed-rate cup game. Whereas the optimal backlog in the variable-rate cup game is  $O(\log n)$ , the optimal backlog in the fixed-rate cup game is  $O(1)$  [4, 6]. Perhaps surprisingly, though, the **Reduce-Max** algorithm is no longer optimal (or even asymptotically optimal!). In fact, the algorithm still allows for backlog  $\Omega(\log n)$  in the fixed-rate setting [1].

Recent work has identified a potential path to redemption for the **Reduce-Max** algorithm, however. Bilò, Gualà, Leucci, Proietti, and Scornavacca [2] showed that, if the emptier is given resource augmentation over the filler, meaning that the emptier is permitted to fully empty a cup on each step rather than removing just a single unit of water, then the backlog achieved by the **Reduce-Max** algorithm becomes  $O(1)$ . Note that, since the backlog is constant, the resource augmentation never results in the emptier removing more than  $O(1)$  units of water at a time.

Although there is a long history of studying resource-augmented variants of the cup game, it is only relatively recently that researchers have begun to study the resource-augmented fixed-rate version of the game [4]. These papers have dubbed the problem as the *Bamboo Garden Trimming Problem*, based on the following (rather creative) problem interpretation. A robotic panda gardener is responsible for maintaining a bamboo garden. The garden consists of  $n$  bamboo  $b_1, \dots, b_n$  with corresponding growth-rates  $h_1 \geq \dots \geq h_n$  satisfying  $\sum_{i=1}^n h_i = 1$ . Each bamboo  $b_i$  starts at height 0 and grows at a steady rate of  $h_i$  every time unit. At the end of each time unit, the player chooses one of the bamboo and chops this bamboo down to height 0. The goal, of course, is to achieve the smallest possible backlog, which is the height of the tallest bamboo.

It is known that no bamboo trimming algorithm can guarantee a backlog less than 2, as it is possible to achieve backlog at least  $2 - 2\epsilon$  against any bamboo

trimming algorithm with two bamboo that have fill rates  $1 - \varepsilon$  and  $\varepsilon$  [2]. Recent work has yielded complex pinwheel algorithms [4] that achieve backlog 2, and are thus optimal in terms of the worst-case backlog; there has also been effort to extend the guarantees of these algorithms to achieve strong competitive ratios for cases where backlog less than 2 is possible.

**The quest for a simple optimal bamboo-trimming algorithm.** The relative complicatedness of the known pinwheel algorithms has sparked a great deal of interest in the question as to whether there exists some *simple* algorithm that achieves the optimal backlog of 2. It would be especially interesting if **Reduce-Max** were to achieve backlog 2, since this would mean that the algorithm is optimal for both bamboo-trimming and the standard cup game. Currently, the best known bound for **Reduce-Max** is a backlog of 9 [2]. Experimental work [3] has found that **Reduce-Max** does, in fact, seem to achieve a backlog of 2, however, leading the authors to pose a backlog of 2 as a conjecture.

Another algorithm family that has been studied for its simplicity is **Reduce-Fastest**( $x$ ). This algorithm trims down the fastest-growing bamboo out of those that have height at least  $x$  at the end of each time unit. Initial study proved that **Reduce-Fastest**(2) achieves backlog at most 4 [4], and further work demonstrated that **Reduce-Fastest**( $x$ ) achieves backlog at most  $\max(x + \frac{x^2}{4(x-1)}, \frac{1}{2} + x + \frac{x^2}{4(x-1/2)})$  for all  $x > 1$ , which yields a bound of 19/6 at  $x = 2$  [2] (and is  $\geq 1.25x$  for all  $x > 1$ ). Extensive computer experimentation [3] suggests that this bound of 19/6 is still not optimal, and has led researchers to conjecture that **Reduce-Fastest**(2) actually achieves a backlog of 3. Based on the same experiments, the authors further conjecture that **Reduce-Fastest**(1) achieves the optimal backlog of 2 [3]. However, as of now, no theoretical bounds on the backlog of **Reduce-Fastest**(1) are known.

**Our results.** Our first result is an improved bound on the backlog of **Reduce-Max** for bamboo trimming. We prove that **Reduce-Max** achieves a backlog of 4, which narrows the gap between the upper and lower bounds from 7 to 2. At a technical level, our bound relies on a novel potential-function argument; we believe that this argument may be of independent interest as a tool that could help in future analyses of similar problems.

Our second set of results analyze **Reduce-Fastest**( $x$ ) for different values of  $x$ . We are able to prove that **Reduce-Fastest**( $x$ ) achieves backlog  $x + 1$  for all  $x \geq 2$ , and we give a matching lower bound showing that this analysis is tight. This is the first time that a tight analysis has been achieved for **Reduce-Fastest**( $x$ ) for any value of  $x$ . For  $x = 2$ , the result gives a backlog of 3, which resolves a conjecture of [3]. On the other hand, we disprove the conjecture of [3] that **Reduce-Fastest**(1) achieves backlog 2. Instead, we show that **Reduce-Fastest**(1) allows for a backlog of  $3 - \varepsilon$  for any  $\varepsilon > 0$ . More generally, we show that there is no  $x$  for which **Reduce-Fastest**( $x$ ) achieves a backlog of 2.01, meaning that **Reduce-Fastest**( $x$ ) is no longer a candidate in the quest for a simple optimal algorithm.

Our final result is a simple algorithm, which we call the *Deadline-Driven Strategy*, that does in fact achieve a backlog of 2. The algorithm, which is based upon Liu and Layland’s algorithm from the early 70s for a related scheduling problem [6], maintains the simplicity associated with **Reduce-Max** and **Reduce-Fastest**( $x$ ), while also matching the backlog bounds of the more complicated pinwheel-based algorithms.

The **Deadline-Driven Strategy** selects the bamboo that will soonest achieve height 2 of the bamboo that have height at least 1. From a scheduling standpoint, we can consider the time at which a bamboo achieves height 2 to be its deadline. From this perspective, the **Deadline-Driven Strategy** is simply chopping down the bamboo with the closest deadline, while not considering very short bamboo with height less than 1.

## Acknowledgments

I would like to thank Michael A. Bender and William Kuszmaul for their mentorship and advice during this project. This research was funded by National Science Foundation Grants CNS-1938709 and CCF-2106827.

## References

- [1] M. Adler, P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. Goldberg, and M. Paterson. A proportionate fair scheduling rule with good worst-case performance. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 101–108, 2003.
- [2] B. Davide, L. Guala, L. Stefano, P. Guido, and S. Giacomo. Cutting bamboo down to size. In *The Tenth International Conference on Fun with Algorithms (FUN 2020)*, 2020.
- [3] M. D’Emidio, G. Di Stefano, and A. Navarra. Bamboo garden trimming problem: Priority schedulings. *Algorithms*, 12(4):74, 2019.
- [4] L. Gaśieniec, R. Klasing, C. Levcopoulos, A. Lingas, J. Min, and T. Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 229–240. Springer, 2017.
- [5] W. Kuszmaul. Achieving optimal backlog in the vanilla multi-processor cup game. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1558–1577. SIAM, 2020.
- [6] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

# An approximation algorithm for sequencing unreliable jobs on parallel machines with job duplication

Ben Hermans (Speaker) \*    Alessandro Agnetis †    Mario Benini ‡  
Paolo Detti §            Marco Pranzo ¶

---

## 1 Introduction

We study the problem of scheduling a set  $N = \{1, \dots, n\}$  of  $n$  unreliable jobs on  $m$  identical machines, where jobs can be duplicated in order to increase the probability of being successful on at least one machine. When a job  $j \in N$  is performed on a certain machine, then it succeeds with a given *success probability*  $p_j$ . If the job fails, then the machine gets blocked and cannot perform any further jobs. We say that a job is *successful* on a machine if and only if all jobs preceding it, as well as the job itself, succeed. There are  $m$  exact copies of each job, such that a copy of each job can be attempted on each machine. A job is then said to be *carried out* if and only if it has a successful copy on at least one machine. Failures are assumed to occur independently, both between jobs and between their copies. Given a revenue  $r_j$  for each job  $j \in N$  that is attained if and only if the job is carried out, we study the problem of finding a sequence of the  $n$  job copies for each of the  $m$  machines so as to maximize the expected revenue. This problem was originally introduced in [1] and is NP-hard, even if  $m = 2$ .

One possible application of this problem is in the setting of scientific computing, where there are multiple unsupervised servers available to perform certain computational tasks. Each successfully completed task leads to a reward, but each task also has a probability to fail (e.g., due to a bug), which then causes the server to be blocked.

If there is only one machine (i.e.,  $m = 1$ ), then it is well-known that an optimal sequence can be found by following the so-called *Z-rule* [2, 3]. More specifically,

---

\*ben.hermans@kuleuven.be. Research Center for Operations Research & Statistics (OR-STAT), KU Leuven, Belgium. Postdoctoral fellow of the Research Foundation – Flanders.

†agnetis@dii.unisi.it. Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, Italy.

‡mbenini@dii.unisi.it. Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, Italy.

§detti@dii.unisi.it. Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, Italy.

¶pranzo@dii.unisi.it. Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, Italy.

Table 1: Approximation guarantee for  $m \in \{1, \dots, 20\}$ .

$m$	$H_m/m$	$m$	$H_m/m$	$m$	$H_m/m$	$m$	$H_m/m$
1	1	6	0.4083	11	0.2745	16	0.2113
2	0.75	7	0.3704	12	0.2586	17	0.2023
3	0.611	8	0.3397	13	0.2446	18	0.1942
4	0.5208	9	0.3143	14	0.2323	19	0.1867
5	0.4567	10	0.2929	15	0.2212	20	0.1799

define the *Z-ratio* of a job  $j \in N$  as

$$Z_j = \frac{p_j r_j}{1 - p_j},$$

then it holds that a sequence is optimal if and only if it schedules the jobs in non-increasing order of their Z-ratio. In this talk, we show that following the Z-rule on each machine guarantees an expected revenue that is within factor  $H_m/m$  of the maximum, and that this bound is asymptotically tight. Here,  $H_m = \sum_{k=1}^m 1/k$  denotes the  $m^{\text{th}}$  harmonic number; Table 1 lists the approximation guarantee for different values of  $m$ .

## 2 Establishing the approximation guarantee

We establish the approximation guarantee in three steps. Firstly, we obtain an upper bound on the maximum expected revenue by studying a relaxation in which one can decide upon a schedule for each machine sequentially. That is, only when a certain machine gets blocked by a job failure, the next machine's schedule needs to be decided. We show that, for this relaxation, it is optimal to always schedule first the jobs that have not been carried out yet, and this in non-increasing order of their Z-ratio. We refer to the resulting upper bound as the *sequential upper bound*. This result and the ensuing bound are interesting in their own right, since they might also be useful to assess the quality of other heuristics.

Secondly, we characterize a number of structural properties of the sequential upper bound using induction on the instance size. Interestingly, we find that if job  $j$  has the maximum Z-ratio, then the maximum expected revenue is upper bounded by  $mZ_j$ . This provides a new interpretation of the Z-ratio. Combined, the derived properties allow us to prove the approximation guarantee.

Finally, we establish that the bound is asymptotically tight. To show this, we consider a family of instances where all jobs have unit reward and the same success probability  $p$ . By letting the number of jobs  $n$  tend to infinity and the success probability  $p$  to one, we then show that the bound is attained. Since all jobs are identical, this also establishes that our bound is essentially the best possible for all heuristics that consider the same schedule on all machines.

## References

- [1] AGNETIS, A., BENINI, M., DETTI, P., HERMANS, B., AND PRANZO, M. (2021). ‘Replication and sequencing of unreliable jobs on parallel machines’, *Computers & Operations Research*. (Accepted).
- [2] AGNETIS, A., DETTI, P., PRANZO, M., AND SODHI, M. S. (2009). ‘Sequencing unreliable jobs on parallel machines’, *Journal of Scheduling* **12**(1), 45–54.
- [3] STADJE, W. (1995). ‘Selecting jobs for scheduling on a machine subject to failure’, *Discrete applied mathematics* **63**(3), 257–265.

# Complexity of partitioned scheduling for periodic tasks

Pontus Ekberg (Speaker) \*      Sanjoy Baruah †

---

## 1 Introduction

We consider the partitioned scheduling of a set of periodic tasks  $\mathcal{T} = \{\tau_1, \dots, \tau_n\}$  on  $m$  identical processors. A periodic task  $\tau_i = (o_i, c_i, d_i, p_i) \in \mathbb{N}^4$  releases a job at each time point  $o_i + kp_i$  for  $k \in \mathbb{N}$ . Every job released by  $\tau_i$  has an execution time requirement of  $c_i$  time units and a deadline  $d_i$  time units after its release.

We say that  $\mathcal{T}$  is *synchronous* if  $o_i = 0$  for all  $\tau_i$  (and *asynchronous* otherwise). We say that  $\mathcal{T}$  has *implicit deadlines* if  $d_i = p_i$  for all  $\tau_i$  and has *constrained deadlines* if  $d_i \leq p_i$  for all  $\tau_i$  (and has *arbitrary deadlines* otherwise).

We want to understand the complexity of the following family of decision problems.

**Definition 1** (PARTITIONED  $\mathcal{A}$ -SCHEDULABILITY).

Instance:  $\langle \mathcal{T}, m \rangle$ , where  $\mathcal{T}$  is a set of periodic tasks and  $m$  is the number of processors.

Question: Is there an  $m$ -partitioning  $\mathcal{T}_1, \dots, \mathcal{T}_m$  of  $\mathcal{T}$  such that each partition  $\mathcal{T}_i$  is schedulable by scheduling algorithm  $\mathcal{A}$  on a single processor with no deadline misses?

Partitioned scheduling is a very commonly used strategy for real-time systems. It reduces overhead costs by disallowing migrations between processors and importantly allows the reuse of mature results for single-processor scheduling in a multiprocessor setting.

It is not difficult to see that PARTITIONED  $\mathcal{A}$ -SCHEDULABILITY generalizes BIN PACKING in most settings and is NP-hard, but other previously known lower bounds on its complexity are only the ones for the corresponding single-processor  $\mathcal{A}$ -schedulability problems. We want to narrow down the complexity, which among other things lets us find a distinction between the partitioned schedulability problems can efficiently be formulated as ILP or SAT instances, and which cannot unless the polynomial hierarchy collapses. To find better bounds we first define a partitioned version of a helpful number-theoretic decision problem that has been

---

\*[pontus.ekberg@it.uu.se](mailto:pontus.ekberg@it.uu.se). Department of Information Technology, Uppsala University, P.O. Box 337, SE-751 05 Uppsala, Sweden.

†[baruah@wustl.edu](mailto:baruah@wustl.edu). Department of Computer Science & Engineering, MSC 1045-213-1010J, Washington University in Saint Louis, 1 Brookings Drive, St. Louis, MO 63130-4899.

used for several hardness results for periodic tasks in single processor systems, the Simultaneous Congruences Problem.

## 2 Partitioned simultaneous congruences

The Simultaneous Congruences Problem (SCP) concerns finding a set of congruence classes that all share some element. The SCP was shown to be NP-complete by Leung and Whitehead [3] (and was later shown to be so in the strong sense by Baruah et al. [1]).

**Definition 2** (SCP).

Instance:  $\langle A, k \rangle$ , where  $A = \{(a_1, b_1), \dots, (a_n, b_n)\}$  is a set of pairs of non-negative integers satisfying  $a_i < b_i$  for each  $i$ , and  $k$  is a positive integer.

Question: Is there an integer  $x$  and an  $A' \subseteq A$  such that  $|A'| = k$  and

$$x \equiv a_i \pmod{b_i}$$

for each  $(a_i, b_i) \in A'$ ?

We define PARTITIONED SCP as a natural generalization to the SCP.

**Definition 3** (PARTITIONED SCP).

Instance:  $\langle A, k, m \rangle$ , where  $A = \{(a_1, b_1), \dots, (a_n, b_n)\}$  is a set of pairs of non-negative integers satisfying  $a_i < b_i$  for each  $i$ , and  $k$  and  $m$  are positive integers.

Question: Is there an  $m$ -partitioning  $A_1, \dots, A_m$  of  $A$  such that  $\langle A_i, k \rangle \notin \text{SCP}$  for each partition  $A_i$ ?

Through a reduction from GENERALIZED GRAPH COLORING [4] we can pinpoint the complexity of PARTITIONED SCP. The reduction is closely inspired by Leung and Whitehead's reduction from CLIQUE to SCP [3] and works by assigning congruences classes to vertices such that a set of congruences classes have a joint element if and only if the corresponding set of vertices form a clique.

**Theorem 4.** PARTITIONED SCP is  $\Sigma_2^P$ -complete, even when  $m = 2$ .

## 3 Asynchronous tasks

For asynchronous periodic tasks we find a new general lower bound by reducing from PARTITIONED SCP in a fairly straight-forward manner.

**Theorem 5.** The partitioned  $\mathcal{A}$ -schedulability problem for asynchronous periodic tasks with constrained (or arbitrary) deadlines is  $\Sigma_2^P$ -hard for any work-conserving scheduler  $\mathcal{A}$  (preemptive or non-preemptive), even when restricted to  $m = 2$  processors and task sets with  $\sum_{\tau_i \in \mathcal{T}} \frac{c_i}{p_i} < \varphi$  for any constant  $\varphi > 0$ .

When  $\mathcal{A}$  is an optimal preemptive single-processor scheduling algorithm, such as Earliest Deadline First (EDF), this can be shown to be a tight bound.

**Theorem 6.** *The partitioned EDF-schedulability problem for asynchronous periodic tasks with constrained (or arbitrary) deadlines is  $\Sigma_2^P$ -complete.*

In contrast, for preemptive Fixed Task Priority (FP) scheduling we still have a gap to the lower bound of Theorem 5.

**Theorem 7.** *The partitioned FP-schedulability problem for asynchronous periodic tasks is in  $\Sigma_3^P$ . This holds both if we are given the priority ordering or are asked to find one.*

The partitioned EDF-schedulability problem with implicit deadlines is essentially just plain BIN PACKING and is NP-complete. However, FP-scheduling with implicit deadlines can be shown to be harder than EDF when we are given a predefined priority ordering.

**Theorem 8.** *The partitioned FP-schedulability problem for asynchronous periodic tasks is  $\Sigma_2^P$ -hard if we are given a priority ordering, even when restricted to implicit-deadline tasks,  $m = 2$  processors and task sets with  $\sum_{\tau_i \in \mathcal{T}} \frac{c_i}{p_i} < \varphi m$  for any constant  $\varphi > 1/2$ .*

It is an open problem if this remains  $\Sigma_2^P$ -hard if we are instead asked if the set of tasks is FP-schedulable with *any* priority ordering.

## 4 Synchronous tasks

There is relatively less to say about synchronous periodic tasks at this time, but we have the following observations.

**Theorem 9.** *With synchronous periodic tasks, partitioned EDF-schedulability with implicit deadlines and partitioned FP-schedulability with implicit or constrained deadlines are NP-complete.*

The above follows directly from observing that the corresponding single processor problems are in NP. However, EDF-schedulability with constrained or arbitrary deadlines is not contained in the first level of the polynomial hierarchy (unless it collapses), but is contained in the second.

**Theorem 10.** *Partitioned EDF-schedulability of synchronous periodic tasks with constrained or arbitrary deadlines is both NP- and coNP-hard, and is in  $\Sigma_2^P$ .*

FP-schedulability with arbitrary deadlines has a larger gap still. (Here even the single processor case is not yet pinpointed!)

**Theorem 11.** *Partitioned FP-schedulability of synchronous periodic tasks with arbitrary deadlines is NP-hard, and is in  $\Sigma_3^P$ .*

We do suspect that the partitioned EDF-schedulability problem with constrained or arbitrary deadlines (as in Theorem 10) is in fact  $\Sigma_2^P$ -complete, just as with asynchronous tasks, but have yet been unable to show this. It seems significantly more difficult to construct a reduction from PARTITIONED SCP here due to the lack of the tasks' offset parameters  $o_i$ , which in many ways mirror the  $a_i$  parameters in PARTITIONED SCP. Still, such a reduction has been done from SCP to the single processor synchronous case [2], so there is still hope that PARTITIONED SCP can help in resolving this important outstanding problem.

## References

- [1] S. BARUAH, R. HOWELL, AND L. ROSIER, *Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor*, Real-Time Systems: The International Journal of Time-Critical Computing, 2 (1990), pp. 301–324.
- [2] P. EKBERG AND W. YI, *Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-complete*, in The 27th Euromicro Conference on Real-Time Systems, 2015, pp. 281–286.
- [3] J. Y.-T. LEUNG AND J. WHITEHEAD, *On the complexity of fixed-priority scheduling of periodic, real-time tasks*, Performance Evaluation, 2 (1982), pp. 237–250.
- [4] V. RUTENBURG, *Complexity of generalized graph coloring*, in Mathematical Foundations of Computer Science, Springer Berlin Heidelberg, 1986, pp. 573–581.

# Periodic scheduling with shared resources

Vít Koštejn (Speaker) \*

Jiří Sgall \*

---

We study a scheduling problem motivated by the creation of a startlist schedule for orienteering. Orienteering is an individual sport in which athletes navigate themselves through the forest with a map. A competition consists of multiple categories with different tasks. There are up to 2000 athletes and up to 50 categories in the national competitions. To give an efficient schedule, multiple athletes from different categories start in parallel at the same time. The maximal number of parallel starts is given. Also, some categories have the same first navigation point, so their athletes cannot start at the same time. To ensure maximum fairness, the athletes of each category start with the same gap between each other. This gap is called period of the category and for each category we are given its minimal value.

## 1 Problem definition

Now, we translate the problem into scheduling terminology. The categories are represented as periodic jobs with the length equal to the number of athletes and a given minimal period. Furthermore, there are resources that correspond to the first navigation point, and each category requires exactly one of these resources. Finally, the number of machines corresponds to the given maximal number of athletes that can start in parallel. The time is discrete, each start of an athlete is an action of the corresponding periodic job that takes one time step, we also say it takes one minute.

**Definition 1** *Let  $R$  be a finite set of resources. Then we define a periodic job  $j$  as a triple  $(p_j, f_j, g_j)$ , where*

- $p_j \in \mathbb{N}$  is the number of actions of job  $j$ ,
- $f_j \in R$  is the resource of job  $j$ , and
- $g_j \in \mathbb{N}$  is the minimal period between the actions of job  $j$ .

The goal of the problem is then to schedule a set of periodic jobs within some restrictions and minimize the makespan.

---

\*v.kostejn.vk@gmail.com, sgall@iuuk.mff.cuni.cz, Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic. Partially supported by the projects 19-27871X and 22-22997S of GA CR.

**Definition 2 (Periodic scheduling with shared resources (PSR))**

Periodic scheduling with shared resources is an optimization problem whose input is a finite set of periodic jobs  $J$  and the number of machines  $m \in \mathbb{N}$  and output is schedule that specifies for each job  $j$

- $S_j \in \mathbb{N}$ , the starting time of job  $j$ ,
- $C_j \in \mathbb{N}$ , the completion time of job  $j$  and
- $G_j \in \mathbb{N}$  the period of job  $j$ .

satisfying

$$(\forall j \in J) (C_j = S_j + (p_j - 1) \cdot G_j), \quad (1)$$

$$(\forall j \in J) (G_j \geq g_j), \quad (2)$$

$$(\forall i, j \in J, f_i = f_j) (\forall a_i \in \{0, \dots, p_i - 1\}, \forall a_j \in \{0, \dots, p_j - 1\}) \quad (3)$$

$$(S_i + a_i \cdot G_i \neq S_j + a_j \cdot G_j),$$

$$(\forall t \in \{0, \dots, e\}) \quad (4)$$

$$(|\{j \in J \mid (\exists a \in \{0, \dots, p_j - 1\})(S_j + a \cdot G_j = t)\}| \leq m).$$

The objective is to minimize the makespan, i.e.,  $C_{\max} = \max_{j \in S} C_j$ .

Condition (1) defines  $C_j$  as the completion time for each job and (2) constrains the period to be at least the minimum period. Condition (3) says that actions of the jobs with the same resource are not scheduled in the same time step. The capacity condition (4) limits the number of actions for each time to be at most the number of machines  $m$ .

## 2 Our contributions

This problem is strongly NP-hard even if all jobs have distinct resources and equal minimal periods, and in addition either we have a single machine or the common minimal period is one. Thus we focus on analyzing greedy approximation algorithms.

It should be noted that complexity considerations are delicate for two reasons.

First, the definition of the problem corresponds to scheduling on multiple machine where migration of jobs among different machines is allowed. This together with the flexibility of periods gives the problem preemptive nature which complicates the NP-hardness proof. Our NP-hardness proof uses a reduction from numerical 3D-matching where we arrange the lengths of the jobs so that all the periods are necessarily equal to the minimal one in the optimal schedule, in which case the migration is irrelevant.

Second, we use a compact representation of the input where the actual schedule can have exponential length if some of  $p_j$  or  $g_j$  on input are large. Thus it is not

even clear if we can check a feasibility of a given schedule in polynomial time. Nevertheless, all our algorithms are polynomial even in the size of this compact representation and guaranteed to produce feasible solutions.

**Multiple machines.** We begin to analyze the special case with the restriction that the minimum periods are the same for all jobs. We analyze a greedy algorithm that schedules each job at the first possible time in the schedule and prove the approximation ratio 3 for this algorithm. We prove that the algorithms that set  $(\forall j)G_j = g_j$  (similarly to the greedy algorithm) cannot have an approximation ratio smaller than 2.

The next special case that we analyze is restricted even more, namely the minimal period is 1 for each job. If the same greedy algorithm is used as before, we can get the approximation ratio 2. We construct a better algorithm that takes the longest job (with the highest  $p_j$ ) from the most frequent resource (according to length of pending jobs using this resource). Our best contribution is upper bound of  $\frac{22+\sqrt{3}}{13}$  ( $\approx 1.83$ ) on the approximation ratio for the algorithm. This improved bound is obtained by a delicate accounting for both large jobs and resources. We also show that this algorithm cannot obtain an approximation ratio lower than  $\frac{11}{8}$ .

**Single machine.** In addition, we analyze the special case on a single machine and with the restriction that the minimum period is a power of two for each job. We give an algorithm with an approximation ratio 2 for this case. We prove that the ratio cannot be better if we use only the length of the job and the volume of all jobs (the number of all actions) to bound the optimum.

### 3 Related work

While scheduling with resource constraints is a wide area, very little is known about periodic jobs in these settings. We are not aware of any previous work studying our specific problem.

Most related is the subarea that studies the special case of resource constraints with renewable resources and a single resource needed for each job. The case of non-periodic jobs with general processing times, corresponds to periodic jobs with all periods equal to 1, without possibility to increase the periods. Hebrard *et al* give a  $(2 - 2/(m + 1))$ -approximation algorithm for makespan in this setting. Janssen *et al* [3] studies the same problem with a different objective, namely the total completion time, and gives a 2-approximation algorithm. See also the survey [1] for this area.

Finally, let us mention that our setting is very different from periodic jobs in real-time scheduling. There the number of repetitions is unbounded (or infinite) and the main question is feasibility; that setting is not compatible with the makespan objective.

## References

- [1] E. B. EDIS, C. OGUZ, I. OZKARAHAN. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European J. Oper. Res.* 230:449–463, 2013.
- [2] E. HEBRARD, M.-J. HUGUET, N. JOZEFOWIEZ, A. MAILLARD, C. PRALET, G. VERFAILLIE. Approximation of the parallel machine scheduling problem with additional unit resources. *Discrete Appl. Math.* 215:126–135, 2016.
- [3] T. JANSSEN, C. SWENNENHUIS, A. BITAR, T. BOSMAN, D. GIJSWIJT, L. VAN IERSEL, S. DAUZÉRE-PÉRÈS, AND C. YUGMA. Parallel Machine Scheduling with a Single Resource per Job. Preprint arXiv:1809.05009, 2018.

# Exact Polynomial Time Algorithm for the Response Time Analysis of Harmonic Tasks

Thi Huyen Chau Nguyen \*      Werner Grass †  
Klaus Jansen (Speaker) ‡

---

## 1 Problem Definition

In this work, we analyze a list  $\mathcal{T}_n = \tau_1, \tau_2, \dots, \tau_n$  of  $n$  hard real-time sporadic *tasks*, each one releasing a sequence of *jobs* at time 0. The tasks are scheduled over a single processor by Preemptive Fixed Priorities (PFP). Task  $\tau_i$  is characterized by:

- a minimum time  $T_i$  (that we call *period*) between the arrival of two consecutive jobs,
- a worst-case *execution time*  $C_i$ ,
- a *relative deadline*  $D_i$  which is the time interval between the arrival time of a job and the time at which the job should be completed, and
- a *fixed priority* which is implicitly given by the order in the task list:  $\tau_i$  has a higher priority than  $\tau_j$  if and only if  $i < j$ . At a given time, the processor executes the task with the highest priority of all tasks currently ready for execution. A running task at this time with a lower priority is preempted.

The task periods are assumed to be harmonic that is  $T_i$  divides  $T_j$  or vice versa ( $T_i|T_j$ ) or ( $T_j|T_i$ ). All task parameters are positive integer numbers. We assume *constrained deadlines* i.e.,  $C_i \leq D_i \leq T_i$ . The ratio  $U_i = C_i/T_i$  denotes the *utilization* of task  $\tau_i$ , that is, the fraction of time required by  $\tau_i$  to execute. We denote with  $U_{k..k'} = \sum_{j=k}^{k'} U_j$  the utilization of a sequence of tasks  $\tau_k, \dots, \tau_{k'}$ . Notice that the utilization  $U_{1..n}$  of entire task set is equal to  $\sum_{j=1}^n C_j/T_j \leq 1$ . If  $U_{1..n} > 1$ , then there are job sequences in  $\mathcal{T}_n$  that cannot be feasibly scheduled by any algorithm [5].

Consider a task system with a given fixed priority order. We can consider without loss of generality the synchronous arrival sequence (SAS) of jobs. In SAS,

---

\*chaunth@thanglong.edu.vn. Department of Information Technology, Thang Long University (TLU), Dai Kim, Hoang Mai, Hanoi, Vietnam.

†grass@fim.uni-passau.de. Faculty of Computer Science and Mathematics, University of Passau, Germany.

‡kj@informatik.uni-kiel.de. Institut für Informatik, Universität Kiel, Germany.

the response time of the first task  $\tau_i$  is the largest among the jobs of  $\tau_i$  [2, 3, 8, 12]. Hence, to test the schedulability of a task system, it is sufficient to compute for each task  $\tau_i$  the response time  $R_i$  of the first job of  $\tau_i$  and to check whether  $R_i \leq D_i$  for each  $i = 1, \dots, n$  [6, 9].

It is known [1] that  $R_i$  of a task  $\tau_i$  is the minimum  $t > 0$  that satisfies the following equality:

$$C_i + \sum_{j < i} \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j = t. \quad (1)$$

## 2 New Result

For simplification in the notation, from now on we consider the *worst-case response time* of task  $\tau_n$  but our results could easily be applicable for any  $i < n$ . Now in order to keep the calculation of the indices simple in the various processing steps described below, we choose an inverse rate-monotonic order. Such rearrangements were also made in [4, 5]. To formally describe this reordering we introduce a bijective mapping

$$\pi : 1 \dots n - 1 \rightarrow 1 \dots n - 1, \quad (2)$$

in which  $\pi(i) = k$  signifies that task  $\tau_k$  with priority  $k$  is at position  $i$  in the new order. The reverse rate monotonic order satisfies the condition that for all  $i < j$  period  $T_{\pi(j)}$  divides the period  $T_{\pi(i)}$  having the priorities  $\pi(i)$  and  $\pi(j)$ , respectively.

In the talk we prove the following main result:

**Theorem 1** *Let a set of  $n$  tasks be given, where the first  $n - 1$  tasks are ordered such that  $T_{\pi(n-1)} | T_{\pi(n-2)} | \dots | T_{\pi(2)} | T_{\pi(1)}$  and  $U_{\pi(1)\dots\pi(n-1)} < 1$  applies. Then the least fixed point of the equation*

$$R_n = C_n + \sum_{i=1}^{n-1} C_{\pi(i)} \left\lceil R_n / T_{\pi(i)} \right\rceil \quad (3)$$

can be obtained in  $\mathcal{O}(n)$  time by applying the iterative formula:

$$\tilde{R}_n^{(0)} = \frac{C_n}{1 - U_{\pi(1)\dots\pi(n-1)}} \quad (4)$$

$$1 \leq i \leq n - 1, \quad \tilde{R}_n^{(i)} = \tilde{R}_n^{(i-1)} + \frac{C_{\pi(i)} \left( \left\lceil \frac{\tilde{R}_n^{(i-1)}}{T_{\pi(i)}} \right\rceil - \frac{\tilde{R}_n^{(i-1)}}{T_{\pi(i)}} \right)}{1 - U_{\pi(i+1)\dots\pi(n-1)}} \quad (5)$$

we finally get  $R_n = \tilde{R}_n^{(n-1)}$ .

In comparison, the algorithm in [5] (which uses the same assumptions as our algorithm) has the complexity  $\mathcal{O}(n \cdot (\log T_{max} + \log n))$  with  $T_{max} = \max_{1 \leq i \leq n} (T_i)$ . In [13] an algorithm has been proposed that is also based on a binary search but

with a slightly reduced complexity  $\mathcal{O}(n \cdot (\log T_{max} - \log T_{min}))$  with  $T_{min} = \min_{1 \leq i \leq n}(T_i)$  to compute the response time of task  $\tau_n$  if the priorities are rate monotonic (i.e. decrease with increasing or equal periods).

Notice that the algorithm above has the disadvantage that it has to perform floating point operations. On the other hand, we are able to modify our algorithm so that only integer operations are required. For this we calculate  $1 \leq i \leq n - 1$ ,  $\tilde{R}_n(1 - U_{\pi(i+1)\dots\pi(n-1)})$  (see also our full version [10, 11]).

### 3 Conclusion

Response time analysis is possible in strongly polynomial time with harmonic tasks, while in the general case it has pseudo-polynomial complexity and is known to be NP-hard [7]. We have proposed a new algorithm that calculates the exact *worst-case response time* of a task in linear time  $\mathcal{O}(n)$  when the higher-priority tasks are ordered by non-increasing periods and in  $\mathcal{O}(n \log n)$  in general.

### References

- [1] N. Audsley, A. Burns, M. Richardson, K.W. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.
- [2] S.K. Baruah and J. Goossens. Scheduling real-time tasks: Algorithms and complexity. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapter 28, CRC Press, 2003.
- [3] E. Bini, T.H.C. Nguyen, P. Richard, and S.K. Baruah. A response-time bound in fixed-priority scheduling with arbitrary deadlines. *IEEE Transactions on Computers*, 58(2):279–286, 2009.
- [4] E. Bini, A. Parri, and G. Dossena. A quadratic-time response time upper bound with a tightness property. In *RTSS 2015*, pages 13–22, 2015.
- [5] V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese. Polynomial-time exact schedulability tests for harmonic real-time tasks. In *RTSS 2013*, pages 236–245, 2013.
- [6] R.I. Davis and A. Burns. Response time upper bounds for fixed priority real-time systems. In *RTSS 2008*, pages 407–418, 2008.
- [7] F. Eisenbrand and Th. Rothvoß. Static-priority real-time scheduling: Response time computation is np-hard. In *RTSS 2008*, pages 397–406, 2008.
- [8] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

- [9] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *RTSS 1990*, pages 201–209, 1990.
- [10] T.H.C. Nguyen, W. Grass, and K. Jansen. Exact polynomial time algorithm for the response time analysis of harmonic tasks with constrained release jitter. in *CoRR 2019*, [arxiv.org/abs/1912.01161](https://arxiv.org/abs/1912.01161).
- [11] T.H.C. Nguyen, W. Grass, and K. Jansen. Exact polynomial time algorithm for the response time analysis of harmonic tasks. to appear in: *IWOCA 2022*.
- [12] K.W. Tindell, A. Burns, and A.J. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.
- [13] Y. Xu, A. Cervin, and K.E. Arzén. *LQG-based scheduling and control co-design using harmonic task periods*. Technical Reports TFRT-7646. Department of Automatic Control, Lund Institute of Technology, Lund University, 2016.

# The Periodic Lock Scheduling Problem

Julian Golak (Speaker) \*    Alexander Grigoriev †    Freija van Lent ‡  
Tom van der Zanden §

---

## 1 Problem statement and related literature

Transportation accounts for a quarter of the EU's greenhouse gas emissions and it is still growing. According to the European Commission [5], it is necessary to accelerate the shift to sustainable and smart mobility. They report that a 90% reduction in transport emissions is needed by 2050 to achieve climate neutrality. Their priority is to shift a substantial part of the 75% of inland freight carried today by road onto rail and inland waterways. This shift will require better management and increased capacity of railways and inland waterways.

In this research, we aim to automate the lock process on inland waterways. Currently, the lock operator processes the vessels in an ad-hoc manner. An automation of this process would reduce human error and would make the efficiency of the lock less dependent on the experience and skill of the lock operator. This research project is motivated by our close collaboration with *Trapps Wise*, a Dutch company that developed smart technology for sustainable transportation.

To our knowledge, the combination of periodicity and lock scheduling has not been addressed in literature. However, several publications have been made on the periodic maintenance problem. In [1], the authors discuss the free periodic maintenance problem (FPMP), a variant of the periodic maintenance problem where the cycle length  $T$  is a decision variable, for  $m$  machines without servicing costs. The cost of operating a machine at a given time linearly depends on the time since its last maintenance. They prove that there exists an optimal schedule that is cyclic and provides an exact, network-flow based algorithm with exponential complexity. Additionally, they introduce an approximation algorithm which even provides an optimal solution for the 2 machine setting. The authors of [2] formulate a generalization of this problem, showing that the feasibility problem

---

\*[julian.golak@tuhh.de](mailto:julian.golak@tuhh.de). Institute for Algorithms and Complexity, Hamburg University of Technology, Blohmstr. 15, 21079 Hamburg, Germany

†[a.grigoriev@maastrichtuniversity.nl](mailto:a.grigoriev@maastrichtuniversity.nl). Department of Data Analytics and Digitalisation, Maastricht University, P.O.Box 616, 6200 MD Maastricht, The Netherlands

‡[f.vanlent@maastrichtuniversity.nl](mailto:f.vanlent@maastrichtuniversity.nl). Department of Data Analytics and Digitalisation, Maastricht University, P.O.Box 616, 6200 MD Maastricht, The Netherlands

§[t.vanderzanden@maastrichtuniversity.nl](mailto:t.vanderzanden@maastrichtuniversity.nl). Department of Data Analytics and Digitalisation, Maastricht University, P.O.Box 616, 6200 MD Maastricht, The Netherlands

is NP-hard. Additionally, they develop a near-optimal solution using non-linear programming and provide several approximation algorithms - one of which is a 1.57-approximation.

## 2 Model Definition

Consider an infinite horizon, discrete time scheduling problem of  $n$  incoming streams of vessels,  $\mathcal{I} = I_1, \dots, I_n$  on a single lock. A stream is defined by the following three parameters:

1. Periodicity of vessels of stream  $i$  arriving at the lock,  $\lambda_i \in \mathbb{N}$ ,
2. Time of first occurrence of a vessel of stream  $i$ ,  $b_i$ . Without loss of generality, we assume for all streams  $i$  that  $b_1 = 0$  and  $b_i < \lambda_i$ , and
3. The direction  $d_i$ , indicating whether vessels of stream  $i$  arrive from upstream,  $U$ , or downstream,  $D$ .

We consider a waterway with a single lock present. The processing time of the lock is the sum of the time needed to change the lock state from high to low (or vice versa) and the duration of loading and unloading the lock chamber. We assume that the time space is discretized, such that the processing time is exactly equal to one time unit. Furthermore, we assume that this lock is uncapacitated, i.e. it can level up or down any number of vessels simultaneously.

At each time period, the lock operator has three possible actions:

1.  $D$ : move the lock from downstream to upstream, while processing a batch of vessels, coming from downstream.
2.  $U$ : move the lock upstream to downstream, while processing a batch of vessels, coming from upstream.
3.  $W$ : wait at current position.

Furthermore, a *policy*  $\sigma$ , is a sequence  $\sigma = \sigma_0, \sigma_1, \dots$  where  $\sigma_t \in D, U, W$  denotes the action of the lock operator during period  $t$ . A policy is *cyclic* if it consists of repetitions of a finite sequence, i.e.  $\sigma_t = \sigma_{t+T}$  for some  $T$  and all  $t$ . In this work, we aim to find policies that minimize the long-run average waiting time of the arriving vessels.

## 3 Research Contribution

**Exact solutions.** For the general problem, the encoding length is  $O(n \log(\max \lambda_i))$ . Given the short input, we expect a small output from a complexity point of view. When restricting the problem to only two streams, we propose a closed-form formula to create a feasible and optimal cyclic policy. In

the general case, we first prove that the length of the cycle with underlying optimal policies is bounded by a constant  $T = O(\text{lcm}(\lambda_i))$ . This allows us to propose a dynamic programming solution on this solution space. The complexity of the dynamic programming solution is  $O(T)$  and therefore it is not polynomial in the input size.

**Approximate solutions.** If  $T$  is very large, it is not only intractable to compute such a schedule but even *storing* such a large schedule would be impractical. For practical purposes, it is useful to have an algorithm that, in reasonable time, computes a good sequence of actions for the immediate future. We adapt our dynamic programming algorithm to only look a "short" amount of time into the future, generating schedules that are close to optimal in a reasonable amount of time. Our algorithm yields solutions with an approximation guarantee. For any  $\epsilon > 0$ , looking  $O(n^2/\epsilon)$  time units into the future provides solutions that are  $1 + \epsilon$ -approximations of an optimal cyclic schedule. Our algorithm is thus an *incremental fully polynomial time approximation scheme (IFPTAS)*. The name is based on the notion of incremental polynomial time algorithms, which means that the next set in the list of output sets is generated in time that is polynomial in the size of the input plus the size of the already generated part of the output. Such algorithms have been applied in enumeration and high multiplicity scheduling, see for example [3] and [4].

**Computational experiments and future research.** Our solution concepts are based on periodic arrival times that do not completely resemble reality. Next to the solutions concepts, we are also implementing heuristics for our problem. These are then tested against real arrival times that are extracted from a real life AIS data set provided by our industrial partner.

For future research, we are considering an algorithm that constructs input instance for the periodic lock scheduling problem. Given a real arrival of vessels (and a number of streams), the problem is then to compute the optimal periodicity, optimal shift, and an assignment of vessels to these streams such that the total error is minimized.

## References

- [1] ANILY, SHOSHANA, CELIA A. GLASS, AND REFAEL HASSIN. "The scheduling of maintenance service." *Discrete Applied Mathematics* 82.1-3 (1998): 27-42.
- [2] BAR-NOY, AMOTZ, ET AL. "Minimizing service and operation costs of periodic scheduling." *Mathematics of Operations Research* 27.3 (2002): 518-544.
- [3] BRAUNER, NADIA, ET AL. "Multiplicity and complexity issues in contemporary production scheduling." *Statistica Neerlandica* 61.1 (2007): 75-91.

- [4] GOLOVACH, PETR A., ET AL. "An incremental polynomial time algorithm to enumerate all minimal edge dominating sets." *Algorithmica* 72.3 (2015): 836-859.
  
- [5] THE EUROPEAN COMMISSION. (2020). *The European Green Deal*. [https://eur-lex.europa.eu/resource.html?uri=cellar:b828d165-1c22-11ea-8c1f-01aa75ed71a1.0002.02/DOC\\_1&format=PDF](https://eur-lex.europa.eu/resource.html?uri=cellar:b828d165-1c22-11ea-8c1f-01aa75ed71a1.0002.02/DOC_1&format=PDF)

# Makespan minimization in tree data gathering networks with memory limits

Joanna Berlińska \*

---

## 1 Introduction

A data gathering network is a computer system comprising a set of worker nodes and a base station. The workers obtain or produce datasets which have to be transferred to the base station. Data gathering wireless sensor networks find applications in military surveillance, environment monitoring and healthcare. A distributed system, where the computation results obtained by many workers have to be passed to a single server, is also a data gathering network.

In a star data gathering network, the workers communicate directly with the base station. Contrarily, in a tree network, data are relayed through a set of intermediate nodes. Scheduling algorithms were designed for optimizing data gathering in several types of star networks, such as, e g., networks with data compression [1, 5, 6], with variable communication speed [3] and with limited base station memory [2]. Tree data gathering networks have attracted less attention so far. Minimizing the maximum lateness in a tree network with dataset release times and due dates was studied in [4].

This work considers scheduling in a tree data gathering network with limited memory. Each of the worker nodes holds a dataset that has to be sent to an appropriate intermediate node. The intermediate node processes the dataset and then sends it to the base station. A dataset occupies the intermediate node's memory buffer from the moment when it starts being received until the time when its transfer to the base station completes. The total size of datasets coexisting in the memory of an intermediate node can never exceed its buffer size. Our goal is to organize dataset transfers and processing so as to minimize the total time required to gather the data.

## 2 Problem formulation

The data gathering network consists of  $n$  worker nodes,  $m$  intermediate nodes and a single base station. An intermediate node  $P_i$ , where  $1 \leq i \leq m$ , gathers data

---

\* [Joanna.Berlinska@amu.edu.pl](mailto:Joanna.Berlinska@amu.edu.pl). Faculty of Mathematics and Computer Science, Adam Mickiewicz University, Poznań, Uniwersytetu Poznańskiego 4, 61-614 Poznań, Poland

from  $n_i$  workers  $P_{ij}$ , where  $1 \leq j \leq n_i$ . Thus,  $n_1 + \dots + n_m = n$ . A worker  $P_{ij}$  holds a dataset  $D_{ij}$  of size  $\alpha_{ij}$ , which has to be sent to the intermediate node  $P_i$  for processing. The processed dataset is then passed by  $P_i$  to the base station. Sending dataset  $D_{ij}$  from  $P_{ij}$  to  $P_i$  takes time  $c_1\alpha_{ij}$ , and processing this dataset by  $P_i$  requires time  $a\alpha_{ij}$ . The processed dataset  $D_{ij}$  is sent to the base station in time  $c_2\alpha_{ij}$ . Preemptions in communication or computation are not allowed. Each node, including the base station, can receive at most one dataset at a time. An intermediate node can process at most one dataset at a time, but it can receive one dataset, process another dataset, and send yet another dataset simultaneously. Each intermediate node processes and sends datasets in the order in which it receives them. The size of the memory buffer of node  $P_i$  is  $B_i \geq \max_{j=1}^{n_i} \alpha_{ij}$ . At the moment when a dataset  $D_{ij}$  starts being sent to  $P_i$ , a memory block of size  $\alpha_{ij}$  is allocated at  $P_i$ . The block is released when the transfer of the processed dataset  $D_{ij}$  to the base station is completed. The total size of memory blocks allocated at  $P_i$  cannot at any time exceed  $B_i$ . The scheduling problem is to minimize the makespan, i.e., the time by which all processed datasets arrive at the base station.

### 3 Results

The analyzed problem generalizes makespan minimization in a star data gathering network with limited base station memory [2], and hence, it is strongly NP-hard. We first formulate it as an integer linear program (ILP). The program contains  $O(n)$  rational variables,  $O(n^3)$  binary variables and  $O(n^3)$  constraints, and in consequence, it cannot be used in practice for solving even moderate size instances. Therefore, we also propose heuristic algorithms.

First, we design simple heuristics running in  $O(n^2)$  time. Each of these algorithms is defined by two rules. The first one determines the order in which the datasets are transferred to each intermediate node  $P_i$ , and the second one defines the order in which the datasets are passed to the base station. We consider the following rules for sending datasets to intermediate nodes.

- Inc: send the datasets in the order of non-decreasing sizes  $\alpha_{ij}$ .
- LF: always choose the largest dataset which fits in the memory currently available at the intermediate node.
- Rnd: send the datasets in a random order.

Since the datasets are sent to the base station by an intermediate node  $P_i$  in the order in which they were received and processed, a rule designed for sending the datasets to the base station only has to choose the order of the datasets received from different intermediate nodes. We analyze two scheduling methods for this stage.

- FIFO: transfer the datasets in the order in which their processing at the intermediate nodes completed.

- B: choose the dataset from the intermediate node which has the smallest currently available memory.

An algorithm that uses Rule1 for sending the datasets to the intermediate nodes, and Rule2 for transferring them to the base station, is denoted by Rule1-Rule2.

Furthermore, we design a variable neighborhood search algorithm VNS. In this algorithm, a schedule is represented by an array  $x[1..n]$  of dataset priorities, where a smaller number means a higher priority. The priority of dataset  $D_{ij}$  is  $x[\sum_{k=1}^{i-1} n_k + j]$ . In order to compute the makespan, the schedule is constructed using the defined priorities both for sending the datasets to the intermediate nodes, and for transferring them to the base station. Variable neighborhood search consists in systematically changing the neighborhoods used during local search. Suppose a sequence of neighborhoods  $N_1, \dots, N_{k_{max}}$  are defined. The algorithm starts with the current neighborhood number  $k = 1$  and a given initial solution  $x$ . In each step, the best solution  $x'$  in neighborhood  $N_k(x)$  is found. If  $x'$  is better than  $x$ , then  $x$  is changed to  $x'$  and  $k$  is set to 1. If  $x'$  is not better than  $x$ , then  $k$  is increased by 1. The search continues until  $k$  exceeds  $k_{max}$ . In our algorithm, the initial solution is  $x = [1, \dots, n]$ , and the following three neighborhoods are used.

- $N_1(x)$  contains all arrays obtained from  $x$  by reversing any subarray  $x[i..j]$ .
- $N_2(x)$  contains all arrays obtained from  $x$  by swapping a pair of values  $x[i]$  and  $x[j]$ .
- $N_3(x)$  contains all arrays obtained from  $x$  by moving a value  $x[i]$  to an arbitrary position  $j \neq i$  in the array.

The proposed algorithms were implemented in C++ and tested in a series of computational experiments on randomly generated instances. The obtained results lead us to the following conclusions.

- The computational cost of ILP is very high even for small instances.
- The B rule for scheduling the transfers to the base station is counterproductive. Much better results are obtained by the FIFO rule.
- Among the simple heuristics, the best solutions are usually produced by LF-FIFO. Still, if  $c_2$ ,  $\delta_B$  or  $m$  is very large, or  $c_1$  or  $a$  is very small, then Inc-FIFO obtains better schedules.
- VNS significantly outperforms the simple heuristics. It delivers very good results in a short time.

**Acknowledgement.** This research was partially supported by the National Science Centre, Poland, grant 2016/23/D/ST6/00410.

## References

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

- [2] J. BERLIŃSKA (2020). Heuristics for scheduling data gathering with limited base station memory. *Annals of Operations Research* 285, 149-159.
- [3] J. BERLIŃSKA (2020). Scheduling in data gathering networks with background communications. *Journal of Scheduling* 23, 681-691.
- [4] J. BERLIŃSKA (2022). A comparison of priority rules for minimizing the maximum lateness in tree data gathering networks. *Engineering Optimization* 54, 218-231.
- [5] W. LUO, B. GU, AND G. LIN (2018). Communication scheduling in data gathering networks of heterogeneous sensors with data compression: Algorithms and empirical experiments. *European Journal of Operational Research* 271, 462-473.
- [6] W. LUO, Y. XU, B. GU, W. TONG, R. GOEBEL AND G. LIN (2018). Algorithms for Communication Scheduling in Data Gathering Network with Data Compression. *Algorithmica* 80, 3158-3176.

## Static schedules for fault-tolerant transmission

Kunal Agrawal (Speaker) \*      Sanjoy Baruah †      Alan Burns ‡  
Jeremy Fineman §      Zhe Wang ¶

---

Consider the following data transmission problem [1]. We have a communication medium (such as a wireless network, a shared bus, CAN, etc.) that is shared amongst several entities (“nodes”) seeking to communicate messages amongst themselves. Each message is characterized by a source node and a destination node, and takes the same duration – a “slot”. We assume that the local clocks in all the nodes are adequately synchronized such that each node knows when each slot begins and ends. The set of messages that is to be transmitted is known beforehand. We wish to pre-compute a static fault-tolerant schedule denoting which message is to be transmitted during which slot, and distribute this schedule to all the nodes beforehand. During run-time, each message will only be transmitted in a slot to which it has been assigned in this schedule. We assume that a failure of such a transmission may arise from either or both of two factors:

1. A *collision* – multiple messages are transmitted during the slot. All the messages that are transmitted during the slot are lost when such a collision occurs.
2. A *transmission error* – this is some (external) fault in the transmission medium. If such an error occurs then even a single (and hence non-colliding) message sent during the slot will not be successfully received, and must therefore be retransmitted.

We assume that at the end of each slot all the nodes can determine (by monitoring the communication medium during the slot interval) whether a successful transmission has occurred during that slot or not, and not retransmit any successfully-transmitted message. Given  $n$  messages and a fault-tolerance parameter  $f \in \mathbb{N}$ , we desire to generate a schedule of minimum length that guarantees the successful transmission of all  $n$  messages in the presence of up to  $f$  transmission errors.

**Results [5].** It is clear that each message must be assigned to at least  $f + 1$  slots – if any message  $m$  is assigned to fewer slots, the  $f$  medium errors can occur on all the slots where  $m$  is assigned and  $m$  will fail to transmit. The simplest schedule is

---

\*kunal@wustl.edu. Washington University in Saint Louis, USA.

†baruah@wustl.edu. Washington University in Saint Louis, USA.

‡alan.burns@york.ac.uk. The University of York, UK.

§jff474@georgetown.edu. Georgetown University, USA.

¶zhe.wang@wustl.edu. Washington University in Saint Louis, USA.

to send each message  $(f + 1)$ , times which uses  $(n + nf)$  slots and the overhead is  $nf$ . This is the best we can do if we assign at most one message to each slot in the static schedule. One can take advantage of the fact that the sender knows when a message has transmitted successfully and will not transmit such a message again to generate shorter schedules. An algorithm was obtained in [1] for generating fault-tolerant static schedules of length  $(n + nf/2)$  slots — this algorithm has an overhead of  $nf/2$ . This algorithm was significantly improved in [5], which contains three major results:

1. An algorithm for designing fault-tolerant static schedules of length  $n + O(f^2 \log^2 n)$
2. Proof of the existence of fault-tolerant static schedules of length  $n + O(f \log f \log n)$
3. The above forms the basis of a randomized algorithm for generating a fault-tolerant static schedule of length  $n + O(f \log f \log n)$  with non-zero probability. Although the algorithm for generating the schedule is efficient, verifying that the schedule is fault tolerant takes time exponential in  $f$ . Therefore, this may not be a viable approach for large  $f$ , but can be practical for small  $f$  since the schedule need be generated just once prior to run-time.

**Future directions.** There are many directions for future work. First, we do not know of any non-trivial lower bounds for the problem. Both the efficient algorithms mentioned above are based on the idea of  $\alpha$ -good mappings [2] or selectors [3]. This problem is also related to cover-free families [4], which are extensively studied. However, the upper and lower bounds from these areas do not directly apply to the fault-tolerant scheduling problem. Second, as mentioned above, the schedule with  $O(f \log f \log n)$  overhead does not have a fast constructive algorithm and is not practical for large values of  $f$  and  $n$ . We would like to find constructive algorithms for this schedule. Finally, a more practical fault model would allow, for instance,  $f$  faults every  $t$  time steps, instead of  $f$  faults regardless of schedule length. While the above algorithms can be used in this model, they generate pessimistic schedules. We would like to design algorithms specifically for these practical fault models.

## References

- [1] Kunal Agrawal, Sanjoy Baruah, and Alan Burns. Fault-tolerant transmission of messages of differing criticalities across a shared communication medium. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, RTNS '19, pages 41–49, New York, NY, USA, 2019. Association for Computing Machinery.

- [2] Michael A Bender, Jeremy T Fineman, and Seth Gilbert. Contention resolution with heterogeneous job sizes. In *European Symposium on Algorithms*, pages 112–123. Springer, 2006.
- [3] Bogdan S. Chlebus, Leszek Gkasienciec, Alan Gibbons, Andrzej Pelc, and Wojciech Rytter. Deterministic broadcasting in unknown radio networks. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, page 861–870, 2000.
- [4] P. Erdős, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of others. *Israel Journal of Mathematics*, 51(1):79–89, 1984.
- [5] Zhe Wang, Kunal Agrawal, and Jeremy T. Fineman. Sub-linear overhead in static schedules for fault-tolerant transmission. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 405–417, 2021.

# Scheduling Messages towards Detecting Monochromatic Pattern on a Bounded Degree Network

Bala Kalyanasundaram (Speaker) \*      Mahe Velauthapillai †

---

## 1 Introduction

A sensor network (SN) is a network of sensor enabled devices that communicate with their neighbors. This network is used in many application areas including environment monitoring, traffic management, and wild life monitoring. Depending on the application, a SN can consist of a few nodes to millions of nodes connected by a graph. A sensor node may have one or more sensor modules for measuring some information, for example intensity of light, temperature, humidity, pressure, or sound. In addition, each sensor node includes components to handle storage, processing, communication, and power. When activated by a simple broadcast message from a source, the network monitors the environment continuously to detect and/or react to certain predefined events or patterns.

As an example, the technology for active pixel image sensors for Complementary Metal Oxide Semiconductor (i.e, CMOS) is chosen over charge-coupled devices (i.e., CCD's). This is due to the possibility of on-chip signal processing for rapid image analysis for applications such as particle tracking in Physics (see [1]). Recent development on highly sensitive active pixel image can be found in [2]. Instead of offloading each image consisting of pixel values, we want each pixel in the SN to act as a mini-processor capable of performing communication with its neighbors and perform active pattern detection and only offload the image if a pattern appears. A pattern in this context is an image specified as a  $(2k + 1)$  by  $(2k + 1)$  matrix.

For simplicity, let us consider a grid network with each node communicating to its eight neighbors. We consider a synchronous network and at each time step, a sensor detects information at its location and then exchanges messages with its immediate neighbors. Let us assume that the sensor detects a 0 or a 1 at its location at each time step. Based on what the sensor detects at its location, if a pattern of pixel values of size  $2k + 1$  by  $2k + 1$  appears at time  $t$  then the sensor at the center of the pattern must detect it no later than time  $t + O(k)$ . This is achieved by devising a protocol to exchange messages between neighboring nodes.

---

\*kalyan@cs.georgetown.edu. Dept of Computer Science, Georgetown University, USA.

†mahe@cs.georgetown.edu. Dept of Computer Science, Georgetown University, USA.

The detection process is not a one-time process but it is an ongoing process. Every occurrence of the pattern (time and space-wise) must be detected by the designated nodes. In 2015 MAPSP ([3]), we considered the grid network and presented many results relevant to the grid network.

As a logical next step, we want to extend the pattern detection problem to any arbitrary graph where each sensor node detects a 0 or a 1 at each time unit. After the sensor detection process at each time unit, we assume that the sensor node communicates some composite information with its neighbors and this process repeats every time unit. In this paper, we consider the problem of checking if all neighbors at a distance  $k$  or less at each time step have the same sensor value (say 0) or not. This is often referred to as monochromatic pattern of size  $k$ . We want every sensor node to detect monochromatic pattern at each time step. In other words, this problem is equivalent to the following problem. Given a graph of sensor nodes, each sensor node must detect if there is a 1 within a distance  $k$  from the node.

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

For now, we assume that each sensor node detects a bit at each time step. In this paper, we consider only monochromatic pattern of size  $k$ . We consider the tree network. While a line-network is a tree [3], its protocol fails for a tree since the tree has many branches. However, if we allow the protocol to differ for each node then we can solve the problem in  $O(\log k)$ .

**Theorem 1** *There is a protocol to detect a monochromatic pattern of size  $k$  (say all 0) on a tree with communication complexity  $O(\log k)$ .*

Each sensor node gets two numbers in the range  $[0, k]$ . One number is designated for *UP* sweep of the tree and the other number is designated for *DOWN* sweep of the tree. Any node can be identified as a root. Once the root is identified then we set the  $DOWN(root)$  to be  $i$  where  $i = -\alpha \bmod (k + 1)$  where  $\alpha$  is the distance to the furthest node from the root. Starting from the root, we go one level down and assign  $DOWN(x) = (i + 1) \bmod (k + 1)$  for all  $x$  reachable directly from the root. In this way, if a node  $y$  is distance  $\ell$  from the root then assign  $DOWN(y) = (-\alpha + \ell) \bmod (k + 1)$  to the node  $y$ .

Let  $z$  be the farthest node from the root. We now set  $DOWN(z) = UP(z) = 0$ . In fact  $z$  cannot communicate with nodes below it since there are none. For all other nodes  $w$ , we set  $UP(w) = -DOWN(w) \bmod (k + 1)$ .

The protocol for detecting monochromatic pattern is given below.

**Protocol For Tree:**

1. Assign  $UP(\cdot)$  and  $DOWN(\cdot)$  numbers for each node.
2. At time  $t$ , each node  $x$  gets its sensor value. The node then transmits the information it has gathered about time  $t - UP(x)$ ,  $t - (k + 1) - UP(x)$ ,  $t - 3(k +$

$1) - DOWN(x)$  and  $t - 4(k + 1) - DOWN(x)$ . For each of the four time units, it sends information only about furthest going 1.

We will give a high level description of the proof. In the *UP*-sweep, we will argue that the node  $x$  learns about the information about how far a given 1 has travelled from below. We will argue that the node keeps the one that travelled the least and discards everything else. It is interesting to note that after *UP*-sweep, every node knows the status of 1's from below. However, it does not know if there is a 1 that is at most distance  $k$  from above. This information is conveyed when we do the *DOWN*-sweep after the *UP*-sweep. When we put both *UP*- and *DOWN*-sweeps we get the answer. In summary, any node will send 4 information about any particular time  $t$  to its neighbors. Each time, it sends the farthest going 1, if any. This information takes  $O(\log k)$ . Hence the bound.

We now explore how we could extend this simple idea to include other graphs that have cycles in it. One such graph is what we call a "tree of cycles." Imagine a tree where each node of the tree is a super-node. When we expand the super-node, it yields a simple cycle. When a super-node has degree  $x$ , the cycle that corresponds to the super-node has  $x$  branches and the branches can happen at any of the nodes of the cycle. The resulting graph is called a planar cycles.

**Theorem 2** *There is a protocol to detect a monochromatic pattern of size  $k$  on a tree of cycles with communication complexity  $O(\log k)$ .*

If a cycle is not a simple cycle but a planar graph with the following limitations. Assume that there are  $r$  nodes in a cycle and they form the outer part of the cycle. These  $r$  nodes are also connected by edges to form a planar graph inside the circle. But there are no nodes inside the circle. We call such a graph, a planar cycle. We can easily extend the result to any tree of planar cycles. The crux of the argument is that the planar cycle can be simulated by another tree of cycles.

**Theorem 3** *There is a protocol to detect a monochromatic pattern of size  $k$  on a tree of planar cycles with communication complexity  $O(\log k)$ .*

Our ultimate goal is to show that monochromatic pattern of size  $k$  has communication complexity  $O(\log k)$  for a bounded degree network.

Finally, we will argue why detecting a monochromatic pattern on a bounded degree is indeed hard. One could do a Breadth First Search, BFS for short, and create a tree. One could then apply *UP* and *DOWN* as we did for a tree. But unfortunately the BFS will have edges between nodes at the same level or easily get to the same level by either going one level up or down. This alternating path between a level and adjacent level below will be a difficult problem to solve. This shows that one has to argue in an asynchronous fashion that the correct information gets to all of the nodes within the time limit of  $O(k)$ .

**Open Problem:** Is there a protocol with communication complexity  $O(\log k)$  to detect a monochromatic pattern of size  $k$  on a bounded degree network? Can we at least show that the communication complexity is  $O(\log k)$  for a bounded degree planar graph?

## References

- [1] GÖTZ GAYCKEN ET AL. *Monolithic active pixel sensors for fast and high resolution vertex detectors*, *Nuclear Instruments & Methods in Physics Research Section A-accelerators Spectrometers Detectors and Associated Equipment*, 2006, Vol. 560, pp. 44-48.
- [2] S. HONG ET AL. *Highly sensitive active pixel image sensor array driven by large-area bilayer MoS<sub>2</sub> transistor circuitry*, *Nature Communications*, Article Number 3559 (2021).
- [3] B. KALYANASUNDARAM AND M. VELAUTHAPILLAI. *Scheduling Messages To Detect Patterns Continuously On A Grid Sensor Network*, *MAPSP 2015*.

# Data-driven scheduling in serverless computing to reduce response time \*

Bartłomiej Przybylski <sup>†</sup>   Paweł Żuk (Speaker) <sup>‡</sup>   Krzysztof Rządca <sup>§</sup>

---

## 1 Introduction

Cloud computing covers a wide landscape of approaches, starting from IaaS, through PaaS, to FaaS. In IaaS (Infrastructure-as-a-Service), customers deploy complete virtual machines. In PaaS (Platform-as-a-Service), they deploy scalable applications running in environments managed by the cloud provider. Eventually, in FaaS (Function-as-a-Service), they deploy functions [1]. FaaS products are offered by all major cloud providers, including Amazon, Google and Microsoft. One can also maintain FaaS services on premises with Apache OpenWhisk, a standard open-source solution.

In FaaS, the *cloud customer* deploys a function (e.g. a snippet of Python code) to the FaaS cloud *provider*. Once deployed, the function may be invoked, by, e.g., an HTTP request. Each invocation received by the cloud provider is passed to a FaaS controller which acts as a load balancer. Thus, the controller sends the invocation to a selected worker *node* (a physical or virtual machine). Finally, the worker node executes the function and returns the result. To use hardware efficiently, a single node usually hosts many functions and executes many invocations in parallel, using containers.

We take a cloud provider’s view on optimization of FaaS serving: given a particular workload (a stream of end-user requests), we want to serve those requests (invocations) as efficiently as possible. There are two main, orthogonal perspectives to take here. First, on the level of a FaaS cluster, one can manipulate the way invocations are assigned to worker nodes. Second, on the level of a single node, the performance can be improved by applying different container- and invocation-management methods. We take the second approach. Our method changes the scheduling of individual invocations on the node’s processors.

---

\*This abstract is based on our paper: B. Przybylski, P. Żuk, and K. Rządca, “Data-driven scheduling in serverless computing to reduce response time” in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 206–216.

<sup>†</sup>bap@mimuw.edu.pl. Institute of Informatics, University of Warsaw, Warsaw, Poland

<sup>‡</sup>p.zuk@mimuw.edu.pl. Institute of Informatics, University of Warsaw, Warsaw, Poland

<sup>§</sup>krzadca@mimuw.edu.pl. Institute of Informatics, University of Warsaw, Warsaw, Poland

In FaaS, each function is usually executed multiple (hundreds to millions) of times. Having this in mind, we gather information that can be used to make better decisions in the future: call frequencies and execution times for each of the hosted functions. Then, we adapt some theoretically-grounded heuristics like SEPT or SERPT, and we propose a number of new ones. We evaluate these strategies using Azure Functions Trace [2], a dataset on FaaS usage, in our simulations. Compared to standard FIFO or Round-Robin, our data-driven scheduling strategies significantly improve the performance of the node.

## 2 Model, strategies and objectives

We consider a set of functions that are already reachable, i.e. loaded into containers stored in the memory of the node. In our model, the node makes online decisions on how to assign incoming invocations to parallel processors. In contrast to standard approaches for interactive systems, we take advantage of the fact that in FaaS each function is usually invoked multiple times. Thus, the local scheduler is able to *learn* and make better decisions when subsequent invocations of the same functions appear. We estimate the expected execution time of the function call based on just a few previous invocations (our experiments show that when considering more than 10 invocations the gains in response time are negligible). Additionally, we count the number of invocations of each function. Then, given a queue of invocations (each invoking a particular function) ready to be executed (and — in preemptive variants — additionally considering currently executing invocations), we prioritize them based on one of the following strategies:

- **FC#** (*Fair Choice*) — the priority is based on the expected number of invocations of the same function in the nearest future and the actual number of invocations in the nearest past;
- **FCP** (*Fair Choice*) — the priority is based on the total execution time of invocations of the same function in the nearest future and the actual total actual execution time in the nearest past;
- **SERPT** (*Shortest Expected Remaining Processing Time*) — the priority is based on the estimated processing time of the invocation and its current execution time;
- **SEPT** (*Shortest Expected Processing Time*) — the priority is based on the estimated processing time;

In addition to standard performance metrics like the *average flow time* (AF), the *average stretch* (AS), the *99th percentile of flow time* (F99), and the *99th percentile of stretch* (S99), we also propose two new objectives specifically for the FaaS environment: the **average function-aggregated flow time** (FF) and the **average function-aggregated stretch** (FS). Our goal is to measure how the

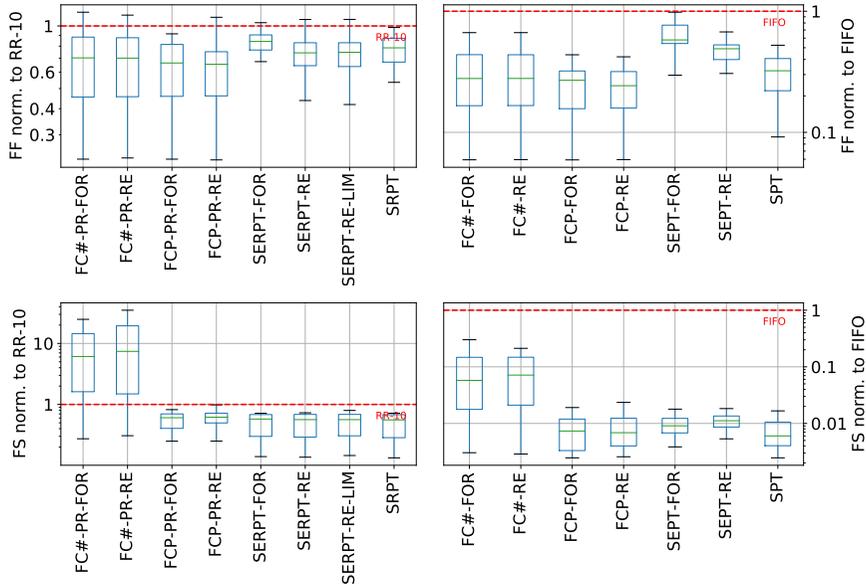


Figure 1: The performance of different strategies for preemptive (left) and non-preemptive (right) cases. Each box shows statistics over 20 independent instances. Each instance is prepared for 20 processors, 30-minute time frame and 90% load. The objective is to minimize the function-aggregated flow time (top) and stretch (bottom). All the results are normalized to Round-Robin with a 10 ms window (left) and FIFO (right). SRPT and SPT are clairvoyant (and we use them as a reference).

response time (or stretch) for a specific function is affected by the calls of other functions. These two metrics take into account that resources of a single node are shared between functions deployed by multiple users — thus, scheduling needs to be fair not only to individual invocations, but also to individual functions.

Using the extended three-field Brucker’s notation, we denote the theoretical variants of the considered problems as  $Pm|on-line, r_i, p_i \sim \mathbb{P}_{f(i)}|\mathbb{E}[\sigma]$  and  $Pm|on-line, pmtn, r_i, p_i \sim \mathbb{P}_{f(i)}|\mathbb{E}[\sigma]$  where  $f(i)$  is the function related to  $i$ -th invocation, and  $\sigma \in \{AF, AS, F99, S99, FF, FS\}$ .

### 3 Simulation results

Our simulations are based on the real-world data from Azure Function Trace. We vary the number of processors on a node (10, 20, 50 or 100), their average load (70%, 80%, 90% or 100%) and the duration of the time frame (30, 60 or 100 min). For each configuration, we generate 20 independent instances. Each instance consists of functions selected randomly from the Azure Function Trace, for which we reproduce the sequence of invocations following the trace data. We compare our strategies against the standard FIFO and Round-Robin, for non-preemptive

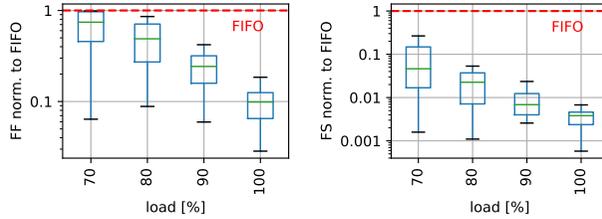


Figure 2: The performance of the FCP strategy in a non-preemptive case, depending on the average load. Each box shows statistics over 20 independent instances. Each instance is prepared for 20 processors and a 30-minute time frame. The objective is to minimize the function-aggregated flow time (left) and stretch (right). All the results are normalized to FIFO.

and preemptive cases, respectively. Example results of such comparison, for 20 processors, 90% load, 30-minute time frame, and average function-aggregated flow time and stretch are presented in Fig. 1.

In general, our SERPT and SEPT implementations show significant improvement over the baselines. For preemptive approaches, our SERPT strategies are close to the clairvoyant SRPT. This suggests that estimations based on just 10 previous invocations are very close to actual processing times. Moreover, compared with the baseline Round-Robin, these strategies reduce the average stretch by a factor of 2, and the 99th percentile of stretch by over an order of magnitude. For non-preemptive approaches, all our strategies are significantly better than the baseline FIFO. Our SEPT variants show improvements of over two orders of magnitude when it comes to all stretch-related metrics. When measuring the average function-aggregated flow-time, our FC#/FCP strategies have the best performance. Additionally, their relative performance, compared to the FIFO baseline, increases with the increase of load, as shown in Fig. 2.

## Acknowledgements

This research is supported by a Polish National Science Center grant Opus (UMO-2017/25/B/ST6/00116).

## References

- [1] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, “The rise of serverless computing”, *Commun. ACM*, vol. 62, no. 12, 2019.
- [2] M. Shahradd, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider”, in *USENIX ATC*. USENIX Association, 2020.

# Automatic HBM Management: Models and Algorithms

Daniel DeLayo (Speaker) \*    Kenny Zhang \*    Kunal Agrawal †  
Michael A. Bender \*    Jonathan W. Berry ‡    Rathish Das §  
Ben Moseley ¶    Cynthia A. Phillips ||

---

## 1 Introduction

Some past and future supercomputer nodes incorporate *high-bandwidth memory* or *HBM*. Compared to standard DRAM, HBM has similar latency, higher bandwidth and lower capacity.<sup>1</sup> HBM’s bandwidth is so high because it is placed directly onto the processor package (unlike DRAM). HBM thus augments the existing memory hierarchy by providing a memory that can be accessed with up to 5x higher bandwidth than DDR4 when feeding a CPU [1], and up to 20x higher bandwidth when feeding a GPU [7].

HBM is not a replacement for DRAM since it is generally about five times smaller than DRAM.<sup>2</sup> Moreover, HBM does not fit into the standard cache hierarchy because its latency is no better than DRAM’s. (With standard “pyramid-shaped” hierarchies, the bandwidth and latency get better as the sizes get smaller.)

In this paper, we evaluate algorithms for managing High-Bandwidth Memory automatically. Previous work suggests that, in the worst case, performance is extremely sensitive to the policy for managing the channel to DRAM. Prior theory shows that a priority-based scheme (where there is a static strict priority-order among  $p$  threads for channel access) is  $O(1)$ -competitive, but FIFO is not, and in the worst case is  $\Omega(p)$  competitive.

It is not known how well the asymptotically good algorithms introduced by Das et al. [2] work on real workloads. It also has been unknown how well the theoretical abstraction accurately predicts empirical performance. Following this

---

\*{ddelayo,kzzhang,bender}@cs.stonybrook.edu. Computer Science, Stony Brook University.

†kunal@wustl.edu. Computer Science and Engineering, Washington University in St Louis.

‡jberry@sandia.gov. Discrete Math & Optimization, Sandia National Laboratories.

§rathish.das@uwaterloo.ca. Computer Science, University of Waterloo.

¶moseleyb@andrew.cmu.edu. Tepper School of Business, Carnegie Mellon University.

||caphill@sandia.gov. Computing Research, Sandia National Laboratories.

<sup>1</sup>Hardware vendors use various brand names such as High-Bandwidth Memory (HBM), Hybrid Memory Cube (HMC), and MCDRAM for this technology.

<sup>2</sup>This is due to constraints such as heat dissipation, as well as economic factors.

theoretical guidance would be a disruptive change for vendors, who currently use FIFO variants in their DRAM-controller hardware. Our goal is to determine theoretically and empirically whether we can justify recommending investment in priority-based DRAM controller hardware.

In order to experiment with DRAM channel protocols, we chose a theoretical model, validated it against real hardware, and implemented a basic simulator. We corroborated the previous theoretical results for the model, conducted a parameter sweep while running our simulator on address traces from memory bandwidth-bound codes (GNU sort and TACO sparse matrix-vector product), and designed better algorithms.

**Central problem of HBM management.** Given this lack of consensus, machines that use HBM, such as Intel’s Knights Landing [6], boot in multiple modes specifying how much of the HBM is system managed and how much is user managed. In *cache mode* the system controls HBM as a last level of cache, in *flat mode* the programmer explicitly copies data in and out of HBM, and in *hybrid mode* the HBM is split into a “flat” piece and a “cache” piece. In Intel’s Sapphire Rapids, there is also *HBM-only* mode, for systems without DRAM [4].

**HBM+DRAM model.** In HBM, the channel(s) to DRAM are a sequential bottleneck [2]. The  $p$  cores can simultaneously send a memory request in each time step. Up to  $p$  requests can be fulfilled by the HBM in parallel, but only one request (or a small number of requests) at a time can use the channel to DRAM. Thus, if multiple cores simultaneously request memory from DRAM, the channel accesses need to be *scheduled* and *serialized*. Waiting for access to DRAM dominates the running time over other considerations. For current systems, the number of channels is generally not more than 6.

A natural performance objective is *makespan* which, given a batch of running processes, is the time when the last process completes. Minimizing cache misses is not the same as minimizing makespan, and can be very far from it [2, 3]. In fact, a workload could have relatively few cache misses, but because it does not take advantage of the parallelism between cache and HBM, it has a poor makespan.

A system-controlled HBM has two algorithmic policies to set:

- **HBM replacement policy.** When the system brings a new block from DRAM to HBM, it first needs to decide which block to evict from HBM.
- **Queue-management policy for channel to DRAM.** Multiple requests (up to  $p$  requests: one request per core) for blocks on DRAM can occur simultaneously, but at most  $q$  requests ( $1 \leq q \ll p$ ) can be fulfilled per time step. The system must decide in each time step which of the outstanding block requests to fulfill (and which ones to keep in the queue).

The traditional way we think about replacement policies (Least Recently Used [5], or LRU) turns out also to work with HBM. That is, LRU can also

be used in the context of HBM management to help obtain theoretically good performance guarantees [2].

**FIFO queue management for DRAM channel is the problem.** This algorithmic challenge can be restated as follows: how to partition the pages of the HBM among all processes and then change this allocation dynamically in each time step. This is because if in each timestep we have control over which process's page is brought into HBM, and which process's page is ejected, then we are exactly determining the partitioning of HBM among the processes. The problem with the FIFO policy for serializing DRAM access is that it tends to have the effect of spreading out HBM evenly and thinly among all the processes.

**Priority queue management for DRAM channel.** An alternative policy called *Priority* was recently proposed by Das et al. [2]. In this scheme, each thread is assigned a fixed priority. These priorities effectively determine at each step, which thread gets to use the DRAM channel.

The Priority scheme has a good makespan because the priority scheme naturally does a good job of partitioning the HBM among the threads. If there is not enough space in the HBM to satisfy all of the threads, then the lower-priority threads starve until the higher-priority threads do not need as much as space.

## 1.1 Results

In this paper, we evaluate priority-based methods for managing High-Bandwidth Memory automatically.

**Explanation of Knight's Landings performance by the HBM+DRAM Model.** We validate that the HBM+DRAM model Knight's Landing (KNL) [6], an example system with HBM, has a performance profile explained by HBM+DRAM model. We develop and run a series of microbenchmarks on KNL in its different modes. In our measurements HBM has a similar but slower access latency than DRAM by 20ns (roughly 10 percent outside of shared L2), HBM has a higher bandwidth by about  $4.8\times$ , and accessing HBM in cache mode can incur an extra latency cost.

**Evaluation of FIFO versus Priority.** We compare FIFO and Priority's makespan when running the TACO Sparse Matrix-Matrix Multiplication and GNU sort workloads.

In our instrumented traces, at low thread counts where HBM is plentiful, Priority gives a worse makespan than FIFO by up to 37%. When the number of threads increases and HBM becomes more scarce, FIFO gives up to a  $3.3\times$  worse makespan than Priority. We also design a request sequence to be bad for FIFO, hold the amount of memory per core constant, and get a linearly worse makespan, up to  $40\times$ . Because Priority is provably good, we cannot create a bad request

sequence for it.

**Demonstration that periodically changing priorities fully eliminates FIFO’s advantage.** We propose Dynamic Priority, which retains the theoretical guarantees of Priority. Unlike Priority, Dynamic Priority is either as good as FIFO or outperforms FIFO in all of our simulations. We also find that Priority suffers from having highly variable response times, where the *response time* of a page request is the duration between sending the page to the DRAM queue and the page being served. Dynamic Priority reduces this variance by occasionally permuting the priorities. Changing priority more often decreases the standard deviation of response times but may increase the makespan. We characterize this tradeoff and, for our experiments, identify a broad range of parameters where variance is small and the makespan is as good as or better than both Priority and FIFO.

## References

- [1] High-performance on-package memory, January 2015. <http://www.micron.com/products/hybrid-memory-cube/high-performance-on-package-memory>.
- [2] Rathish Das, Kunal Agrawal, Michael A Bender, Jonathan Berry, Benjamin Moseley, and Cynthia A Phillips. How to manage high-bandwidth memory automatically. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 187–199, 2020.
- [3] Alejandro López-Ortiz and Alejandro Salinger. Paging for multi-core shared caches. In *Proc. Innovations in Theoretical Computer Science (ITCS)*, pages 113–127, 2012.
- [4] Ruchira Sasanka. Enabling high-bandwidth memory in future intel processors, 2022.
- [5] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, February 1985.
- [6] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. Knights landing: Second-generation intel xeon phi product. *Ieee micro*, 36(2):34–46, 2016.
- [7] Jack Wells, Buddy Bland, Jeff Nichols, Jim Hack, Fernanda Foertter, Gaute Hagen, Thomas Maier, Moetasim Ashfaq, Bronson Messer, and Suzanne Parete-Koon. Announcing supercomputer summit. Technical report, ORNL (Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States)), 2016.

## Paging and the Address-Translation Problem\*

Michael A. Bender<sup>†</sup>    Abhishek Bhattacharjee<sup>‡</sup>    Alex Conway<sup>§</sup>  
Martín Farach-Colton (Speaker) <sup>¶</sup>    Rob Johnson<sup>§</sup>  
Sudarsun Kannan<sup>¶</sup>    William Kuszmaul<sup>||</sup>    Nirjhar Mukherjee<sup>\*\*</sup>  
Don Porter<sup>\*\*</sup>    Guido Tagliavini<sup>¶</sup>    Janet Vorobyeva<sup>†</sup>    Evan West<sup>†</sup>

---

In the classical on-line resource augmentation problems is *paging*, in which a sequence of page requests  $p_1, p_2, \dots$  must be serviced using a memory of size  $P$  pages [12]. The cost of servicing a page request is 0 if the page is currently cached in memory. Otherwise there is a *page fault* and an *IO* must be performed, which means that the page must be fetched from disk to RAM (perhaps evicting another page) at a cost of 1.

Paging is critical to *virtual memory* systems, where programs reference pages by *virtual page addresses*. When a page is cached in memory, it also has a *physical page address* in the range  $\{1, 2, \dots, P\}$ , specifying the location where it is actually stored. Every virtual address referenced by a program must be translated to a physical address by a process called *address translation (AT)*. If a virtual page does not have a physical address, the page's data must be fetched from external storage, a physical page must be allocated (and potentially freed first), the physical page filled with the contents from storage, and assigned to that virtual address. Address translations are stored in an in-RAM dictionary called the *page table*.

AT incurs such a significant cost on real computers that modern CPUs come with specialized hardware accelerators called *translation lookaside buffers (TLBs)* that cache part of the page table. *TLB hits*, that is, successful lookups in the TLB, are fast, typically a single or small number of cycles [11]. In contrast, it can take hundreds or even thousands of CPU cycles to perform an address translation in the page table, when there is a *TLB miss* [1, 7].

Although the cost of AT is ignored in the paging problem, the cost can be high because *every memory reference* must undergo address translation, whereas page fetches may be rare. Moreover, one can avoid paging by purchasing more RAM,

---

\*The results presented are from the paper of the same name [2].

<sup>†</sup>Stony Brook University, USA.

<sup>‡</sup>Yale University, USA

<sup>§</sup>VMWare Research, USA

<sup>¶</sup>Rutgers University, USA

<sup>||</sup>MIT, USA

<sup>\*\*</sup>UNC Chapel Hill, USA

and this is generally considered money well spent. In contrast, TLBs have hit hard physical and power limits, making AT costs effectively unavoidable. Larger TLBs would have higher hit rates, but the size of TLBs is limited because it is expensive—in terms of time, transistors, and power [3]—to perform (parallel) hardware key-value lookups in tables with many entries. TLBs are so small that some workloads spend as much as 83% of their execution time on address translations [1] (see also [5, 6, 8, 13]).

Here, we address the resource allocation problem of how to organize both the TLB and the physical-address assignment in order to simultaneously optimize the total cost of address translation and paging. We show that, by combining ideas from low-associativity paging, recent advances in hashing, and compression, one can achieve strong, provable guarantees on the costs incurred by both the TLB and the page fetches. The main idea is to take into account not just which pages are cold but where they are stored when scheduling which pages to evict.

A critical benefit of proliferating address-translation hardware is scalability: by decentralizing and replicating the translation, each device can independently mediate and translate memory accesses, without the CPU and OS kernel becoming a bottleneck. This is similar to the fact that in multi-CPU systems, each CPU core has its own TLB, facilitating more scalable AT. These hardware address translators have different names in different contexts, so for simplicity, in this paper we will call them all TLBs.

**Huge pages and how TLBs change page allocations.** What makes TLBs interesting is that, rather than caching data, they cache pointers to data. Notably, this means that a *single pointer* can potentially point to a *very large amount* of data.

Indeed, the main thrust of increasing the effectiveness of TLBs in systems design has been to use *huge pages*, which are runs of pages that are contiguous in the virtual address space [5]. Critically, existing huge-page methods require the run of pages also be placed contiguously in RAM (i.e., physical memory), so that a single TLB entry can translate any address in any page that is included in the huge page.<sup>1</sup> In this case, the TLB is used as a key-value store in which the keys are virtual addresses of huge pages (rather than of standard-size pages) and the values are physical addresses of huge pages (rather than of standard-size pages).

We call the set of page translations that a TLB entry encodes its *coverage*. If the coverage of a TLB entry forms a contiguous run of virtual addresses defined by the high-order bits of the virtual addresses so encoded, we say that that entry encodes a *virtual huge page*. If, additionally, the corresponding physical pages are stored contiguously in *physical memory*, then we say those pages form a *physical huge page*.

---

<sup>1</sup>In our discussion, we elide many details of huge pages and TLBs, such as that most systems that implement huge pages use different TLBs for each size [4, 14] and only between one and three sizes are allowed, depending on the implementation. The algorithmic problems are the same, whether we are considering TLBs in the wild or the semi-domesticated TLBs described here.

**Virtual and physical huge pages, the good and the bad.** Virtual huge pages are an effective technique for reducing TLB misses [8,10], not merely because they increase the coverage of each individual TLB entry, but also because they translate a contiguous run of virtual addresses. Programs that exhibit spacial locality in their memory-access patterns benefit from the large coverage of each huge page.

On the other hand, physical huge pages *increase* IO costs for three reasons. *Page-fault amplification:* a huge page must have all constituent pages in memory, which may increase the cost of paging. *Reduced RAM utilization:* Huge pages may waste memory, reducing the effective size of RAM. *Fragmentation:* Mixing regular and huge pages in a RAM (or even having more than one size of huge page) induces fragmentation, or it induces more IO in order to defragment.<sup>2</sup>

In summary, huge pages come with a tradeoff: *virtual* huge pages enable a reduction in TLB misses, but *physical* huge pages cause an increase in IOs. There is a vast architecture and operating systems literature on optimizing the benefits and costs of huge pages (see, e.g. [10] and citations therein).

**The full cost of address translation.** In order to fully quantify the cost of address translation, one must consider TLB misses and IOs together. We call the software/hardware algorithm that manages the TLB, the page table, and the layout of pages in RAM a *memory-management algorithm*.

To measure the cost of a memory-management algorithm we introduce the *address-translation cost model*: each IO costs 1, each TLB miss costs  $\epsilon \in (0, 1)$ , and each TLB hit costs 0.

**Huge-page decoupling: all the virtual with none of the physical.** We show how to transform any paging algorithm into a low-associativity paging algorithm without increasing the IOs, while using minimal resource augmentation. Using this transformation, we can implement huge-page decoupling in order to realize the benefits of virtual huge pages without the need for physical huge pages.

## References

- [1] Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, and Michael M. Swift. Efficient virtual memory for big memory servers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2013.
- [2] Michael A Bender, Abhishek Bhattacharjee, Alex Conway, Martín Farach-Colton, Rob Johnson, Sudarsun Kannan, William Kuszmaul, Nirjhar Mukherjee, Don Porter, Guido Tagliavini, et al. Paging and the address-translation problem. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 105–117, 2021.

---

<sup>2</sup>Rather than evicting pages to disk, one could also try to defragment those pages in RAM [8,9]. The challenge in practice is that the performance overheads of defragmentation, even in memory, can easily exceed the performance benefits of huge pages.

- [3] Abhishek Bhattacharjee. Preserving virtual memory by mitigating the address translation wall. *IEEE Micro*, 37(5):6–10, 2017.
- [4] Intel’s Cascade Lake microarchitecture. [https://en.wikichip.org/wiki/intel/microarchitectures/cascade\\_lake](https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake). Accessed: 02/02/2020.
- [5] Mel Gorman. Linux huge pages. <https://lwn.net/Articles/375096/>, 2010.
- [6] Mel Gorman. AMD Zen architecture. <https://en.wikichip.org/wiki/amd/microarchitectures/zen>, 2018.
- [7] V. Karakostas, J. Gandhi, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. S. Unsal. Energy-efficient address translation. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 631–643, 2016.
- [8] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and Emmett Witchel. Coordinated and efficient huge page management with ingens. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 705–721. USENIX Association, November 2016.
- [9] Juan Navarro, Sitaram Iyer, Peter Druschel, and Alan L. Cox. Practical, transparent operating system support for superpages. In *5th Symposium on Operating System Design and Implementation (OSDI)*, 2002.
- [10] Ashish Panwar, Sorav Bansal, and K. Gopinath. Hawkeye: Efficient fine-grained os support for huge pages. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 347–360. Association for Computing Machinery, 2019.
- [11] SandyBridge. <https://www.7-cpu.com/cpu/SandyBridge.html>.
- [12] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, February 1985.
- [13] Michael M. Swift. Towards  $O(1)$  memory. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS)*, pages 7–11, 2017.
- [14] AMD’s Zen microarchitecture. <https://en.wikichip.org/wiki/amd/microarchitectures/zen>. Accessed: 07/15/2020.

# Tight Bounds for Parallel Paging and Green Paging\*

Kunal Agrawal<sup>†</sup>   Michael A. Bender (Speaker)<sup>‡</sup>   Rathish Das<sup>§</sup>  
William Kuszmaul<sup>¶</sup>   Enoch Peserico<sup>||</sup>   Michele Scquizzato<sup>§</sup>

---

## 1 Introduction

In the *parallel paging* problem, there are  $p$  processors that share a cache of size  $k$ . The goal is to partition the cache among the processors over time in order to minimize their average completion time. For this long-standing open problem, we give tight upper and lower bounds of  $\Theta(\log p)$  on the competitive ratio with  $O(1)$  resource augmentation.

A key idea in both our algorithms and lower bounds is to relate the problem of parallel paging to the seemingly unrelated problem of *green paging*. In green paging, there is an energy-optimized processor that can temporarily turn off one or more of its cache banks (thereby reducing power consumption), so that the cache size varies between a maximum size  $k$  and a minimum size  $k/p$ . The goal is to minimize the total energy consumed by the computation, which is proportional to the integral of the cache size over time.

We show that any efficient solution to green paging can be converted into an efficient solution to parallel paging, and that any lower bound for green paging can be converted into a lower bound for parallel paging. We then show that, with  $O(1)$  resource augmentation, the optimal competitive ratio for deterministic online green paging is  $\Theta(\log p)$ , which, in turn, implies the same bounds for deterministic online parallel paging.

## 2 Background on Sequential and Parallel Paging

The problem of managing the contents of a *cache* (i.e., a small *fast memory*) is critical to achieving good performance on large machines with multi-level memory hierarchies. This problem is classically known as *paging* or *caching*. When a processor accesses a location in fast memory, the access cost is small (the access

---

\*Most of these results were presented at SODA '21, but we also include more recent work.

<sup>†</sup>Washington University in St. Louis, USA. [kunal@wustl.edu](mailto:kunal@wustl.edu).

<sup>‡</sup>Stony Brook University, USA. [bender@cs.stonybrook.edu](mailto:bender@cs.stonybrook.edu).

<sup>§</sup>Waterloo, CA. [rathishdas@gmail.com](mailto:rathishdas@gmail.com).

<sup>¶</sup>MIT, USA. [kuszmaul@mit.edu](mailto:kuszmaul@mit.edu).

<sup>||</sup>Università degli Studi di Padova, Italy. [enoch@dei.unipd.it](mailto:enoch@dei.unipd.it), [scquizza@math.unipd.it](mailto:scquizza@math.unipd.it).

is a *hit*); when it accesses a location that is not in fast memory, the access cost is large (the access is a *miss* or a *fault*). The paging algorithm decides which *pages* (or *blocks*) remain in fast memory at any point in time or, in other words, which page(s) to evict when a new page is brought into fast memory. This problem is generally formulated as being *online*, meaning that the paging algorithm does not know the future requests.

Sequential paging—when there is a single processor accessing the fast memory—has been studied for decades and is a very well-understood problem [2, 13, 15], including several of its extensions.

In this paper, we study the problem of *parallel paging* where  $p$  processors share the same fast memory of some size  $k$ . Each processor runs its own program, and the set of pages accessed by different programs are disjoint. At each point in time, the paging algorithm gets to decide how much cache goes to each processor, and also gets to dictate each processor’s eviction strategy. The goal is to share the small memory among the processors in a way that minimizes some objective function of processors’ completion times. We focus on minimizing average completion time, and we also give bounds on makespan and median completion time.

The parallel paging problem introduces complexity that is not seen in the sequential problem. First, multiple processors compete for the same resource and the paging algorithm must decide, for each processor and at each time, how many and which of its pages to keep in cache. The marginal benefit of having more memory may vary across processors and this relationship may not have good structure. For instance, it may be that Processor 1 derives more marginal benefit from one extra page compared to Processor 2 while at the same time, Processor 2 derives more benefit from ten extra pages compared to Processor 1. In addition, this marginal benefit of extra cache can vary over time and the online paging algorithm must change the number of pages of fast memory allocated to the processors accordingly. Complicating matters even further, the actual scheduling of processors matters. For instance, running a small subset of processors—and temporarily stalling all others—may allow for better performance compared to running all processors. Therefore, how the different processor’s accesses are *interleaved* is an important part of what the parallel paging algorithm must decide, and different interleavings of accesses can lead to vastly different performances.

Whereas sequential paging has been understood for decades [2, 13, 15], parallel paging has largely resisted analysis. The only known upper bounds for parallel paging [1, 4, 6, 7, 9, 10, 14] consider relaxations of the problem in which the interleaving of accesses by different processors is fixed ahead of time—in particular, this is enforced by making it so that, whenever any processor blocks on a miss, *all of the processors block*, which in turn eliminates a large amount of the parallelism inherent to the problem. The known lower bounds [8, 12], on the other hand, focus only on the offline parallel paging problem, and on analyzing the performance of traditional paging algorithms such as LRU. Parallel paging has also been extensively studied within the systems community, particularly after multicore processors became mainstream—starting from some pioneering work on (offline

and online) heuristics that dynamically adjust the sizes of the cache partitions dedicated to each processor (see, e.g., [5, 11, 16–19]).

**This Paper.** We consider this long-standing open problem of parallel paging in its general form when the interleavings are not fixed. We analyze this problem in a *timed* model where a hit takes unit time and a miss takes  $s$  time units for some parameter  $s \geq 1$ . In the traditional model used for paging [3, 15], a hit costs 0 and a miss costs 1 — we call this the 0-1 model. The timed model is more general and captures the 0-1 model as a special case (by letting  $s$  tend towards infinity).

An important feature of the timed model is that it captures the passage of time even when a processor experiences a hit. Modeling this passage of time allows for a more fine-grained analysis of parallel paging since it allows us to compare the progress of one processor to another even if one is experiencing hits while the other is experiencing misses. This avoids the unrealistic assumption in the 0-1 model that a particular processor can have an infinite number of hits in the time that another processor has a single miss.

We give tight upper and lower bounds for the deterministic online parallel paging problem in the timed model showing that the optimal competitive ratio for average completion time is  $\Theta(\log p)$ , using constant resource augmentation. We also consider makespan, proving a lower bound of  $\Omega(\log p)$  and an upper bound of  $O(\log^2 p)$  for the competitive ratio. This result represents significant progress on a long-standing open problem—it was first articulated by Fiat and Karlin [7] in 1995 and has remained open even in the 0-1 model. A remarkable feature of our algorithms is that they are oblivious to  $s$ , achieving optimal competitive ratios for all values of  $s$  simultaneously.

A foundational idea in both our algorithms and lower bounds is to relate the problem of parallel paging to the (seemingly unrelated) problem of *green paging*, which focuses on minimizing the memory usage (hence, e.g., the energy consumption) of a (single) processor’s cache for a computation.

**Green Paging.** In green paging, the fast memory consists of *memory banks* that can be turned on or off over time. Memory banks that are active (i.e., turned on) can store pages—and thus requests for those pages result in a hit—but also consume energy; memory banks that are inactive cannot store pages, but do not consume energy. The goal in green paging is to minimize the total energy consumption of a computation. More formally, there is a single processor that is running, and the green paging algorithm gets to control both the page-eviction policy and the size of the processor’s cache over time, assigning any size between a minimum of  $k/p$  and a maximum of  $k$ , where  $p$  is a given parameter. The goal is to service the request sequence while minimizing the integral of memory capacity over time—a quantity we call *memory impact*. This is a simple model for studying the total amount of memory usage over time by a computation.

In this paper, green paging serves both as a problem to be studied on its own, and as an *analytical tool* for studying parallel paging. Indeed, both our upper and lower bounds hinge on unexpected relationships between the two problems.

## References

- [1] R. D. Barve, E. F. Grove, and J. S. Vitter. Application-controlled paging for a shared cache. *SIAM Journal on Computing*, 29(4):1290–1303, 2000.
- [2] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.
- [3] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [4] P. Cao, E. W. Felten, and K. Li. Application-controlled file caching policies. In *Proceedings of the USENIX Summer 1994 Technical Conference (USTC)*, pages 171–182, 1994.
- [5] J. Chang and G. S. Sohi. Cooperative cache partitioning for chip multiprocessors. In *ACM International Conference on Supercomputing 25th Anniversary*, pages 402–412, 2007.
- [6] E. Feuerstein and A. Strejilevich de Loma. On-line multi-threaded paging. *Algorithmica*, 32(1):36–60, 2002.
- [7] A. Fiat and A. R. Karlin. Randomized and multipointer paging with locality of reference. In *STOC*, pages 626–634, 1995.
- [8] A. Hassidim. Cache replacement policies for multicore processors. In *Proc. 1st Symposium on Innovations in Computer Science (ICS)*, pages 501–509, 2010.
- [9] A. K. Katti and V. Ramachandran. Competitive cache replacement strategies for shared cache environments. In *IPDPS*, pages 215–226, 2012.
- [10] R. Kumar, M. Purohit, Z. Svitkina, and E. Vee. Interleaved caching with access graphs. In *SODA*, pages 1846–1858, 2020.
- [11] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In *14th Symposium on High Performance Computer Architecture (HPCA)*, pages 367–378, 2008.
- [12] A. López-Ortiz and A. Salinger. Paging for multi-core shared caches. In *Proc. 3rd Innovations in Theoretical Computer Science conference (ITCS)*, pages 113–127, 2012.
- [13] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.
- [14] I. Menache and M. Singh. Online caching with convex costs. In *SPAA*, pages 46–54, 2015.
- [15] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
- [16] H. S. Stone, J. Turek, and J. L. Wolf. Optimal partitioning of cache memory. *IEEE Transactions on Computers*, 41:1054–1068, 1992.
- [17] G. E. Suh, L. Rudolph, and S. Devadas. Dynamic partitioning of shared cache memory. *The Journal of Supercomputing*, 28(1):7–26, 2004.
- [18] D. Thiébaud, H. S. Stone, and J. L. Wolf. Improving disk cache hit-ratios through cache partitioning. *IEEE Transactions on Computers*, 41:665–676, 1992.
- [19] Y. Xie and G. H. Loh. Pipp: promotion/insertion pseudo-partitioning of multi-core shared caches. *ACM SIGARCH Computer Architecture News*, 37(3):174–183, 2009.

# Relations between Periodic Scheduling with Harmonic Periods and 2D Packing

Claire Hanen (speaker) \*

Zdeněk Hanzálek †

---

## 1 Introduction

An embedded processor that executes computation tasks needed for control loops is a typical example of an application that must operate periodically within the bounds of the control loop's periods. Periodic scheduling problems [12] are frequent in many applications, including avionics [3], automotive [11], software-defined radio [1], and periodic machine maintenance [13].

In this paper we consider a set of  $n$  independent jobs,  $J_1, \dots, J_n$ . Each job  $J_i$  is characterized by its processing time  $p_i$ , and its period  $T_i$ . The jobs are to be performed on a single machine. A periodic schedule thus assigns a starting time  $s_i < T_i$  to the first occurrence of each job  $J_i$ , the  $k^{\text{th}}$  occurrence of  $J_i$  is then performed at time  $s_i + (k - 1)T_i$ . Two occurrences of two jobs  $J_i, J_j$  cannot overlap. We assume that periods belong to a harmonic set  $\{T^1, \dots, T^r\}$ , so that  $\forall k \in \{2, \dots, r\}, T^k = b_k T^{k-1}$  where  $b_k$  is an integer, and we first address the existence of a feasible periodic schedule. Notice that we can assume without loss of generality that processing times are not greater than  $T^1$  (otherwise there could not be a feasible single machine schedule).

Periodic scheduling on a single machine with unit processing times was shown to be Weakly NP-complete by [1] using a reduction from the graph coloring problem. Furthermore, [7] tightened the result by proving that no pseudo-polynomial algorithm exists unless SAT can be solved by a randomized algorithm in expected time  $n^{\mathcal{O}(\log(n) \log \log(n))}$ . When non unit processing times are considered on a single machine, [8] showed that the non-preemptive periodic scheduling with arbitrary task initial phases (i.e., each task is released at its release-date and must be finished within its period time units) is strongly NP-hard. For the case of the harmonic period set efficient heuristic algorithms have been designed ([6]). However, Cai and Kong [4] established that with arbitrary job processing times, the problem is NP complete in the strong sense. On another hand, periodic scheduling on parallel identical resources while processing times and periods belong to the same

---

\* [claire.hanen@lip6.fr](mailto:claire.hanen@lip6.fr) Sorbonne Université, CNRS, LIP6, F-75005 Paris, France and UPL, Université Paris Nanterre, F-92000 Nanterre, France

† [zdenek.hanzalek@cvut.cz](mailto:zdenek.hanzalek@cvut.cz) Czech Technical University in Prague, CIIRC, Czech Republic

harmonic set has been proved to be solvable in polynomial time by Korst [9]. The problem with power of two periods and processing times on parallel machines is also addressed in [2]. The authors provide an approximation algorithm for the minimization of the number of machines for the windows scheduling problem, in which the job periods are allowed to be shortened.

## 2 Relation between 2D packing and feasible schedules

In this section we consider the decision problem of scheduling the jobs on one machine so that no collision occurs denoted by  $1|T_i^{harm}, p_i^{nonharm}|-$  according to extended Graham notations.

It is noted as a simple remark in [9] that the problem with harmonic periods and processing times is similar to a 2D packing problem. Similarly Zhao et al [14] indicate their algorithm is related to 2D bin packing. The relation between the two problems is shown for powers of two periods and for any processing times in Lukaszewycz [10]. In this paper we explicitly show which 2D packing problem is equivalent to the periodic scheduling decision problem, according to the following sketch:

- if we shift the occurrence number of jobs, or choose any time origin  $o$ , it will not affect the existence of a periodic schedule. Let  $o$  be the starting time of the first job  $J_f$  with period  $T^1$ . We define  $s'_i = s_i$  if  $s_i \geq o$  and  $s'_i = T_i + s_i$  otherwise. We can decompose  $s'_i - o \geq 0$  modulo  $T^1$  :  $s'_i - o = u_i + v_i T^1$ , with  $u_i < T^1$ , and we have  $u_i + p_i \leq T^1$ , since an occurrence of job  $J_i$  takes place at  $o + u_i$  between two occurrences of  $J_f$  at  $o$  and  $o + T^1$ .
- Each job  $J_i$  corresponds to a rectangle  $R_i$  of width  $p_i$  and height  $h_i = \frac{T^r}{T_i}$ . A packing defines for each rectangle  $R_i$  a coordinate  $(x_i, y_i)$  of its lower left corner. The rectangles are to be packed without overlapping in a rectangle  $\mathcal{R}$  of width  $T^1$  and height  $H = \frac{T^r}{T^1}$ .
- For each job  $J_i$ ,  $v_i$  can be expressed with the mixed radix system representation with base vector  $b_2, \dots, b_r$ , and we define a flip operation that reverses the first digits of this representation. The resulting number is  $flip_i(v_i)$ .

**Theorem 1** *A feasible periodic schedule  $\sigma$  of the jobs  $J_1, \dots, J_n$  on one machine exists if and only if there exists a 2D packing of the rectangles  $R_1, \dots, R_n$  into the rectangle  $\mathcal{R}$  in which the  $y$  coordinate of each rectangle is a multiple of its height. We have the following relation between the schedule and the packing for any job  $J_i$ :  $x_i = u_i, y_i = h_i \times flip_i(v_i)$ .*

As a corollary, this result can also be used to give a new proof of NP-Completeness for the problem with only two different harmonic periods (by a reduction from bin packing).

We also show that a 2D packing associated to a schedule can be viewed as a sequence of series and parallel composition of rectangles. We then establish that a

dominant form of packing, called  $T$ -shape packing can be defined. In a  $T$ -shape packing, the rectangles of greater height ( associated with jobs of lower period ) are always to the left of smaller height rectangles. Rate monotonic algorithms that schedule jobs sorted by increasing periods do in fact build such a  $T$ -shape packing.

### 3 Optimization questions

If a feasible schedule exists, Zhao et al [14] discuss an application for which for each time cycle  $T^1$ , the first part of width  $w$  of the cycle  $T^1$  is booked for time triggered traffic, i.e. the periodic jobs, while the rest is booked for event triggered communications. This lead us to an optimization criteria: minimizing  $w$  so that the time devoted to event-triggered communication is maximum. On the other hand, we can also consider that  $T^1$  might vary to ensure the existence of a feasible schedule, while the harmonic proportions between periods  $b_k = \frac{T^k}{T^{k-1}}$  remain identical. Both approaches lead to a 2D packing problem where the jobs rectangles are to be packed in a big rectangle  $\mathcal{R}$  with height  $H$  and variable width  $w$  (to be minimized).

We prove that any shelf packing algorithm, which are among the most used heuristics, can be modified to produce a packing satisfying the particular property stated in Theorem 1. It follows that the performance of the most used shelf algorithm FFDH from Coffman, Garey and Johnson [5] for 2D strip packing with harmonic height of rectangles applies to our periodic scheduling problem. The performance guarantee of this algorithm is valid for the width of the periodic schedule built by this algorithm  $w^{FFDH}$  :

**Theorem 2** (corollary of [5]) *If  $w^{opt}$  is the optimal width :*

$$w^{FFDH} \leq w^{opt} + \max_{i \in \{1, \dots, n\}} p_i \quad (1)$$

To conclude, by making explicit the relation between schedules of periodic jobs and 2D packings on one machine, we believe that new exact and approximation algorithms can be built for problems involving periodic jobs. Moreover, these ideas can be extended to handle parallel machines by the juxtaposition of rectangles associated with each machine.

### Acknowledgment

Thanks to Karolina Rezkova for fruitful discussions during her master thesis.

### References

- [1] Amotz Bar-Noy, Randeep Bhatia, Joseph (Seffi) Naor, and Baruch Schieber. Minimizing service and operation costs of periodic scheduling. *Mathematics of Operations Research*, 27(3):518–544, 2002.

- [2] Amotz Bar-Noy, Richard E. Ladner, Tami Tamir, and Tammy VanDeGrift. Windows scheduling of arbitrary-length jobs on multiple machines. *J. Sched.*, 15(2):141–155, 2012.
- [3] Sofiene Beji, Sardaouna Hamadou, Abdelouahed Gherbi, and John Mullins. SMT-based cost optimization approach for the integration of avionic functions in IMA and TTEthernet architectures. *Proceedings of the 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*, pages 165–174, 2014.
- [4] Yang Cai and MC Kong. Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems. *Algorithmica*, 15(6):572–599, 1996.
- [5] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin-packing — an updated survey. *Algorithm Design for Computer System Design International Centre for Mechanical Sciences*, 284:49–106, 1984.
- [6] Jan Dvořák and Zdeněk Hanzálek. Multi-variant scheduling of critical time-triggered communication in incremental development process: Application to flexray. *IEEE Transactions on Vehicular Technology*, 68(1):155–169, 2019.
- [7] Tobias Jacobs and Salvatore Longo. A new perspective on the windows scheduling problem. *arXiv preprint arXiv:1410.7237*, 2014.
- [8] Kevin Jeffay, Donald F Stanat, and Charles U Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *IEEE real-time systems symposium*, pages 129–139, US, 1991. IEEE.
- [9] Jan Korst, Emile Aarts, and Jan Karel Lenstra. Scheduling periodic tasks. *INFORMS journal on Computing*, 8(4):428–435, 1996.
- [10] Martin Lukaszewycz, Michael Glaß, Jürgen Teich, and Paul Milbredt. FlexRay schedule optimization of the static segment. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 363–372, France, Grenoble, 2009. IEEE/ACM.
- [11] Anna Minaeva, Benny Akesson, Zdeněk Hanzálek, and Dakshina Dasari. Time-triggered co-scheduling of computation and communication with jitter requirements. *IEEE Transactions on Computers*, 67(1):115–129, 2017.
- [12] Anna Minaeva and Zdeněk Hanzálek. Survey on periodic scheduling for time-triggered hard real-time systems. *ACM Comput. Surv.*, 54(1), Mar 2021.
- [13] W.D. Wei and C.L. Liu. On a periodic maintenance problem. *Operations Research Letters*, 2(2):90–93, 1983.
- [14] R. Zhao, G. H. Qin, and J. Q. Liu. Optimal scheduling of the flexray static segment based on two-dimensional bin-packing algorithm. *International Journal of Automotive Technology*, 17(4):703–715, August 2016.

# Filling a Theater During the COVID-19 Pandemic\*

Danny Blom <sup>†</sup>    Rudi Pendavingh <sup>‡</sup>    Frits Spieksma (Speaker) <sup>§</sup>

---

## 1 Introduction

All around the world, the COVID-19 crisis has hit the cultural sector, as well as some professional sports, hard. Festivals are canceled, choirs have stopped performing, and theaters and some sport organizations are struggling to survive. Different countries or regions have imposed different rules in an attempt to stop the spread of the virus. We do not aim here to overview the precise (dynamic) contents of all these rules; a number of descriptions of such rules can be found on governmental websites, see e.g. [2]. The economic impact of these measures is huge; it is estimated in [3] that in Britain alone, the cultural sector has suffered 200m pounds in covid-related losses by March 2021.

Thus, the implementation of distancing rules has (had) a dramatic impact on the operation of any theater or stadium. Indeed, for many theaters, the challenge is to find a way to welcome their guests while satisfying the distance rules and still be commercially viable.

Here, we focus on the question to what extent large audiences can still be accommodated in a theater (or stadium) when distance rules must be satisfied. We describe and solve an optimization problem that, given the layout of the seats in a theater and the distribution of the demand, allows a theater to compute a safe seating arrangement that attains the maximum occupancy.

The Music Building Eindhoven (MBE), located in the city of Eindhoven in the Netherlands, features a “Grand Room” (1,250 seats) and a “Small Room” (400 seats). This theater has served as a motivation for this study, and all our computational efforts are based on its two theater rooms. Our findings have been implemented by MBE, allowing them to remain open (as long as governmental rules allow).

---

\*The full paper corresponding to this abstract has been accepted for the INFORMS Journal on Applied Analytics

<sup>†</sup>[d.a.m.p.blom@tue.nl](mailto:d.a.m.p.blom@tue.nl). Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

<sup>‡</sup>[r.a.pendavingh@tue.nl](mailto:r.a.pendavingh@tue.nl). Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

<sup>§</sup>[f.c.r.spieksma@tue.nl](mailto:f.c.r.spieksma@tue.nl). Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

## 2 A problem description

Here, we describe the crucial ingredients of our problem.

Due to the exception of the distance rules for family members, a relevant factor is the *size*  $t \in T$  of a family, with  $T \subset \mathbb{Z}_+$  the set of allowed family sizes. In particular, we will call a family of size 1 a *singleton*, of size 2 a *pair*, of size 3 a *triple*, and a family of size 4 a *quad*. Guests from the same household are allowed to sit next to each other, within the 1.5m bound; in fact, we assume that a family of size  $t$  occupies  $t$  consecutive seats.

Consider a theatre, and let  $\mathcal{S}$  denote the set of seats of the theatre. Each seat is specified by its *row*  $r$ , and its *position*  $s$  in row  $r$ . Thus, a seat is a pair of integers  $(r, s) \in \mathbb{Z} \times \mathbb{Z}$  and the set of seats is a collection  $\mathcal{S} \subseteq \mathbb{Z} \times \mathbb{Z}$ .

Using a separating distance of 1.5 meter (as stipulated by the Dutch government), a simple calculation reveals that when accommodating a pair, there is a “ring” of 14 other seats around it that are forbidden for use by any member of another family. The forbidden zone corresponding to a pair is depicted in red in Figure 1. In general, as a family of size  $t$  located at seat  $(r, s)$  will occupy the consecutive seats  $\mathcal{S}_{r,s,t} := \{(r, s + i) : i = 0, \dots, t - 1\}$ , the corresponding forbidden zone for a family of size  $t$  consists of  $2t + 10$  seats.

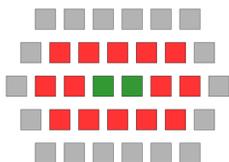


Figure 1: The red seats cannot be occupied whenever the green seats are occupied by a pair. This example is based on the measures of the MBE.

A subset  $\mathcal{A} \subseteq \mathcal{S} \times T$  is a *seating arrangement* if each  $(r, s, t) \in \mathcal{A}$  indicates a possible location of a family of size  $t$  at  $(r, s)$ . Thus, a seating arrangement specifies for each seat  $(r, s)$  whether it is occupied by a member of a family of size  $t$ . A seating arrangement can be regarded as a plan to fill the theatre. We call a seating arrangement  $\mathcal{A}$  *safe* if no two guests of different families are seated within 1.5 meter, or, in other words, when no member of a family is in the forbidden zone of another family. A relevant property of a seating arrangement is the number of guests it contains; we denote this number by the *size* of the seating arrangement; thus, the size is nothing else but the number of guests present in the theatre.

Finally, we use so-called *target profiles* to take the size of families visiting the performance into account. Indeed, we use prior information on the distribution of the customers over singletons, pairs, triples and quads as a proxy for customer behaviour.

We can now state our optimization problem. Given the theatre characteristics, the separating distance, and a target profile:

- *Maximize* the number of customers present in the theatre (i.e., the size of

a seating arrangement) *by selecting* occupied/unoccupied seats subject to covid-distancing and family-group constraints.

### 3 Results

We establish a connection between safe seating arrangements and packing of disjoint trapezoids. More formally, we record the following statement:

**Theorem 1** *A collection of pairwise disjoint trapezoids corresponds to a seating arrangement that is safe, and vice versa.*

See Figure 2 for an illustration.

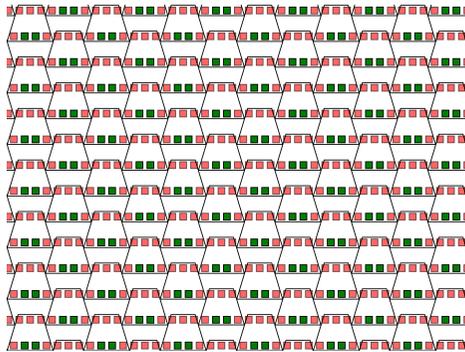


Figure 2: An illustration that suggests how disjoint trapezoids relate to safe seating arrangements.

We use integer programming to compute optimum packings of trapezoids (see also [4] and [1]). These results, combined with other measures and ideas (such as holding consecutive shows, and being able to assess the effect of alternating empty rows) have contributed to the enduring operation of the MBE.

### References

- [1] Datema, G.J. (2022), *Optimising the Occupancy of Theatres under Minimal Distance Constraints*, Bachelor Thesis, Eindhoven University of Technology.
- [2] *Dutch measures against coronavirus*, <https://www.government.nl/topics/coronavirus-covid-19/tackling-new-coronavirus-in-the-netherlands/public-life>, accessed April 22, 2022.
- [3] The Economist (June 2021), *Covid-19 is an existential threat to Britain's theatres*.
- [4] Wang, A., C. L. Hanselman, and C. E. Gounaris (2018), *A customized branch-and-bound approach for irregular shape nesting*, *Journal of Global Optimization* **71**, 935-955.

# Queuing Safely for Elevator Systems amidst a Pandemic

Sai Mali Ananthanarayanan \*      Charles Branas †

Adam N. Elmachtoub ‡      Clifford Stein (Speaker) §      Yeqing Zhou ¶

---

## 1 Introduction

The COVID-19 pandemic has made it imperative to design interventions for people to stay safe in potentially crowded areas. For high-rise buildings, social distancing reduces the capacity of elevators, cutting the number of passengers per elevator by two-thirds or as much as over 90% the normal amount [1, 2, 3]. Reduced elevator capacity can cause large lobby queues and long wait times, resulting in crowding and reduced social distancing [1, 4, 5]. With no interventions and reduced capacity on elevators, the increased waiting times and queue lengths in the lobby could pose significant safety risks. Thus, an intervention to the public health problem of safely managing queues for elevator systems amidst a pandemic is needed (our team is composed of operations researchers and an epidemiologist). *In fact, this project was directly in collaboration with the NYC Mayor’s Office of the Chief Technology Officer and the Department of Citywide Administrative Services. These offices had continuous input into our work throughout the process and allowed us to conduct several on-site visits with building managers, where we talked to frontline staff and gathered input. We also presented our findings multiple times to a variety of agency staff, developed an instructional video, and provided open source code that can be tailored to the needs of different types of buildings.*

One can broadly consider two major forms of interventions based on (i) changing passenger behavior and (ii) elevator artificial intelligence. A variety of technological innovations from elevator companies and building management have been considered during the pandemic [4]. In many elevator systems, especially in older ones, changing the algorithms and technology of how the elevators navigate through the building is challenging or infeasible, and would require long-term

---

\*sai.mali@columbia.edu, SafeElevatorQueues@columbia.edu Department of Industrial Engineering and Operations Research and Data Science Institute, Columbia University, New York, NY 10027

†c.branas@columbia.edu Department of Epidemiology, Columbia University, New York, NY 10027

‡adam@ieor.columbia.edu Department of Industrial Engineering and Operations Research and Data Science Institute, Columbia University, New York, NY 10027

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

¶y.zhou2@tue.nl Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology (TU/e), Netherlands

planning and expensive modifications. Thus, in order to safely manage how passengers use and board elevators, we focus on *technology-free* interventions that are more accessible and practical for an overwhelming majority of buildings.

## 1.1 Contributions

Using mathematical modeling, epidemiological expertise, and simulation, we design and evaluate simple interventions to load passengers in elevators that can drastically reduce the length of lobby queues amidst a pandemic, rather than a hands-off approach such as first-come first-serve [6]. The proposed interventions increase efficiency of the elevator system, and are effective beyond the constraints imposed by a pandemic, making them useful even after the pandemic to manage lobby queues. Our interventions<sup>1</sup> do not require programming the elevators, and rely on using only signage and/or a queue manager (QM) to guide passengers. Our interventions use the elevators more efficiently – getting passengers to their destination at a faster aggregate rate (higher throughput) – by more carefully managing who uses which elevator when. We outline our contributions in detail.

1. We develop a general, open-sourced<sup>2</sup> simulation model that captures many of the details of elevator systems and allows us to study the impact of various interventions and queuing behavior. Our simulation model allows us to specify the number of elevators, capacity, elevator speed, and boarding times, and to measure and visualize the queue length and wait time of elevator systems for the various interventions we consider. We primarily focus on a case study calibrated by data from a large government building in New York City in need of managing elevator traffic amidst the COVID-19 pandemic.
2. We propose an intervention we call *Cohorting*, where we attempt to find any and all passengers going to the same floor as the first person in the queue. Simulations show Cohorting reduces waiting time for passengers and the queue length in the lobby significantly. In limited lobby spaces, we recommend the *Cohorting with Pairing* intervention, where we pair passengers going to the same floors. Pairing requires matching only two people at a time (rather than four, for instance) and hence, is practically easier to implement. We show the queue length can be further reduced if just a small fraction of passengers are willing to walk up or down one floor from their destination.
3. We propose the *Queue Splitting* intervention where we create separate queues for different groups of floors and load the elevators from queues in a round-robin fashion. The travel time of elevators is naturally reduced since passengers are likely to be going to the same or nearby floors. Queue Splitting needs less management effort compared to Cohorting, and splitting into just two groups achieves comparable performance to Cohorting in our case study.

---

<sup>1</sup>[https://youtu.be/5KvX7\\_WNGFw](https://youtu.be/5KvX7_WNGFw)

<sup>2</sup><https://github.com/saimali/elevators>

4. We analytically investigate the reason behind the strong performance of Cohorting and Queue Splitting using a technique from queuing theory known as stability analysis. Specifically, we characterize the system parameters required for each intervention to ensure that the queues do not increase in length over time, i.e., the queues are stable. Our theoretical analysis reveals that these interventions can effectively reduce the service time of elevators.

## 2 Interventions

The standard way most elevator systems operate is akin to first-come first-serve (FCFS). With a social distancing rule and reduced elevator capacity, safe interventions require moving away from FCFS, which means that some people may be allowed to “cut in line” in order to decrease queue lengths and waiting times, while serving as many passengers as possible. Our interventions may rely on a queue manager (QM) for implementation, either a personnel or a device with a screen.

We propose the *Cohorting* intervention, which seeks to group together passengers going to the same floor. In this intervention, passengers line up in a queue in order of arrival. When an elevator arrives, the first passenger boards. Then, the QM asks if anyone in the queue is going to the same floor as the first passenger and then they board as well (according to their arrival order). This process is repeated until the elevator is full or the queue is empty.

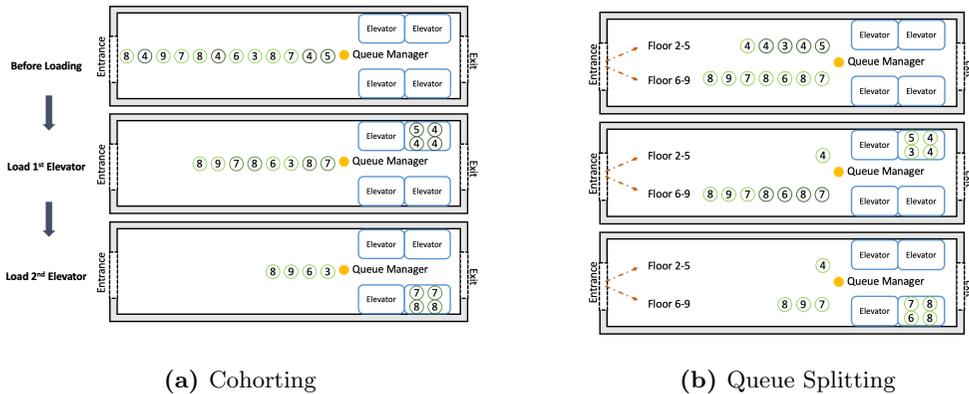
The next intervention we propose is *Queue Splitting*, where we form a separate queue for disjoint groups of floors, e.g., 2 – 8 and 9 – 16. Arriving passengers join a queue corresponding to their floor group, and elevators are boarded from the queues in a round robin fashion (possibly with the help of a QM). The queues are chosen in a round-robin fashion (or in a way to dynamically balance the length of each queue). Figure 1 shows an implementation of Cohorting and Queue Splitting.

The fundamental idea behind these interventions is to try to reduce queue buildup by maximizing the number of people in an elevator trip going to the same floor (or nearby floors), which in turn reduces the round trip time of an elevator trip (service time). In a simplified model, the key drivers of round trip time are the number of stops made  $S$  and the highest reversal floor  $H$  (i.e., the highest destination floor among all passengers boarding in the lobby),

$$\underbrace{\text{Round Trip Time}}_{\text{per elevator trip}} = \underbrace{2 \times \text{speed per floor} \times H}_{\text{Time spent moving up/down}} + \underbrace{\text{stop time per floor} \times S}_{\text{Time spent on stops}}$$

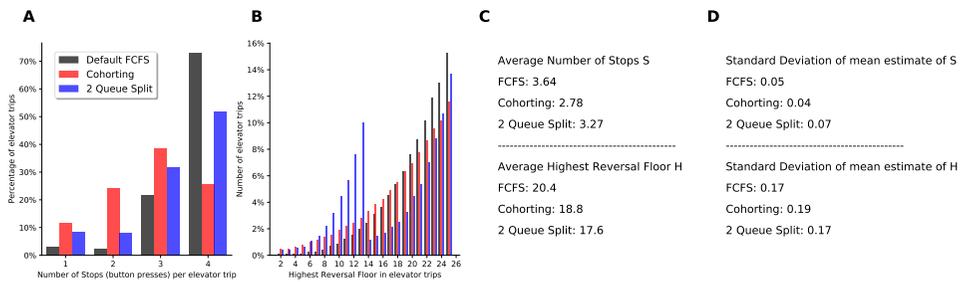
In Figure 2, we observe that elevators under FCFS makes an average of  $3.64 \pm 0.05$  stops, whereas under Cohorting, they make on average only  $2.78 \pm 0.04$  stops. FCFS has the highest average  $H$  of  $20.3 \pm 0.17$ , whereas 2 Queue Split has the lowest  $H$  at  $17.6 \pm 0.17$ . Thus, Cohorting and Queue Splitting reduce the key driver of system performance- the round trip time for the elevator trips, and serve more passengers in a given time period.

Figure 1: Illustrating the Cohorting and Queue Splitting interventions



Note. Passengers enter the lobby from the left and are guided by the Queue Manager (QM) to the elevators. We indicate the passengers who will board the next elevator using dark green circles.

Figure 2: Comparison of interventions using number of stops and highest reversal floor of elevator trips for our large building case study



Note. We report the average performance of 100 independent random instances. (A) The percentage of elevator trips with different number of stops made across interventions. (B) The percentage of elevator trips with different highest reversal floor  $H$  across interventions. (C,D) Average and standard deviation of the estimated mean of  $S$  and  $H$  for all interventions.

## References

- [1] L. Weber. NPR-Office Elevator In COVID Times: Experts Weigh In On How To Stay Safe, 2020.
- [2] D. Swinarski. Modelling elevator traffic with social distancing in a university classroom building. *Building Services Engineering Research and Technology*, 42(1):82–97, 2021.
- [3] D. Swinarski. Inside Higher Ed-Modeling Elevator Traffic With Social Distancing, 2020.
- [4] M. Wilson. New York Times-It Might Become the Scariest Part of Your Commute:The Elevator, 2020.
- [5] P. Smith. Bloomberg Law-Distancing at Reopened Offices Will Mean Long Elevator Lines, 2020.
- [6] A. Fujino, T. Tobita, K. Segawa, K. Yoneda, and A. Togawa. An elevator group control system with floor-attribute control method and system optimization using genetic algorithms. *IEEE Transactions on Industrial Electronics*, 44(4):546–552, 1997.

# Learning-Augmented Dynamic Power Management with Multiple States via New Ski Rental Bounds\*

Antonios Antoniadis (Speaker) <sup>†</sup>    Christian Coester <sup>‡</sup>    Marek Eliáš <sup>§</sup>  
Adam Polak <sup>¶</sup>    Bertrand Simon <sup>||</sup>

---

**The DPM Problem.** We study the online problem of minimizing power consumption in systems with multiple power-saving states. During idle periods of unknown lengths, an algorithm has to choose between power-saving states of different energy consumption and wake-up costs. More specifically, in the problem of *dynamic power management (DPM)*, we are given  $k + 1$  power states denoted by  $0, 1, \dots, k$ , with power consumptions  $\alpha_0 > \dots > \alpha_k \geq 0$  and wake-up costs  $\beta_0 < \dots < \beta_k$ . For state 0 we have  $\beta_0 = 0$  and we call this state the *active state*. The input is a series of idle periods of lengths  $\ell_1, \dots, \ell_T$  received online, i.e., the algorithm does not know the length of the current period before it ends. During each period, the algorithm can transition to states with lower and lower power consumption, paying energy cost  $x\alpha_i$  for residing in state  $i$  for time  $x$ . If  $j$  is the state at the end of the idle period, then it has to pay the wake-up cost of  $\beta_j$  to transition back to the active state 0. The goal is to minimize the total cost.

**Connection to Ski-Rental.** The special case of 2-state DPM systems, i.e., when there is only a single sleep state (besides the active state), is essentially equivalent to the classical *ski-rental problem*. This problem is defined as follows: A person goes skiing for an unknown number of days. On every day of skiing, the person must decide whether to continue renting skis for one more day or to buy skis. Once skis are bought there will be no more cost on the following days, but the cost of buying is much higher than the cost of renting for a day. It is easy to see that this captures a single idle period of DPM with a single sleep state whose running cost is 0. Given this equivalence, the known 2-competitive deterministic algorithm and  $e/(e - 1) \approx 1.58$ -competitive randomized algorithm for ski rental carry over to 2-state DPM, and these competitive ratios are tight. In fact, it was

---

\*The current abstract is based on a paper recently published at NeurIPS'21. A more extensive version can be found under [1].

<sup>†</sup>[a.antoniadis@utwente.nl](mailto:a.antoniadis@utwente.nl). University of Twente, Enschede, The Netherlands.

<sup>‡</sup>[christian.coester@gmail.com](mailto:christian.coester@gmail.com). University of Sheffield, United Kingdom.

<sup>§</sup>[marek.elias@unibocconi.it](mailto:marek.elias@unibocconi.it). Università Bocconi, Milan, Italy.

<sup>¶</sup>[adam.polak@epfl.ch](mailto:adam.polak@epfl.ch). EPFL, Lausanne, Switzerland.

<sup>||</sup>[bertrand.simon@cnrs.fr](mailto:bertrand.simon@cnrs.fr). IN2P3 Computing Center, CNRS, Villeurbanne, France.

shown by Irani et al. [2] and Lotker et al. [3] that the same competitive ratios carry over even to multi-state DPM.

**The Learning-Augmented Setting.** We are interested in developing learning-augmented algorithms for DPM. To that end, we assume that the algorithm receives at the beginning of the  $i$ th idle period a prediction  $\tau_i \geq 0$  for the value of  $\ell_i$  as additional input. The accuracy of the predicted lengths  $\tau_i$  is unknown to the algorithm and measured by means of a *prediction error*  $\eta := \sum_{i=1}^T \alpha_0 |\tau_i - \ell_i|$ . The goal is to develop an online algorithm that makes decisions based on (potentially inaccurate) predicted lengths of the idle periods. Ideally, the algorithm's performance is near-optimal when predictions are accurate and degrades gracefully with increasing prediction error, with a worst-case guarantee almost identical to the optimal classical online algorithm for the problem.

**Extension of Competitive-Analysis to Learning-Augmented Setting.** Classical online algorithms are typically analyzed in terms of competitive ratio. A (randomized) algorithm  $\mathcal{A}$  for an online minimization problem is said to be  $\rho$ -competitive (or alternatively, obtain a competitive ratio of  $\rho$ ) if for any input instance it holds:

$$\text{cost}(\mathcal{A}) \leq \rho \text{OPT} + c,$$

where  $\text{cost}(\mathcal{A})$  and  $\text{OPT}$  denote the (expected) cost of  $\mathcal{A}$  and the optimal cost of the instance and  $c$  is a constant independent of the online part of the input (i.e., the lengths  $\ell_i$  in case of DPM). In the learning-augmented setting, for  $\rho \geq 1$  and  $\mu \geq 0$ , we say that  $\mathcal{A}$  is  $(\rho, \mu)$ -competitive if

$$\text{cost}(\mathcal{A}) \leq \rho \cdot \text{OPT} + \mu \cdot \eta.$$

for any instance. This corresponds to a competitive ratio of  $\rho + \mu \frac{\eta}{\text{OPT}}$  (with  $c = 0$ ). While this could be unbounded as  $\eta/\text{OPT} \rightarrow \infty$ , as we will see our DPM algorithm achieves a favorable competitive ratio even in this case. For a  $(\rho, \mu)$ -competitive algorithm,  $\rho$  is also called the *consistency* (i.e., competitive ratio in case of perfect predictions) while  $\mu$  describes the dependence on the prediction error.

**Our Results.** Our first result is a new algorithm for the learning-augmented version of the ski-rental problem, that is  $(\rho, \mu(\rho))$ -competitive, where  $\rho \in [1, \frac{e}{e-1}]$ , and

$$\mu(\rho) := \max \left\{ \frac{1 - \rho \frac{e-1}{e}}{\ln 2}, \rho(1 - D)e^{-D} \right\},$$

where  $D \in [0, 1]$  is the solution to  $D^2 e^{-D} = 1 - \frac{1}{\rho}$ . Note that  $\mu(1) = 1$  and  $\mu\left(\frac{e}{e-1}\right) = 0$ . The above result is tight, in the sense that for any  $\rho \in \left[1, \frac{e}{e-1}\right]$  and any (randomized) algorithm  $\mathcal{A}$ , there exists a ski-rental instance with some

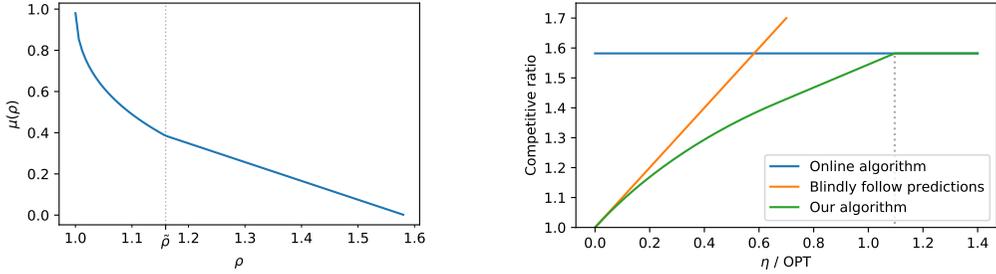


Figure 1: Illustration of  $\mu(\rho)$  and of the resulting competitive ratio in function of  $\eta/\text{OPT}$ .

prediction error  $\eta$  such that the expected cost of  $\mathcal{A}$  on that instance is at least  $\rho\text{OPT} + \mu(\rho)\eta$ . See Figure 1 (left) for an illustration of  $\mu(\rho)$ .

We extend the results of [2, 3] by giving a reduction from DPM to ski-rental in the learning-augmented setting, under a mild and natural assumption on the ski-rental algorithm – which our aforementioned algorithm satisfies. This reduction between the two problems, when applied to our ski-rental algorithm and in combination with techniques from online learning gives our main result: An algorithm that is “almost”  $(\rho, \mu(\rho))$ -competitive *simultaneously* for all  $\rho$ :

**Theorem 1** *For any  $\epsilon > 0$ , there is a learning-augmented algorithm  $\mathcal{A}$  for dynamic power management whose expected cost can be bounded as*

$$\text{cost}(\mathcal{A}) \leq (1 + \epsilon) \min \left\{ \rho\text{OPT} + \mu(\rho) \cdot \eta \quad \left| \rho \in \left[ 1, \frac{e}{e-1} \right] \right. \right\} + O \left( \frac{\beta_k}{\epsilon} \log \frac{1}{\epsilon} \right).$$

The above theorem implies a competitive-ratio that is arbitrarily close to  $\min_{\rho} \left\{ \rho + \mu(\rho) \cdot \frac{\eta}{\text{OPT}} \right\}$ , which is equal to 1 if  $\eta = 0$  and never greater than  $\frac{e}{e-1}$ . In other words our algorithm achieves near-optimal consistency and robustness (over many idle periods). See Figure 1 (right) for an illustration.

## References

- [1] A. ANTONIADIS, C. COESTER, M. ELIÁŠ, A. POLAK AND B. SIMON (2021). *Learning-Augmented Dynamic Power Management with Multiple States via New Ski Rental Bounds*. CoRR:abs/2110.13116.
- [2] S. IRANI, S. SHUKLA AND R. GUPTA (2003). *Online Strategies for Dynamic Power Management in Systems with Multiple Power-Saving States*. ACM Trans. Embed. Comput. Syst., 2(3):325–346.
- [3] Z. LOTKER, B. PATT-SHAMIR AND D. RAWITZ (2012). *Rent, Lease, or Buy: Randomized Algorithms for Multislope Ski-Rental*. SIAM J. Discret. Math., 26(2):718–736.

# Speed-Robust Scheduling: Sand, Bricks, and Rocks\*

Franziska Eberle (Speaker)<sup>†</sup>   Ruben Hoeksma<sup>‡</sup>   Nicole Megow<sup>§</sup>  
Lukas Nölke<sup>‡</sup>   Kevin Schewior<sup>¶</sup>   Bertrand Simon<sup>||</sup>

---

## 1 Introduction

Scheduling problems with incomplete knowledge of the input data have been studied extensively. There are different ways to model such uncertainty, the major frameworks being *online optimization*, where parts of the input are revealed incrementally, *stochastic optimization*, where parts of the input are modelled as random variables, and *robust optimization*, where uncertainty in the data is bounded. Most scheduling research in this context assumes uncertainty about the job characteristics. A lot less research addresses uncertainty about the machine environment, particularly, where the processing speeds of machines change in an unforeseeable manner.

In this paper, we are concerned with the question of how to design a partial schedule by committing to groups of jobs, to be scheduled on the same machine, before knowing the actual machine speeds. We consider a two-stage robust scheduling problem in which we aim for a schedule of minimum makespan on multiple machines of unknown speeds. Given a set of  $n$  jobs and  $m$  machines, we ask for a partition of the jobs into  $m$  groups, we say *bags*, that have to be scheduled on the machines after their speeds are revealed without being split up. That is, in the second stage, when the machine speeds are known, a feasible schedule assigns all jobs in the same bag to the same machine. The goal is to minimize the second-stage makespan.

More formally, we are given  $n$  jobs with processing times  $p_j \geq 0$ , for  $j \in \{1, \dots, n\}$ , and the number of machines,  $m \in \mathbb{N}$ . Machines run in parallel but

---

\*The paper was published in the proceedings of IPCO 2021 [1].

<sup>†</sup>[f.eberle@lse.ac.uk](mailto:f.eberle@lse.ac.uk) Department of Mathematics, London School of Economics, United Kingdom.

<sup>‡</sup>[r.p.hoeksma@utwente.nl](mailto:r.p.hoeksma@utwente.nl) Department of Applied Mathematics, University of Twente, Enschede, The Netherlands.

<sup>§</sup>[{nmegow,noelke}@uni-bremen.de](mailto:{nmegow,noelke}@uni-bremen.de) Faculty of Mathematics and Computer Science, University of Bremen, Germany.

<sup>¶</sup>[kschewior@gmail.com](mailto:kschewior@gmail.com) Department of Mathematics and Computer Science, Universität zu Köln, Cologne, Germany.

<sup>||</sup>[bertrand.simon@cc.in2p3.fr](mailto:bertrand.simon@cc.in2p3.fr) IN2P3 Computing Center, CNRS, Villeurbanne, France.

their speed is a priori unknown. In the first stage, the task is to group jobs into at most  $m$  bags. In the second stage, the machine speeds  $s_i \geq 0$ , for  $i \in \{1, \dots, m\}$ , are revealed. The time needed to execute job  $j$  on machine  $i$  is  $\frac{p_j}{s_i}$ , if  $s_i > 0$ . If a machine has speed  $s_i = 0$ , then it cannot process any job; we say the machine *fails*. Given the machine speeds, the second-stage task is to assign bags to the machines such that the makespan is minimized, where the makespan is the maximum sum of execution times of jobs assigned to the same machine.

Given a set of bags and machine speeds, the second-stage problem emerges as classical makespan minimization on related parallel machines. It is well-known that this problem can be solved arbitrarily close to optimality by polynomial-time approximation schemes [2]. As we are interested in the information-theoretic tractability, we allow superpolynomial running times for our algorithms and assume that the second-stage problem is solved optimally. Thus, an *algorithm* for speed-robust scheduling defines a job-to-bag allocation, i.e., it gives a solution to the first-stage problem.

We evaluate the performance of algorithms by a worst-case analysis comparing the makespan of the algorithm with the optimal makespan achievable when all machine speeds are known in advance. We say that an algorithm is  $\rho$ -robust if, for any instance, its makespan is within a factor  $\rho \geq 1$  of the optimal solution. The *robustness factor* of the algorithm is defined as the infimum over all such  $\rho$ .

The special case of speed-robust scheduling where all machine speeds are either 0 or 1 has been studied previously by Stein and Zhong [3]. They introduced the problem with identical machines and an unknown number of machines that fail (speed 0) in the second stage.

## 2 Our Contribution

We introduce the speed-robust scheduling problem and present robust algorithms. The algorithmic difficulty of this problem is to construct bags in the first stage that are robust under any choice of machine speeds in the second stage.

The straightforward approach of using *any* makespan-optimal solution on  $m$  identical machines is not sufficient to achieve a bounded robustness factor since we show that such a makespan-minimizing algorithm might have an arbitrarily large robustness factor. However, Longest Processing Time first (LPT) does the trick and is  $(2 - \frac{1}{m})$ -robust for arbitrary job sizes. However, we also show that LPT, since it aims at “balancing” the bag sizes, cannot lead to a better robustness factor than  $2 - \frac{1}{m}$ . Hence, to improve upon this factor, we need to carefully construct bags with imbalanced bag sizes. There are two major challenges with this approach: (i) finding the ideal imbalance in the bag sizes independent from the actual job processing times that would be robust for all adversarial speed settings simultaneously and (ii) adapting bag sizes to accommodate discrete jobs.

A major contribution of this paper is an optimal solution to the first challenge by considering infinitesimal jobs. One can think of this as filling bags with *sand* to the desired level. In this case, the speed-robust scheduling problem boils down

	General speeds		Speeds from $\{0, 1\}$	
	Lower bound	Upper bound	Lower bound	Upper bound
Discrete jobs (Rocks)	$\bar{\rho}(m)$	$2 - \frac{1}{m}$	$\frac{4}{3}$ by [3]	$\frac{5}{3}$ by [3]
Equal-size jobs (Bricks)	$\bar{\rho}(m)$	1.8	$\frac{4}{3}$ by [3]	$\frac{4}{3}$
Infinitesimal jobs (Sand)	$\bar{\rho}(m) \leq \frac{e}{e-1} \approx 1.58$		$\bar{\rho}_{01}(m) \leq \frac{1+\sqrt{2}}{2} \approx 1.207$ by [3]	

Table 1: Summary of results on speed-robust scheduling.

to identifying the best bag sizes as placing the jobs into bags becomes trivial.

These tight results for infinitesimal jobs are crucial for our further results for discrete jobs. Following the figurative notion of sand for infinitesimal jobs, we think of equal-size jobs as *bricks* and arbitrary jobs as *rocks*. Building on those ideal bag sizes, our approaches differ substantially from the methods in [3]. When all jobs have equal processing time, we obtain a 1.8-robust solution through a careful analysis of the trade-off between using slightly imbalanced bags and a scaled version of the ideal bag sizes computed for the infinitesimal setting.

When machines have only speeds in  $\{0, 1\}$  and jobs have arbitrary equal sizes, i.e., unit size, we give an optimal  $\frac{4}{3}$ -robust algorithm. This is an interesting class of instances as the best known lower bound of  $\frac{4}{3}$  for discrete jobs uses only unit-size jobs [3]. To achieve this result, we, again, crucially exploit the ideal bag sizes computed for infinitesimal jobs by using a scaled variant of these sizes. Some cases, depending on  $m$  and the optimal makespan on  $m$  machines, have to be handled individually. Here, we use a direct way of constructing bags with at most four different bag sizes, and some cases can be solved by an integer linear program. We summarize our results in Table 1.

Inspired by traditional one-stage scheduling problems where jobs have *machine-dependent execution times* (unrelated machine scheduling), one might ask for such a generalization of our problem. However, it is easy to rule out any robustness factor for such a setting: Consider four machines and five jobs, where each job may be executed on a unique pair of machines. Any algorithm must build at least one bag with at least two jobs. For this bag, there is at most one machine to which it can be assigned with finite makespan. If this machine fails, the algorithm cannot complete the jobs whereas an optimal solution can split this bag on multiple machines to get a finite makespan.

## References

- [1] Eberle, F., Hoeksma, R., Megow, N., Nölke, L., Schewior, K., Simon, B.: Speed-Robust Scheduling - Sand, Bricks, and Rocks. IPCO, 283–296 (2021).

doi:10.1007/978-3-030-73879-2\_20

- [2] Hochbaum, D.S., Shmoys, D.B.: A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach. *SIAM J. Comput.* **17**(3), 539–551 (1988). doi:10.1137/0217033
- [3] Stein, C., Zhong, M.: Scheduling when you do not know the number of machines. *ACM Trans. Algorithms* **16**(1), 9:1–9:20 (2020). doi:10.1145/3340320

# Balancing Flow Time and Energy Consumption

Shirley Zhang (Speaker) \*    Sami Davies †    Samir Khuller ‡

---

We consider the following parallel batch scheduling model: find a schedule that minimizes total flow time for  $n$  uniform length jobs, with release times and deadlines, where the machine only actively processes jobs in at most  $k$  batches of size at most  $B$ . Our work is motivated by the increasing focus in the scheduling literature on power conservation. Overall, our work is applicable in the energy minimization setting when the cost of turning on a machine is low, but the cost of keeping it running is high, such as in minimizing the fiber costs of Optical Add Drop Multiplexers and VM consolidation in cloud computing [2, 3]. Existing algorithms that minimize active time produce solutions with arbitrarily bad flow time, which are not valuable from the perspective of the client.

Let  $[n]$  denote the set of jobs, and let  $j \in [n]$  denote a single job with integral release time  $r_j$  and deadline  $d_j$ . Jobs have length  $p$ , and each job  $j$  must be scheduled at  $p$  consecutive time slots in the interval  $[r_j, d_j]$ . There is no job preemption, and once a batch has started, no other job can be processed until that batch has finished. In the unit length setting,  $p = 1$ . The total flow time of a schedule with completion times  $\{C_j\}$  is  $\sum_j (C_j - r_j)$ . We assume that time is slotted. Our goal is to find an assignment of jobs to time slots that minimizes total flow time such that at most  $k$  ‘active’ time slots process jobs with at most  $B$  jobs per active slot.

Assume jobs are ordered by release time. We focus on results with the assumption that deadlines are agreeable, i.e., for all  $i, j \in [n]$ , if  $r_i \leq r_j$  then  $d_i \leq d_j$ . Agreeable deadlines are both practical and common in related literature. For example, this assumption encompasses the case where all jobs must be completed within the same amount of time  $t$  after they are released, i.e.  $d_j = r_j + t$  for all  $j$ .

A classic and more general dynamic programming approach by Baptiste can be modified to handle active time constraints, but it is expensive, with runtime  $O(B \cdot k^2 \cdot n^7)$  and space complexity  $O(B \cdot k \cdot n^5)$  [1]. We introduce a new DP that is significantly faster and simpler than Baptiste’s. With the natural assumption that deadlines are agreeable, the reduction in runtime and space complexity between naïvely augmenting Baptiste’s DP and our DPs is notable in the uniform length

---

\*skzhang@alumni.princeton.edu. Department of Computer Science, Northwestern University, Evanston IL, USA.

†sami@northwestern.edu. Department of Computer Science, Northwestern University, Evanston IL, USA.

‡samir.khuller@northwestern.edu. Department of Computer Science, Northwestern University, Evanston IL, USA.

setting—with runtime  $O(B \cdot k \cdot n^5)$  and space  $O(k \cdot n^3)$ —and drastic in the unit length setting—with runtime  $O(B \cdot k \cdot n)$  and space  $O(k \cdot n)$ .

Additionally, we can generalize our results in both the uniform and unit length settings to when the goal is to only complete a subset of  $m \leq n$  jobs. We consider  $m$  as part of the input, and we wish to choose the  $m$  jobs that will be completed such that total flow time is minimized out of all  $\binom{n}{m}$  possible choices.

**Unit length jobs** To prove the correctness of our dynamic program for unit length jobs with agreeable deadlines, we rely on several lemmas. The first lemma shows that an instance of unit length jobs can be preprocessed such that **(i)** there are no more than  $B$  jobs with the same release time and **(ii)** a schedule for the preprocessed instance with minimal total flow time is also an optimal schedule for the original instance. Another lemma shows there exists an optimal schedule such that jobs are scheduled by increasing release time. Intuitively, this holds because given some optimal schedule that does not have this property, we can feasibly swap the time slot assignment of jobs that are not in order without changing the flow time to obtain a schedule that does satisfy this property. Our last lemma for this setting says that any optimal schedule for jobs  $[j]$  has its last active slot at  $r_j$ .

A pair  $(\alpha, j)$  represents jobs  $[j]$  ordered by release times, equipped with their release times and deadlines, that we want to schedule in at most  $\alpha$  time slots.  $\text{OPT}(\alpha, j)$  represents the minimal total flow time over all feasible schedules. We call  $(\alpha, j)$  extraneous when the number of unique release times in  $[j]$  is at most  $\alpha$ .

**Theorem 1** *Let  $(k, n)$  be an instance of unit length jobs with agreeable deadlines. One can either certify  $(k, n)$  is infeasible or find a schedule for  $(k, n)$  that minimizes flow time with a dynamic program in time  $O(B \cdot k \cdot n)$  and space  $O(k \cdot n)$ .*

Our dynamic program for this setting is the following:

If  $(\alpha, j)$  has  $B \cdot \alpha < j$ , then  $\text{OPT}(\alpha, j) = \infty$ .

If  $(\alpha, j)$  has  $B \cdot \alpha \geq j$  and is extraneous, then  $\text{OPT}(\alpha, j) = j$ .

If  $(\alpha, j)$  has  $B \cdot \alpha \geq j$  and is non-extraneous, then:

$$\text{OPT}(\alpha, j) = \min_{\substack{b \in [j-B, i_j]: \\ d_{b+1} > r_j}} \left( \text{OPT}(\alpha - 1, b) + \sum_{u=b+1}^j (r_j - r_u + 1) \right).$$

If our DP returns  $\text{OPT}(k, n) = \infty$ , it is impossible to schedule all jobs in  $[n]$  in  $k$  active time slots. Our recurrence considers the last active slot and the jobs scheduled there. We know the last active slot should be at  $r_j$ , and we know that  $j$  is scheduled at  $r_j$  along with up to  $B$  consecutive jobs directly before  $j$ . For  $b$  the latest released job not scheduled with  $j$ , we minimize over all  $b$  to find  $\text{OPT}(\alpha, j)$ .

**Uniform length jobs** One obstacle to adapting our previous DP formulation for the uniform length case is that it is no longer sufficient to schedule active slots only at the release times of jobs, as it may be the case that a release time falls

within the processing time of another batch. Therefore, we forego preprocessing here and instead add an additional parameter,  $t$ , that keeps track of the last  $p$  time slots used by an active slot. Similarly, we forego the notion of extraneous here, as even if there are enough active slots to place one at every distinct release time, this does not necessarily lead to a feasible schedule.

We consider instances  $(\alpha, j, t)$  to be feasible if all jobs in  $[j]$  can be scheduled in at most  $\alpha$  active time slots, where the last  $p$  active time slots cover  $[t, t + p]$ . For feasible  $(\alpha, j)$ , we have that  $\text{OPT}(\alpha, j) = \min_t \text{OPT}(\alpha, j, t)$ .

**Theorem 2** *Let  $(k, n)$  be an instance of uniform length jobs with agreeable deadlines. One can either certify  $(k, n)$  is infeasible or find a schedule for  $(k, n)$  that minimizes flow time with a dynamic program in time  $O(B \cdot k \cdot n^5)$  and space  $O(k \cdot n^3)$ .*

Recall jobs are ordered by release times, and let  $T = \{r_j + p \cdot u\}$  for  $j \in [n]$  and  $0 \leq u \leq n$ . Then the following holds for all  $t \in T \cup \{\min(T) - p\}$ ,  $0 \leq \alpha \leq k$ , and  $0 \leq j \leq n$ :

If  $j > 0$  and either  $t < r_j$  or  $t + p > d_j$ , then  $\text{OPT}(\alpha, j, t) = \infty$ .

If  $B \cdot \alpha/p < j$ , then  $\text{OPT}(\alpha, j, t) = \infty$ .

For  $\alpha \geq 0$  and  $t \geq -p + 1$ ,  $\text{OPT}(\alpha, 0, t) = 0$ .

Otherwise,

$$\text{OPT}(\alpha, j, t) = \min_{\substack{b \in [j-B, j-1]: \\ d_{b+1} \geq t+p}} \min_{t' \leq t-p} \left( \text{OPT}(\alpha - p, b, t') + \sum_{u=b+1}^j (t - r_u + p) \right).$$

Then  $\text{OPT}(\alpha, j) = \min_t \text{OPT}(\alpha, j, t)$ .

There are several differences between the DP for the uniform length case and the earlier unit length one. Most notable is the addition of parameter  $t$ , which is necessary because it is no longer true that all active slots are scheduled at release times. We also need to consider the possibility of active slots being scheduled immediately after another batch finishes processing. Since it takes  $p$  active slots to complete a batch, the number of active slots goes down by  $p$  every time we schedule a batch. We also need to ensure that we are able to fully schedule  $[j]$  if job  $j$  starts processing at time  $t$ .

## References

- [1] BAPTISTE, P. Batching identical jobs. *Math. Methods Oper. Res.* 52, 3 (2000), 355–367.
- [2] CHANG, J., KHULLER, S., AND MUKHERJEE, K. LP rounding and combinatorial algorithms for minimizing active and busy time. *J. Sched.* 20, 6 (2017), 657–680.
- [3] KOEHLER, F., AND KHULLER, S. Busy time scheduling on a bounded number of machines (extended abstract). In *Workshop on Algorithms and Data Structures WADS* (2017), Springer, pp. 521–532.

# Speed Scaling with Explorable Uncertainty\*

Evripidis Bampis <sup>†</sup>    Konstantinos Dogeas <sup>‡</sup>    Alexander Kononov <sup>§</sup>  
Giorgio Lucarelli (Speaker) <sup>¶</sup>    Fanny Pascual <sup>||</sup>

---

## 1 Introduction

Speed scaling is a standard and well-known mechanism to handle energy consumption in computing systems [7]. According to this mechanism, the speed of a machine can be modified by the scheduler in order to save energy. Specifically, we assume that if the machine at a time  $t$  runs at speed  $s(t)$ , then the power needed is  $P(s(t))$ . We study here the general case where the power is described by the function  $P(s(t)) = s(t)^\alpha$ , where  $\alpha > 1$  is considered to be constant. Then, the energy consumption is computed as  $E = \int P(s(t))dt$ .

In real systems, the characteristics of the jobs may not be known in advance. However, in some situations it is possible to obtain the exact job characteristics at some extra cost. The operation that allows to obtain the exact value of some part of the input is called a *query*. Kahan [6] was the first to formalize this notion known as *explorable uncertainty*. Since then, a series of problems have been studied (e.g. see the survey [5]). In most of these works, the aim is the minimization of the number of queries needed to produce the desired solution. In this paper, we introduce a model for speed scaling problems, inspired by the model introduced recently for classical scheduling problem under explorable uncertainty in [4]. In the model of Dürr et al. [4], the uncertain information concerns the processing time of each job for which an upper bound is known in advance. It is possible to learn the exact processing time by querying at the price of a unit cost. If a job is executed without a query, then its execution time is equal to its upper bound. Contrary to the previous approaches, queries are executed directly on the machine

---

\*This work has been presented in SPAA 2021.

<sup>†</sup>evripidis.bampis@lip6.fr. Sorbonne Université, CNRS, LIP6, 4 Place Jussieu, Paris 75005, France.

<sup>‡</sup>konstantinos.dogeas@lip6.fr. Sorbonne Université, CNRS, LIP6, 4 Place Jussieu, Paris 75005, France.

<sup>§</sup>alvenko@math.nsc.ru. Novosibirsk State University, Sobolev Institute of Mathematics, 1 Thörvöld Circle, Novosibirsk, Russia.

<sup>¶</sup>giorgio.lucarelli@univ-lorraine.fr. Université de Lorraine, LCOMS, Metz 57000, France.

<sup>||</sup>fanny.pascual@lip6.fr. Sorbonne Université, CNRS, LIP6, 4 Place Jussieu, Paris 75005, France.

running the jobs and so it is important to balance the time spent on queries and the time spent on the execution of jobs. More recently, an extension of this model has been considered in [2], where the querying times are job-dependent.

In this paper, we propose the study of the following natural extension of this model in the speed scaling setting: each job has a release time  $r_j$ , a deadline  $d_j$  and an unknown workload  $w_j^*$  that can be revealed to the algorithm only after executing a query that induces a given additional job-dependent load  $q_j$ . Alternatively, the job may be executed without any query, but in that case its workload is equal to a given upper bound  $w_j$ . This assumption is motivated by the fact that a query could correspond to a code optimizer which needs some workload to process the job and potentially reduces its workload [4]. Another possible application for this assumption is file compression. The objective is to minimize the total energy consumption for executing all the jobs in their time windows (between their release dates and deadlines). We also consider the related problem of minimizing the maximum speed used by the algorithm. We call the above enhanced model as *Query-Based Speed-Scaling* model (QBSS).

## 2 Results

The QBSS model is online by nature, since the value of  $w_j^*$  for each job  $j$  is revealed only after the potential execution of the query  $q_j$ . However, we distinguish between the *offline* and the *online* versions with respect to the classical scheduling setting. In the offline version, the entire input is known in advance, i.e., the total number of jobs to be scheduled, as well as their characteristics, except for the exact loads  $w_j^*$ . In the online version, the input becomes available to the algorithm over time: at time  $t = r_j$ , a new job  $j$  and its characteristics are revealed, except again for its exact load  $w_j^*$ . In other words, the algorithm does not know in advance how many jobs it has to schedule, at which time they will arrive or what their characteristics are. In both cases, if the exact load of a job  $j$  becomes known at the same time as its other characteristics, then the QBSS model reduces to the classical speed scaling setting, since the scheduler can simply decide whether to make the query for  $j$  or not based on the value of  $\min\{w_j, q_j + w_j^*\}$ . We study both versions of the problem.

There are two additional questions to answer for each job  $j$  in the QBSS model: whether the query will be done or not, and, if yes, how to partition the active interval of the job among the execution of its query and its exact load. Both decisions have a crucial impact on the speeds and on the consumed energy. For the first question, doing always the query leads to constant approximation algorithms, whereas never doing it leads to unbounded ratios. However, in most cases a better decision can be made by comparing the values of  $q_j$  and  $\frac{w_j}{\phi}$ , where  $\phi \approx 1.6180$  is the golden ratio. Note that the optimal algorithm has complete knowledge of the instance, including the exact loads. Hence, it can take this decision by comparing  $w_j$  and  $q_j + w_j^*$ . For the second question, the algorithm has to determine a *splitting point*  $\tau_j = r_j + x(d_j - r_j)$ , with  $0 < x < 1$  so as  $\tau_j \in (r_j, d_j)$ , indicating the latest

time at which the query has to finish execution and the earliest time at which the exact work of  $j$  may start its execution.

We introduce the notion of *equal window algorithms* according to which the active interval of a job is split in two equal sub-intervals: the query is executed in the first half, and the exact work in the second half. This is motivated by an instance consisting of a single job, where a different splitting leads to stronger lower bounds. A further discussion, as well as several lower bounds for the offline version of our model when a single machine is available are also provided, where the use of randomization or oracles that answers optimally to one of the questions above are explored.

For the offline case where all jobs have a common release date, we present a series of results based on different assumptions on the deadlines. Specifically, if all jobs have a common deadline, we propose an algorithm (CRCD) which achieves a 2-approximation ratio with respect to maximum speed and a  $\min\{2^{\alpha-1}\phi^\alpha, 2^\alpha\}$ -approximation ratio with respect to energy. A better analysis is also given for special values of  $\alpha$ . Moreover, we consider the case where all deadlines are powers of two and we propose a  $(4\phi)^\alpha$ -approximation algorithm (CRP2D) with respect to energy. We extend the previous result for arbitrary deadlines and we obtain an approximation ratio of  $(8\phi)^\alpha$  (algorithm CRAD) by rounding down the deadlines of the instance to the closest power of two.

For the online case, we adapt the well-known AVR [7] and BKP [3] online algorithms for the classical speed scaling setting to the QBSS model. The competitive ratio of our algorithms (AVRQ and BKPQ) has an additional multiplicative factor with respect to their version in the classical setting: a factor of  $2^\alpha$  for AVRQ in which the query is made for all jobs, and a factor of  $(2 + \phi)^\alpha$  for BKPQ in which the query is decided based on the golden ratio. Note that, BKPQ is also  $(2 + \phi)e$ -competitive with respect to maximum speed. Finally, we study the QBSS model on parallel identical machines and we propose a modification of the algorithm AVR(m) [1], which turns out to be  $2^\alpha(2^{\alpha-1}\alpha^\alpha + 1)$ -competitive with respect to energy.

Our results are summarized in the following table.

		Energy	
		Lower Bound	Upper Bound
Offline	Oracle	$\phi^\alpha$	-
	CRCD	$\max\{\phi^\alpha, 2^{\alpha-1}\}$	$\min\{2^{\alpha-1}\phi^\alpha, 2^\alpha\}$
	CRP2D		$(4\phi)^\alpha$
	CRAD		$(8\phi)^\alpha$
Online	AVRQ	$(2\alpha)^\alpha$	$2^\alpha 2^{\alpha-1} \alpha^\alpha$
	BKPQ	$3^{\alpha-1}$	$(2 + \phi)^\alpha 2 \left(\frac{\alpha}{\alpha-1}\right)^\alpha e^\alpha$
	AVRQ(m)	$(2\alpha)^\alpha$	$2^\alpha (2^{\alpha-1} \alpha^\alpha + 1)$

## Acknowledgments

This work was partially supported by the French National Research Agency (Energumén ANR-18-CE25-0008 and Algoridam ANR-19-CE48-0016). The research of the third author was supported by the program of fundamental scientific researches of the SB RAS, project 0314-2019-0014.

## References

- [1] SUSANNE ALBERS, ANTONIOS ANTONIADIS, GERO GREINER (2015). *On multi-processor speed scaling with migration*. J. Comput. Syst. Sci., 81:1194–1209.
- [2] SUSANNE ALBERS AND ALEXANDER ECKL (2020). *Explorable Uncertainty in Scheduling with Non-Uniform Testing Times*. WAOA, pp. 127–142.
- [3] NIKHIL BANSAL, TRACY KIMBREL, KIRK PRUHS (2007). *Speed Scaling to Manage Energy and Temperature*. J. ACM, vol. 54, article 3.
- [4] CHRISTOPH DÜRR, THOMAS ERLEBACH, NICOLE MEGOW AND JULIE MEISSNER (2018). *Scheduling with Explorable Uncertainty*. ITCS, pp. 30:1–30:14.
- [5] THOMAS ERLEBACH AND MICHAEL HOFFMANN (2015). *Query-Competitive Algorithms for Computing with Uncertainty*. Bulletin of the EATCS, vol. 116.
- [6] SIMON KAHAN (1991). *A Model for Data in Motion*. STOC, pp. 267–277.
- [7] F. FRANCES YAO, ALAN J. DEMERS, SCOTT SHENKER (1995). *A Scheduling Model for Reduced CPU Energy*. FOCS, pp. 374–382.

# Allocation and pricing of planetarium seats

Freija van Lent (Speaker) \*    Julian Golak †    Alexander Grigoriev ‡

---

## 1 Introduction and Related Literature

We consider a planetarium allocating and selling  $m$  seats to  $n$  customers. Every customer naturally wants to sit in the middle of the planetarium, to have the better view, and she has an individual valuation  $v_i$ ,  $i = 1, \dots, n$ , for this central spot  $j^*$ . The planetarium determines the prices of the seats  $p_j$ ,  $j = 1, \dots, m$ , and allocates the customers to them. The utility of a customer  $i$  linearly decreases in price and distance from  $j^*$  to the seat  $j$  the customer is allocated to, i.e.,  $u_{ij} = \max\{v_i - p_j - \alpha_i d(j, j^*), 0\}$ , where  $d(j, j^*)$  is the distance from  $j$  to  $j^*$  and  $\alpha_i > 0$  is an individual distance deterioration rate of customer  $i$ . We assume non-negativity of the utility function because, for high prices and high distances to the central spot, a customer becomes insensitive to pricing and allocation as it is equivalent to not going to the planetarium. It would also be natural to assume that higher valuations for the central seat imply higher distance deterioration rates for the customers. A customer  $i$  having non-negative utility  $u_{ij} \geq 0$  for seat  $j$  where she is allocated to, pays the price  $p_j$  to the planetarium.

The economic motivation and practical application behind the problem is very intuitive and it comes from the concept of fairness of allocation and pricing. When a customer in a theater or cinema or planetarium observes a more preferred seat at a cheaper price than she is paying, this customer naturally becomes jealous of another seat. Generally, this would be regarded as unfair. Avoiding this unfairness is done by introducing envy-free pricing. The concept of envy-free pricing was, for instance, addressed in [1] and it is known to be a competitive or Walrasian equilibrium [7]. Particularly, in [1], the authors first define the problem of optimal envy-free pricing, then they show that the problem is *APX*-hard and design an  $O(\log n)$ -approximation algorithm for solving it for  $n$  customers. Many contemporary articles address a variety of special cases of the optimal envy-free pricing problem, stating complexity, and positive and negative approximability results, see [5, 3, 4, 2, 8]. Compared to the previous literature, we consider a special case

---

\*[f.vanlent@maastrichtuniversity.nl](mailto:f.vanlent@maastrichtuniversity.nl). Department of Data Analytics and Digitalisation, Maastricht University, P.O.Box 616, 6200 MD Maastricht, The Netherlands

†[julian.golak@tuhh.de](mailto:julian.golak@tuhh.de). Institute for Algorithms and Complexity, Hamburg University of Technology, Blohmstr. 15, HIPOne, 21079 Hamburg, Germany

‡[a.grigoriev@maastrichtuniversity.nl](mailto:a.grigoriev@maastrichtuniversity.nl). Department of Data Analytics and Digitalisation, Maastricht University, P.O.Box 616, 6200 MD Maastricht, The Netherlands

of the optimal envy-free allocation and pricing problem where the utilities of the customers are determined by geometric properties of customers' allocations over the plane.

## 2 Model definition

We assume the planetarium is the entire 2-dimensional plane and that the seats are points  $x \in R^2$  in that plane. In this case, the planetarium searches for a pricing strategy  $p(x) : R^2 \rightarrow R^+$  on the plane. In turn, the utility of a customer  $i = 1, \dots, n$  allocated to a point  $x \in R^2$  is determined by  $u_i(x) = \max\{v_i - p(x) - \alpha_i d(x, \mathbf{0}), 0\}$ , where the origin  $\mathbf{0}$  is the most preferred place to be seated and  $d(x, y)$  is the Euclidean distance between points  $x$  and  $y$ . Notice that for any point  $x \in R^2$  such that  $d(x, \mathbf{0}) \geq \max_i v_i / \alpha_i$ , the utility of every customer is 0. We refer to all points equidistant from the origin as *ranges* of seats, and the number of points in a range we refer to as the *capacity* of a range. Customers are assumed to be indifferent between any two points in any range.

Let  $x_i, i = 1, \dots, n$ , be the distance determining the range customer  $i$  is allocated to. The goal of the planetarium is to assign the customers in an envy-free allocation that maximizes the total revenue. The complete model of the continuous case, called the *continuous concentric envy-free pricing and allocation problem*, reads

$$\sup_{p,x} \sum_{i=1}^n p(x_i) \tag{1}$$

subject to

$$u_i(x_i) \geq u_i(x) \forall x > 0. \tag{2}$$

In the 2-dimensional continuous case, there is a continuum number of points equidistant from the origin and the capacity of a range is therefore unlimited. We assume that the origin cannot be occupied by any customer as its capacity of 1 is inconsistent with that of other ranges. As a result, the objective of the model is to find the supremum.

The *maximum price* the customer is willing to pay for an allocation at point  $x$  is denoted by  $\ell_i(x)$ . This function  $\ell_i(x)$  can be computed by setting the utility  $u_i(x) = 0$ , therefore  $\ell_i(x) = v_i - \alpha_i x$ . The *maximum willingness-to-pay*  $\ell_i(x)$ , is later also referred to as maxWTP.

## 3 Research contribution

**Continuous Concentric Envy-Free Pricing and Allocation problem** For all customers having the same preferred point in the planetarium, we introduce a dynamic programming algorithm solving the problem in polynomial time. First, we show that pricing and allocation is optimal at the intersections of any two curves in the maximum willingness-to-pay graph. Then, we propose our exact dynamic programming algorithm. The algorithm iterates backwards over all intersections

of maxWTP-curves, and compares possible pricing and allocation points of one or more customers in current and previous intersections. The running time of the algorithm is  $O(n^2)$ .

**Discrete Concentric Envy-Free Pricing and Allocation problem** We also consider a discrete variant of the problem in which seats are represented by tiles of a regular tessellation of the plane, e.g., squares of the grid or hexagons in the honeycomb. In this discrete setting, the distance between two seats is the Euclidean distance between the center points of the respective tiles. We solely consider arrangements of seats in which the distance between the preferred seat and any other seat is at most some constant  $k$ . This implies a circular seat arrangement of radius  $k$ . To determine the capacity of a *range* in this setting, we use the coefficients of the theta series for square lattice for the squares of the grid alignment. If the seats are considered to be hexagons in the honeycomb, the coefficients of the theta series for the  $A_2$  triangular lattice specify the capacities for the ranges.

To solve this setting of the problem, we introduce a lowering step size of  $\lambda$  and a variable  $l$  that denotes how often a valuation curve has been lowered. If we lower customer  $i$ 's valuation curve  $l$  times, we consider the curve  $\ell_i^l = (v_i - l\lambda) - \alpha_i x$  that is parallel to the original valuation curve. We propose a dynamic programming algorithm that evaluates all customers' WTP curves at different heights and selects a combination of curves that corresponds to an allocation of customers maximizing the planetarium's revenue. The proposed algorithm runs in  $O(mnLC)$ , where  $L = \lfloor \max_i v_i / \lambda \rfloor$  and  $C = \max_j \text{cap}(j)$  with  $\text{cap}(j)$  denoting the capacity of seat range  $j$ , and provides an optimal solution for  $\lambda \downarrow 0$ .

**Future research** To make the work more appealing to practitioners, we are considering variations of the proposed problem. The variations we are evaluating include more complex alignments of seats, allocation and pricing in an online setting, and the quality of simple heuristics.

## References

- [1] GURUSWAMI, V., HARTLINE, J. D., KARLIN, A. R., KEMPE, D., KENYON, C., AND MCSHERRY, F. (2005, January). *On profit-maximizing envy-free pricing*. In SODA (Vol. 5, pp. 1164-1173).
- [2] CHEN, N., GHOSH, A., AND VASSILVITSKII, S. (2011). *Optimal envy-free pricing with metric substitutability*. SIAM Journal on Computing, 40(3), 623-645.
- [3] CHEN, N., AND DENG, X. (2014). *Envy-free pricing in multi-item markets*. ACM Transactions on Algorithms (TALG), 10(2), 1-15.

- [4] CHEN, N., DENG, X., GOLDBERG, P. W., AND ZHANG, J. (2016). *On revenue maximization with sharp multi-unit demands*. Journal of Combinatorial Optimization, 31(3), 1174-1205.
- [5] BRIEST, P. (2008, July). *Uniform budgets and the envy-free pricing problem*. In International Colloquium on Automata, Languages, and Programming (pp. 808-819). Springer, Berlin, Heidelberg.
- [6] CONWAY, J. H., AND SLOANE, N. J. A. (2013). *Sphere packings, lattices and groups* (Vol. 290). Springer Science & Business Media.
- [7] NISAN, NOAM AND ROUGHGARDEN, TIM AND TARDOS, ÉVA AND VAZIRANI, VIJAY V. (2007) *Algorithmic Game Theory*. Cambridge University Press.
- [8] FELDMAN, M., FIAT, A., LEONARDI, S., AND SANKOWSKI, P. (2012, June). *Revenue maximizing envy-free multi-unit auctions with budgets*. In Proceedings of the 13th ACM conference on electronic commerce (pp. 532-549).

# Towards hardness of approximate pure equilibria

Vipin Ravindran Vijayalakshmi \*      Alexander Skopalik (Speaker) †

---

## 1 Introduction

The complexity of computing equilibria has been an important problem which has been extensively studied since the beginning of algorithmic game theory. For both mixed [2] and pure [1] equilibria it turned out that computing an equilibrium is hard for the classes PPAD and PLS, respectively. Consequently, research focused on approximate equilibria. In this work we focus on pure equilibria which are guaranteed to exist by a potential function argument in potential games for which we know that they are isomorphic to the class of congestion games. The first and in fact the only inapproximability result was obtained by Skopalik and Vöcking [3]. They show that computing  $\alpha$ -approximate pure Nash equilibrium is PLS-complete for any  $\alpha > 1$ . Their reduction however uses very artificial cost functions. On the positive side, Chien and Sinclair [4] showed that for symmetric<sup>1</sup> congestion games with mild restrictions on the cost-functions,  $(1 + \epsilon)$ -approximate equilibria can be computed efficiently. However, their result crucially requires symmetric players. For asymmetric games, Caragiannis et al. [5] present a polynomial-time algorithm that computes  $(2 + \epsilon)$ -approximate pure Nash equilibria for linear cost functions (with non-negative coefficients). This result extends to polynomial cost functions albeit with larger approximation factor. In [6], the authors improved the approximation guarantee to  $(1.61 + \epsilon)$  for linear functions. They also give improved bounds for polynomial cost functions.

To the best of our knowledge there is currently no lower bound known on the smallest value of  $\alpha$  for which an  $\alpha$ -approximate pure Nash equilibrium can be computed in polynomial time in games with linear or polynomial cost functions. The lower bound proof of [3] crucially relies on cost function that are step functions which jump from 0 to a very large value.

**The Model.** A congestion game denoted by  $G = (\mathcal{N}, E, (S_u)_{u \in \mathcal{N}}, (f_e)_{e \in E})$  consists of a set of  $N$  players,  $\mathcal{N} = \{1, 2, \dots, N\}$ , who compete over a set of resources,  $E = \{e_1, e_2, \dots, e_m\}$ . Each player  $u \in \mathcal{N}$  has a set of strategies denoted by  $S_u \subseteq 2^E$ . Each resource  $e \in E$  has a non-negative and non-decreasing cost

---

\*vipin.rv@oms.rwth-aachen.de. RWTH Aachen University, Germany.

†a.skopalik@utwente.nl. Mathematics of Operations Research, University of Twente, Netherlands.

<sup>1</sup>A game is called symmetric if the set of available strategies is identical for all players.

function  $f_e : \mathbb{N} \mapsto \mathbb{R}_+$  associated with it. Let  $n_e(s)$  denote the number of players on a resource  $e \in E$  in the strategy profile or state  $s$ , then the cost contributed by a resource  $e \in E$  to each player using it is denoted by  $f_e(n_e(s))$ . Therefore, the cost of a player  $u \in \mathcal{N}$  in a state  $s = (s_1, \dots, s_N)$  of the game is given by  $c_u(s) = \sum_{e \in E: e \in s_u} f_e(n_e(s))$ . For a state  $s$ ,  $c_u(s'_u, s_{-u})$  denotes the cost of player  $u$ , when only  $u$  deviates. A state  $s \in S$  is a pure Nash equilibrium (PNE), if there exists no player who could deviate to another strategy and decrease their cost, i.e.,  $\forall u \in \mathcal{N}$  and  $\forall s'_u \in S_u$ ,  $c_u(s) \leq c_u(s'_u, s_{-u})$ . It is well known that a congestion game always exhibits a PNE. A weaker notion of PNE is the  $\alpha$ -approximate pure Nash equilibrium for some  $\alpha > 1$ , which is a state  $s$  in which no player has an improvement that decreases their cost by a factor of at least  $\alpha$ , i.e.,  $\forall u \in \mathcal{N}$  and  $\forall s'_u \in S_u$ ,  $\alpha \cdot c_u(s'_u, s_{-u}) \geq c_u(s)$ .

## 2 Our Contribution

In an effort to make a step towards showing an inapproximability lower bound for games with linear cost functions, we provide a hardness proof for a related problem. We believe that our techniques are also useful for proving hardness of the actual problem. We study the complexity of the reachability problem in congestion games with linear cost functions. That is, we ask for the computation of the equilibrium that is reachable by approximate improvement step from a given initial strategy profile.

We know that this problem is hard for exact equilibria and also for approximate equilibria for games with arbitrary cost functions. Both are a consequence of the PLS-reductions in [1] and [3] which imply PSPACE-hardness of the reachability problem.

**Theorem 1** *Given a linear congestion game and an initial state  $s$ , it is PSPACE-complete to find a 1.06-approximate equilibrium that is reachable by a sequence of 1.06-improving moves from state  $s$ .*

We note that our result does not imply PLS-hardness of computing a 1.06-approximate equilibrium. Such a result is still a major open problem. In the following we give a short overview of our proof and outline the major technical challenges that one has to overcome to prove PLS-hardness.

Our construction is a reduction from the canonical PSPACE-hard problem: Decide for a given polynomially space-bounded Turing machine  $M$ , whether it stops. For a given TM  $M$ , we construct a congestion game that consists of two major parts. It contains a collection of  $k$  gadgets, where  $k$  corresponds roughly to the space bound of the TM  $M$ . The gadgets are organized in  $k$  levels. Each gadget consists of main player and several auxiliary players. The main player has 9 strategies and our construction ensures that in every improvement sequence, they visit them in a certain order. We call such a sequence of the improvement steps of a main player as a run of the gadget. In every run of a gadget in level  $i$ , the

gadget of level  $i - 1$  is triggered twice to perform a run. This eventually leads to exponentially many runs of the gadget in level 1.

The second part of our construction is a gadget which simulates the TM  $M$ . This gadget is triggered by the gadget at level 1 described above and each time simulates one step of the TM. By choosing  $k$  sufficiently large enough, we make sure that we have simulated enough steps of  $M$ . From the equilibrium, we can easily read whether  $M$  has stopped. If it hasn't, we know it never will as it must have already visited a configuration twice and hence, loops forever.

### 3 Towards PLS-hardness

To prove PLS-hardness, the authors of [3] reduced from the problem **CIRCUITFLIP**: Given a Boolean circuit with input bits  $x_1, \dots, x_n$  and output bits  $y_1, \dots, y_m$  (which are interpreted as a binary number  $y$ ), find a vector of input bits that is locally optimal (with respect to  $y$ ). That is, there is no bit of  $x$  that can be flipped to decrease  $y$ . To that end, they simulate a circuit by players that simulate the gates and inputs of the circuit. A major problem that they identify and solve is the so called feedback-loop problem: The costs of the players that simulate the gates of the lower levels of the circuit is not sufficient to initiate strategy changes of the players that represent the input bits.

They overcome this problem by powering the circuit with a construction that is similar to the first part of our construction. However, note that they prove PLS-hardness of computing some equilibrium, whereas we show PSPACE-hardness of computing the specific equilibrium reachable from a specific initial profile. Hence, the PLS-reduction needs to ensure that there are no unwanted equilibria that do not correspond to local optima of the **CIRCUITFLIP** problem. In particular, they needed to ensure that the construction that powers the circuit, can trigger sufficiently many improvement steps to search for a local optimum. To guarantee that, they make sure that the binary counter powering the simulation of the circuit, has a value larger than the output of the circuit.

To ensure that there are no unwanted equilibria, e.g., where the counter has value 0 but  $x$  is not a local optimum, they introduced *blocking* players and *blocking* resources. The latter are resource with cost functions that have cost of 0 for one player and extremely high costs for two or more players. With the help of such players they guarantee that the game is either in a *legal* state, or there is a player with high costs. For the latter they ensured that every sequence of improvement steps leads to a legal state.

For our construction, we needed a similar mechanism. But we were able to achieve this with linear cost functions. The key idea was to keep the interaction between players local. That is, only players in neighboring levels interact. We believe that a PLS-hardness proof for linear or polynomial cost functions would have to have a similar property.

## References

- [1] ALEX FABRIKANT, CHRISTOS PAPADIMITRIOU, AND KUNAL TALWAR. The complexity of pure Nash equilibria. In Proceedings of STOC 2004.
- [2] CONSTANTINOS DASKALAKIS, PAUL GOLDBERG, CHRISTOS PAPADIMITRIOU. The complexity of computing a Nash equilibrium. In Proceedings of STOC 2006.
- [3] ALEXANDER SKOPALIK AND BERTHOLD VÖCKING. Inapproximability of pure Nash equilibria. In Proceedings of STOC 2008.
- [4] STEVE CHIEN AND ALISTAIR SINCLAIR. Convergence to approximate Nash equilibria in congestion games. *Games and Economic Behavior* 71(2), 2011.
- [5] IOANNIS CARAGIANNIS, ANGELO FANELLI, NICK GRAVIN, AND ALEXANDER SKOPALIK. Efficient computation of approximate pure Nash equilibria in congestion games. In Proceedings of FOCS 2011.
- [6] VIPIN RAVINDRAN VIJAYALAKSHMI AND ALEXANDER SKOPALIK. Improving Approximate Pure Nash Equilibria in Congestion Games. In Proceedings of WINE 2020.

# Scheduling Games with Rank-Based Utilities

Shaul Rosner (Speaker) \*

Tami Tamir<sup>†</sup>

---

## 1 Introduction

In traditional job scheduling games, every job corresponds to a player whose strategy space is the set of machines. The goal of a player is to minimize its cost, given by the completion time of its machine. In this work we consider a different measure for a player's utilization, which fits environments with strong competition among the participants.

In our *scheduling game with rank-based utilities* (SRBG, for short), the jobs are partitioned into *competition sets*. The primary goal of a job is to minimize the *rank* of its completion time relative to its competitors, while minimizing the completion time itself is a secondary objective.

Our model is motivated by studies in behavioural science, where it is shown that individuals are concerned primarily about their relative welfare, rather than their absolute well-being. For example, workers care more about their relative salary than they do about their actual salary, and happiness depends not just on absolute, but mainly on relative consumption. Similarly, racing drivers are ready to spend valuable time in obstructing other drivers, and a company may be willing to take a hit to its income if its competitors are hurt more significantly. The participants' ranking is crucial in numerous additional fields, including auctions with a limited number of winners, where, again, the participants' rank is more important than their actual offer, transplant queues, sport leagues, and even submission of papers to competitive conferences.

Games in which the the players utilities are based also on their ranking were studied in [1, 3]. These papers consider a different model or analyzed different problems.

### 1.1 Problem Description

A *scheduling game with rank-based utilities* (henceforth referred to as a SRBG) is given by  $G = \langle \mathcal{J}, \mathcal{M}, \{p(j)\}_{j \in \mathcal{J}}, \{d_i\}_{i \in \mathcal{M}}, S \rangle$ , where  $\mathcal{J}$  is a set of  $n$  jobs,  $\mathcal{M}$  is a set of  $m$  machines,  $p(j)$  is the *length* of job  $j$ ,  $d_i$  is the *delay* of machine  $i$ , and  $S$  is a partition of the jobs into *competition sets*. Specifically,  $S = \{S_1, \dots, S_c\}$  such

---

\*shaul.rosner@post.idc.ac.il. School of Computer Science, Reichman University, Israel

<sup>†</sup>tami@idc.ac.il. School of Computer Science, Reichman University, Israel

that  $c \leq n$ ,  $\cup_{\ell=1}^c S_\ell = \mathcal{J}$ , and for all  $\ell_1 \neq \ell_2$ , we have  $S_{\ell_1} \cap S_{\ell_2} = \emptyset$ . For every job  $j \in S_\ell$ , the other jobs in  $S_\ell$  are denoted the *competitors* of  $j$ . Let  $n_\ell$  denote the number of jobs in  $S_\ell$ . A profile of a SRBG is a schedule  $s = \langle s_1, \dots, s_n \rangle \in \mathcal{M}^n$  describing the machines selected by the players.

For a profile  $s$ , the *rank* of player  $j \in S_\ell$  in profile  $s$ , denoted  $rank_j(s)$  is the rank of its completion time among its competitors. If several players in a competition set have the same completion time, then they all have the same rank, which is the corresponding median value. For example, if  $n_\ell = 4$  and the completion times of the jobs in  $S_\ell$  are  $(7, 8, 8, 13)$  then the players' ranks are  $\langle 1, 2.5, 2.5, 4 \rangle$ , and if all players in  $S_\ell$  have the same completion time then they all have rank  $(n_\ell + 1)/2$ . Note that, independent of the profile,  $\sum_{j \in S_\ell} rank_j(s) = n_\ell(n_\ell + 1)/2$ .

The objective of every player is to minimize its rank. A profile is a pure Nash equilibrium (NE) if no player can improve its objective by unilaterally deviating from its current strategy, assuming that the strategies of the other players do not change. Best-Response Dynamics (BRD) is a local-search method where in each step some player is chosen and plays its best improving deviation (if one exists).

Some of our results consider SRBGs with *homogeneous* competition sets. We denote by  $\mathcal{G}_h$  the corresponding class of games. Formally,  $G \in \mathcal{G}_h$  if, for every  $1 \leq \ell \leq c$ , all the jobs in  $S_\ell$  have the same length,  $p_\ell$ .

The following example summarizes the model and demonstrates several of the challenges in analyzing SRBGs. Consider a game  $G \in \mathcal{G}_h$  on 3 identical machines, played by 9 jobs in two homogeneous competition sets.  $S_1$  consists of four jobs having length 4, and  $S_2$  consists of five jobs having length 3. Fig. 1 presents four profiles of this game. Unlike traditional load-balancing games, schedule (d), produced by LPT rule, is not a NE since the 3-job on  $M_1$  can reduce its rank from 5 to 4 by migration to either  $M_2$  or  $M_3$ . Note also that a cost-reducing deviation may hurt the rank of a player. For example, a deviation of a player from  $S_1$  from  $M_1$  in profile (a), will result in profile (b). The deviating player reduces its cost from 12 to 11 but increases its rank from 3 to 3.5.

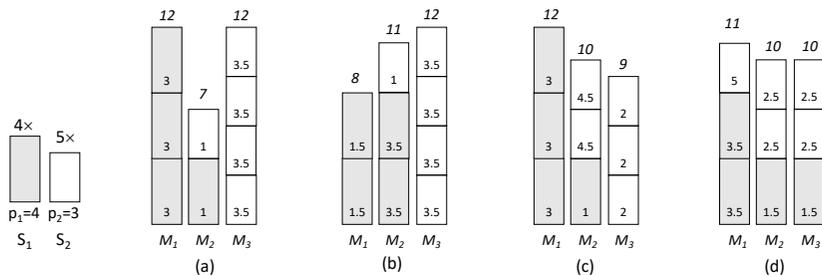


Figure 1: Jobs are labeled by their ranks. (a) A NE profile. (b) and (c) Deviations from the NE are harmful. (d) An LPT schedule.

## 2 Our Results

We show that competition dramatically impacts job-scheduling environments that are controlled by selfish users. SRBGs are significantly different from classical games; their analysis is misleading, and known tools and techniques fail even on simple instances.

Consider an instance with two machines and two competing jobs of different lengths. The long job will benefit from joining the short one, and the short one will benefit from escaping to an empty machine. Thus, even a very simple SRBG need not have a NE. For identical machines, we show that deciding whether a game instance has a NE is an NP-complete problem, even for an instance with homogeneous competition sets. Moreover, even in cases where a NE exists, BRD may not converge (see Fig. 2). On the other hand, we identify several non-trivial classes of instances for which a NE exists and can be calculated efficiently. Each of these positive results is tight in a sense that a slight relaxation of the class characterization yields a game that may not have a NE. Specifically, we present an algorithm for calculating a NE for games with unit-length jobs, for games in  $\mathcal{G}_h$  with a limited number of competition sets and machines, or with limited competition-set size, and games in  $\mathcal{G}_h$  in which the job lengths form a divisible sequence (e.g., powers of 2).

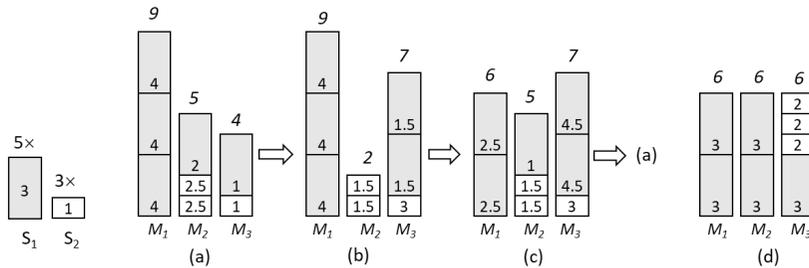


Figure 2: An example of a SRBG with  $c = 2$  homogeneous competition sets.  $S_1$  consists of 5 jobs of length 3 and  $S_2$  consists of 3 jobs of length 1. BRD may loop (profiles (a)-(b)-(c)-(a)), while a NE exists (profile (d)). Jobs are labeled by their ranks.

We provide tight bounds on the equilibrium inefficiency with respect to the minimum makespan objective. The inefficiency is analyzed using the common measures of Price of Anarchy (PoA) and Price of Stability (PoS). For classical job scheduling games on identical machines, it is known that  $PoS = 1$  and  $PoA = 2 - \frac{2}{m+1}$  [2]. We show that for SRBGs,  $PoS = PoA = 3 - \frac{6}{m+2}$ . This result and in particular the fact that  $PoS > 1$ , demonstrates the ‘price of competition’.

For related machines, we show that even the seemingly trivial case of unit-length jobs is tricky, and a NE may not exist, even if all jobs are in a single competition set. For this class of games, however, it is possible to decide whether a game has a NE, and to calculate one if it exists. Without competition, for

unit-jobs and related machines, a simple greedy algorithm produces an optimal schedule. Moreover,  $PoA = PoS = 1$ . We show that for SRBGs with unit jobs and related machines,  $PoS = PoA = 2$ . For games in  $\mathcal{G}_h$ , arbitrary-length jobs, and two related machines, we present an algorithm for calculating a NE, and prove that any application of BRD converges to a NE. We show that  $PoS = PoA = 2$ . The  $PoA$  upper bound is valid also for games with non-homogeneous competition sets.

Finally, we showed that the analysis of rank-based utilities is interesting and challenging in additional congestion games; in particular, non-symmetric scheduling games, network-formation games, and singleton cost-sharing games.

## References

- [1] I. Ashlagi, P. Krysta, and M. Tennenholtz. Social context games. In *International Workshop on Internet and Network Economics*, pages 675–683. Springer, 2008.
- [2] G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT Numerical Mathematics*, 19(3):312–320, 1979.
- [3] N. Immorlica, R. Kranton, and G. Stoddard. Striving for social status. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 672–672, 2012.

# Scheduling Games with Machine-Dependent Priority Lists

Vipin Ravindran Vijayalakshmi\*    Marc Schröder (Speaker) †

Tami Tamir‡

---

## 1 The Model

An instance of a *scheduling game with machine-dependent priority lists* is given by a tuple  $G = (J, M, (p_j)_{j \in J}, (s_i)_{i \in M}, (\pi_i)_{i \in M})$  where  $J$  is a finite set of  $n \geq 1$  jobs,  $M$  is a finite set of  $m \geq 1$  machines,  $p_j \in \mathbb{R}_+$  is the processing time of job  $j \in J$ ,  $s_i \in \mathbb{R}_+$  is the speed of machine  $i \in M$ , and  $\pi_i : J \rightarrow \{1, \dots, n\}$  is the priority list of machine  $i \in M$ .

A strategy profile  $\sigma = (\sigma_j)_{j \in J} \in M^J$  assigns a machine  $\sigma_j \in M$  to every job  $j \in J$ . Given a strategy profile  $\sigma$ , the jobs are processed according to their order in the machines' priority lists. The set of jobs that delay job  $j \in J$  in  $\sigma$  is  $B_j(\sigma) = \{j' \in J \mid \sigma_{j'} = \sigma_j \wedge \pi_{\sigma_j}(j') \leq \pi_{\sigma_j}(j)\}$ . Note that job  $j$  itself also belongs to  $B_j(\sigma)$ . Let  $p_j(\sigma) = \sum_{j' \in B_j(\sigma)} p_{j'}$ . The cost of job  $j \in J$  is equal to its completion time in  $\sigma$ , given by  $C_j(\sigma) = p_j(\sigma)/s_{\sigma_j}$ .

Each job chooses a strategy so as to minimize its costs. A strategy profile  $\sigma \in M^J$  is a *pure Nash equilibrium (NE)* if for all  $j \in J$  and all  $i \in M$ , we have  $C_j(\sigma) \leq C_j(i, \sigma_{-j})$ , where  $(i, \sigma_{-j})$  is the strategy profile in which job  $j$  chooses machine  $i$  and all other jobs choose according to  $\sigma$ . Let  $\mathcal{E}(G)$  denote the set of Nash equilibria for a given instance  $G$ . We would like to remark that  $\mathcal{E}(G)$  may be empty.

For a strategy profile  $\sigma$ , let  $C(\sigma)$  denote the cost of  $\sigma$ . The cost is defined with respect to some objective, e.g., the makespan, i.e.,  $C_{max} = \max_{j \in J} C_j(\sigma)$ , or the sum of completion times, i.e.,  $\sum_{j \in J} C_j(\sigma)$ . It is well known that decentralized decision-making may lead to sub-optimal solutions from the point of view of the society as a whole. For a game  $G$ , let  $P(G)$  be the set of feasible profiles of  $G$ . We denote by  $OPT(G)$  the cost of a social optimal (SO) solution; i.e.,  $OPT(G) = \min_{\sigma \in P(G)} C(\sigma)$ . We quantify the inefficiency incurred due to

---

\*vipin.rv@oms.rwth-aachen.de. Chair of Management Science, RWTH Aachen, Germany.

†m.schroder@maastrichtuniversity.nl. School of Business and Economics, Maastricht University, Netherlands.

‡tami@idc.ac.il. School of Computer Science, Reichman University, Israel.

self-interested behavior according to the *price of anarchy* (PoA) [4] and *price of stability* (PoS) [1]. The PoA is the worst-case inefficiency of a pure Nash equilibrium, while the PoS measures the best-case inefficiency of a pure Nash equilibrium.

Congestion games with priorities were studied by Farzad et al. [3], who considered priority based selfish routing for non-atomic and atomic users and analyzed the inefficiency of equilibria. More recently, Biló and Vinci [2] studied a congestion game with global priority classes that can contain multiple jobs and characterize the price of anarchy as a function of the number of classes.

## 2 Our Contribution

We first show that a pure Nash equilibrium does not always exist, and use this example to show that it is NP-complete to decide whether a particular game has a pure Nash equilibrium. We then provide a characterization of instances in which a pure Nash equilibrium is guaranteed to exist. Specifically, existence is guaranteed if the game belongs to at least one of the following four classes:  $\mathcal{G}_1$  : all jobs have unit processing time,  $\mathcal{G}_2$  : there are two machines,  $\mathcal{G}_3$  : all machines have the same speed and  $\mathcal{G}_4$  : all machines have the same priority list. For all four of these classes, there is a polynomial time algorithm that computes a Nash equilibrium. This characterization is tight in a sense that our inexistence example disobeys it in a minimal way: it describes a game on three machines, two of them having the same speed and the same priority list. Another characterization we consider is the number of jobs in the instance. We present a game of 4 jobs that has no pure NE, and show that every game of 3 jobs admits an NE.

We analyze the efficiency of Nash equilibria by means of two different measures of efficiency: the makespan and the sum of completion times. For all four classes of games with a guaranteed pure Nash equilibrium, we provide tight bounds for the price of anarchy and the price of stability with respect to both measures. Our results are summarized in Table 1.

Instance class \ Objective	Makespan PoA/PoS	Sum of Comp. Times PoA/PoS
$\mathcal{G}_1$ : Unit jobs	1	1
$\mathcal{G}_2$ : Two machines	$(\sqrt{5} + 1)/2$	$\Theta(n)$
$\mathcal{G}_3$ : Identical machines	$2 - 1/m$	$\Theta(n/m)$
$\mathcal{G}_4$ : Global priority list	$\Theta(m)$	$\Theta(n)$

Table 1: Our results for the equilibrium inefficiency.

(i) If jobs have unit processing times, we show that the price of anarchy is equal to 1, which means that selfish behavior is optimal. (ii) For two machines with speeds 1 and  $s \leq 1$  respectively, we prove that the PoA and the PoS are at most  $s + 1$  if  $s \leq \frac{\sqrt{5}-1}{2}$ , and  $\frac{s+2}{s+1}$  if  $s \geq \frac{\sqrt{5}-1}{2}$ . Moreover, our analysis is tight for

all  $s \leq 1$ . The maximum inefficiency, listed in Table 1, is achieved for  $s = \frac{\sqrt{5}-1}{2}$ . In case the sum of completion times is considered as an objective, the price of anarchy can grow linearly in the number of jobs. (iii) If machines have identical speeds, but potentially different priority lists, the price of anarchy with respect to the makespan is equal to  $2 - 1/m$ . The upper bound follows because every Nash equilibrium can be seen as an outcome of Graham’s List-Scheduling algorithm. The lower bound example shows the bound is tight, even with respect to the price of stability. For the sum of completion times objective, we show that the price of anarchy is at most  $O(n/m)$ , and provide a lower bound example for which the price of stability grows in the order of  $O(n/m)$ . (iv) If there is a global priority list, and machines have arbitrary speeds, we show that the  $\Theta(m)$ -approximation of List-Scheduling carries over for the makespan inefficiency, and the results for two machines carry over for the sum of completion times.

We conclude with results regarding the complexity of calculating a good NE. While a simple greedy algorithm can be used to compute an NE for an instance with identical machines (the class  $\mathcal{G}_3$ ), we show that it is NP-hard to compute an NE schedule that approximates the best NE of a game in this class. Specifically, it is NP-hard to approximate the best NE with respect to the minimum makespan within a factor of  $2 - 1/m - \epsilon$  for all  $\epsilon > 0$ , and it is NP-hard to approximate the best NE with respect to the sum of completion times within a factor of  $r$  for any constant  $r > 1$ .

For the full version and formal proofs, we refer to [5].

## References

- [1] E. ANSHELEVICH, A. DASGUPTA, J. KLEINBERG, E. TARDOS, T. WEXLER, T. ROUGHGARDEN (2008). *The Price of stability for network design with fair cost allocation*. SIAM Journal on Computing.
- [2] V. BILÒ AND C. VINCI (2020). *Congestion games with priority-based scheduling*. International Symposium on Algorithmic Game Theory. pp. 67–82. Springer.
- [3] B. FARZAD, N. OLVER, A. VETTA (2008) *A priority-based model of routing*. Chicago Journal of Theoretical Computer Science 1.
- [4] E. KOUTSOPIAS AND C. PAPADIMITRIOU (1999). *Worst-case equilibria*. Annual Symposium on Theoretical Aspects of Computer Science,
- [5] V. RAVINDRAN VIJAYALAKSHMI, M. SCHRÖDER, T. TAMIR (2020) *Scheduling games with machine-dependent priority lists*. Theoretical Computer Science, Volume 855, pp. 90-103.

# Scheduling Parallel Jobs in Two-processor Systems with Energy Constraint

Yulia Zakharova (Speaker) \*      Alexander Kononov †

---

## 1 Introduction

We consider the problem of scheduling a set of jobs  $\mathcal{J} = \{1, \dots, n\}$  on two speed scalable processors. Each job  $j \in \mathcal{J}$  is characterized by release time  $r_j$ , processing volume (work)  $V_j$  and the number  $size_j$  or the set  $fix_j$  of required processors. Note that parameter  $size_j$  for job  $j \in \mathcal{J}$  indicates that the job can be processed on any subset of parallel processors of the given size. Such jobs are called rigid jobs [5]. Parameter  $fix_j$  states that the job uses the prespecified subset of dedicated processors. Such jobs are called single mode multiprocessor jobs [5]. Job preemption might or might not be allowed in the exploring of scheduling in this work.

The standard homogeneous model in speed-scaling is considered. When a processor runs at a speed  $s$ , then the rate with which the energy is consumed (the *power*) is  $s^\alpha$ , where  $\alpha > 1$  is a constant. Each processor may operate at variable speed. We assume that the total work  $V_j$  of a rigid or single mode job  $j \in \mathcal{J}$  should be uniformly divided between the utilized processors. If job  $j$  uses two processors, then both processors run simultaneously at the same speed.

The aim is to find a feasible schedule with the minimum sum of completion times  $\sum C_j$  or the minimum makespan  $C_{\max}$  so that the energy consumption is not greater than a given energy budget  $E$ . This is a natural assumption in the case when the energy of a battery is fixed, i.e., the problem finds applications in computer devices whose lifetime depends on a limited battery efficiency (for example, multi-core laptops).

For the makespan problem on a single processor, there is an optimal non-preemptive schedule where all jobs are executed with the same speed. This schedule may be computed in linear time [12]. Bunde [3] developed an exact constructive algorithm for the uniprocessor setting with arbitrary release times of jobs. Pruhs [11] et al. investigated the single-processor problem of minimizing the average flow time of jobs, given a fixed amount of energy and release times of jobs. For unit-work jobs, they proposed a polynomial-time algorithm that simultaneously computes, for each possible energy level, the schedule with smallest average

---

\*kovaenko@ofim.oscsbras.ru. Sobolev Institute of Mathematics, Omsk, Russia

†alvenko@math.nsc.ru. Sobolev Institute of Mathematics, Novosibirsk, Russia

flow time. Bunde [3] adopted the approach to multiple processors and arbitrary work jobs. Moreover, online and offline algorithms have been developed for single-processor jobs and arbitrary number of processors (see, e.g., [1, 12, 13]).

Now we briefly present the known results for the classic problem of scheduling parallel jobs with given durations and without energy constraint. The non-preemptive problem with the total completion time criterion for rigid jobs is strongly NP-hard in the case of two processors [9]. Constant factor approximation algorithms were proposed for the non-preemptive independent jobs. These algorithms use list-type scheduling [15] and scheduling to minimize average response time (SMART) [14]. The non-preemptive single-mode problem is NP-hard in the two-processors case [7]. The strategy from preemptive schedule to the non-preemptive one gives a 2-approximation algorithm [4].

The preemptive two-processor problem with makespan for rigid jobs is polynomially solvable even in the case of arbitrary release dates [5]. The non-preemptive two-processor problem is ordinary NP-hard, but it becomes strongly NP-hard when jobs are related by precedence constraints [2, 6]. Approximation algorithms and non-approximability bounds were proposed for particular cases [5]. These algorithms use list-type and level-type strategies.

In this work we provide algorithms for speed scaling problems with rigid jobs and single mode jobs on two processors.

## 2 Makespan

We propose two-stage approximation and exact algorithms. At the first stage, we find a lower bound on the objective and calculate processing times of jobs using auxiliary convex programs, KKT conditions [8], and the Ellipsoid method [10]. Then, at the second stage, we transform our problem to the classic scheduling problem without speed scaling and use list-type scheduling via greedy rule to obtain feasible solutions.

**Theorem 1** *1) Problem  $P2|size_j, energy|C_{\max}$  is NP-hard. A  $\frac{3}{2}$ -approximate schedule can be found in polynomial time. 2) A  $\frac{5}{2}$ -approximate schedule can be found in polynomial time for  $P2|r_j, size_j, energy|C_{\max}$ .*

**Theorem 2** *Problems  $P2|r_j, size_j, pmtn, energy|C_{\max}$ ,  $P2|fix_j, energy|C_{\max}$  and  $P2|r_j, fix_j, pmtn, energy|C_{\max}$  are polynomially solvable.*

## 3 Total Completion Time

Here we also use two-stage method, but at the first stage we solve an auxiliary single-processor problem, and then, at the second stage, we transform the obtained single-processor schedule to a feasible two-processor one. We note that the sequence of jobs is important in the problems with  $\sum C_j$  criterion, so, at the first stage we determine not only durations but also a permutation of jobs.

**Theorem 3** *A 2-approximate schedule can be found in polynomial time for  $P2|size_j, energy| \sum C_j$ .*

**Theorem 4** *1) A  $2^{2\alpha-1/\alpha-1}$ -approximate schedule can be found in polynomial time for problem  $P2|fix_j, energy| \sum C_j$ . 2) A  $2^{\alpha/\alpha-1}$ -approximate schedule can be found in polynomial time for problem  $P2|fix_j, pmtn, energy| \sum C_j$ .*

The complexity status of the considered problems with  $\sum C_j$  criterion is open.

This research is supported by the Russian Science Foundation grant 21-41-09017.

## References

- [1] E. BAMPIS, D. LETSIOS, AND G. LUCARELLI (2014) *A note on multiprocessor speed scaling with precedence constraints* 26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2014. P. 138–142. ACM
- [2] P. BRUCKER, S. KNUST, D. ROPER, Y. ZINDER (2000) *Scheduling UET task systems with concurrency on two parallel identical processors*. Math. Methods Oper. Res. **52**(3). P. 369–387
- [3] D.P. BUNDE (2009) *Power-aware scheduling for makespan and flow*. J. Sched. **12**. P. 489–500
- [4] X. CAI, C.-Y. LEE, AND C.-L. LI (1998) *Minimizing total completion time in two-processor task systems with prespecified processor allocations*. Naval Research Logistics (NRL). **45**(2). P. 231–242
- [5] M. DROZDOWSKI (2009) *Scheduling for Parallel Processing*. London: Springer
- [6] J. DU, J.T. LEUNG (1989) *Complexity of scheduling parallel task systems*. SIAM J. Discrete Math. **2**(4). P. 472–478
- [7] J.A. HOOGEVEEN, S.L. VAN DE VELDE, B. VELTMAN (1994) *Complexity of scheduling multiprocessor tasks with prespecified processor allocations*. Discrete Applied Mathematics. **55**(3). P. 259–272
- [8] H.W. KUHN, A.W. TUCKER (1951) *Nonlinear programming*. Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability. P. 481–492
- [9] C.-Y. LEE AND X. CAI (1999) *Scheduling one and two-processor tasks on two parallel processors*. IEE Transactions **31**(5). P. 445–455
- [10] Y. NESTEROV (2018) *Lectures on Convex Optimization*. London: Springer

- [11] K. PRUHS, P. UTHAISOMBUT, AND G. WOEGINGER (2008) *Getting the best response for your erg.* ACM Trans. Algorithms. **4**(3). 17 p.
- [12] K. PRUHS AND R. VAN STEE (2007) *Speed scaling of tasks with precedence constraints.* Theory Comput. Syst. **43**. P. 67–80
- [13] D. SHABTAY AND M. KASPI (2006) *Parallel machine scheduling with a convex resource consumption function* Eur. J. Oper. Res. **173**. P. 92–107
- [14] U. SCHWIEGELSHOHN, W. LUDWIG, J.L. WOLF, J. TUREK, AND P.S. YU (1998) *Smart SMART bounds for weighted response time scheduling.* SIAM Journal on Computing. **28**. P. 237–253
- [15] J. TUREK, W. LUDWIG, J. WOLF, L. FLEISCHER, P. TIWARI, J. GLASGOW, U. SCHWIEGELSHOHN, AND P. YU (1994) *Scheduling parallelizable tasks to minimize average response time.* Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures. P. 200–209

# The parallel AGV Scheduling Problem with battery constraints: exact and heuristic approaches

Adriano Masone (Speaker) \*    Maurizio Boccia    Andrea Mancuso  
   Claudio Sterle

---

## 1 Automated guided vehicle systems

Automated guided vehicles (AGVs), firstly introduced in 1955, are driverless transportation systems used for horizontal movement of materials. Since then, the use of AGVs has grown enormously due to the wide range of benefits they produce. In particular, the increased productivity, the enhanced safety, the reduced labor cost, energy consumption, and emissions, lead the AGVs on the rise in several application fields, such as manufacturing, logistics, and healthcare, to cite only a few [4].

In these fields, AGVs might be part of a larger system used for the transportation of goods and materials from one location to another. This system is known as AGV-based transportation system (AGV system). It consists of a transportation network, a physical interface between the production/storage system, an AGV fleet, and a control system. The transportation network connects all the pick-up and delivery points. These points operate as interfaces between the production/storage system and the transportation system. Finally, AGVs move from one pick-up/delivery point to another on fixed or free paths. The main aim of an AGV system is to transfer the right amount of the right material to the right place at the right time [7]. To this end, many tactical (e.g., fleet sizing and flowpath design) and operational (e.g., dispatching, routing and scheduling) issues have to be addressed when designing an AGV system [9].

In this work, we focus on the AGV scheduling issues whose effective and efficient management allows to improve the overall productivity of transfer operations and to avoid delays in production and material handling processes [6]. A significant research activity on AGV systems and on related scheduling problems has been conducted in the last twenty years, as witnessed by the survey works reported in [5, 10]. In particular, different scheduling problems were proposed on the basis of the considered AGV requirements and capabilities (e.g., capacity, single/multi-

---

\*{adriano.masone, maurizio.boccia, andrea.mancuso, claudio.sterle}@unina.it. Department of Electrical Engineering and Information Technology, University of Naples "Federico II", via Claudio 21, 80125 Naples, Italy.

load, and schedules of additional equipment) [3]. However, most of the contributions on the topic frequently neglected the issues related to the battery depletion and recharge, as noticed in [2]. In this context, we study the Automated Guided Vehicle Scheduling Problem with battery constraints (*ASP-BC*). To the best of our knowledge, the only other work addressing this kind of problem is reported in [7] where the *ASP-BC* was defined to address a real problem of a manufacturing company implementing the Industry 4.0 paradigm for internal logistics [8]. Moreover, in [7], the authors proposed a Mixed Integer Linear Programming (*MILP*) formulation for the *ASP-BC* and demonstrated the impact of the AGV battery recharge times on the completion time of the overall material handling process.

## 2 The AGV Scheduling Problem with battery constraints

The *ASP-BC* can be schematized, on the basis of the following assumptions:

- 1) a set of AGVs must move a set of packages from a central warehouse to different workstations;
- 2) all the AGVs are initially located at the central warehouse with fully charged batteries;
- 3) an AGV can be loaded with a single package on each trip. Being the number of available AGVs much lower than the number of packages, an AGV could perform more than one trip;
- 4) the AGV battery consumption depends on the travel time and the weight carried;
- 5) an AGV battery has to be fully recharged at the central warehouse before it is completely depleted. The charging time is fixed and does not depend on the residual energy;
- 6) the *ASP-BC* objective is to minimize the makespan of the handling process.

On this basis, it is evident that *ASP-BC* differs from other AGV scheduling problems due to its peculiar operational characteristics. Therefore, its solution requires the development of exact and ad-hoc heuristic methods that exploit these characteristics.

The main contributions of this work can be summarized as follows:

- 1) We propose an original mixed-integer linear programming (*MILP*) model for the *ASP-BC*. In particular, we formulate the problem as a variant of the Bottleneck Generalized Assignment Problem (*BGAP*). In our model, unlike other *BGAPs* in literature, there are two levels of assignment: transfer jobs to charging operations and charging operations to AGVs.
- 2) We develop a three-step matheuristic to solve large size instances of the *ASP-BC*. A bin-packing problem (*BPP*) with the aim of determining the minimum number of required battery charges is solved through an IP solver at the first step. The *BPP* solution determines a lower bound for the *ASP-BC* and an

assignment of transfer jobs to charging operations. Then, a feasible solution is obtained solving a bottleneck generalized assignment problem in the second step. Finally, the solution is improved through a local search performing reassignment and swap moves of transfer jobs among available AGVs.

3) We solve instances built from real data provided by a manufacturing company. The results on different kinds of instances, in terms of AGV fleet size, number of jobs, job duration, and related energy consumption, show that the complexity of the ASP-BC is strongly related to the AGV energy consumption. Moreover, the performed experimentation confirms the applicability and the effectiveness of the proposed approaches. The new formulation allows us to solve instances with up to 150 jobs and 10 AGVs within 1800 seconds, outperforming the results of the formulation proposed in [7]. The matheuristic is able to solve instances with up to 200 jobs and 10 AGVs in few minutes and an average optimality gap lower than 1%.

Future works will be aimed at investigating the possibility of taking into account other operational aspects affecting the battery consumption (e.g., the AGV routing). Indeed, for the sake of completeness, we highlight that *ASP-BC* can be considered as a sub-problem of the Electric Vehicle Routing Problem, which is a special case of the AGV routing problems with battery constraints [1], where scheduling decisions are integrated with routing decisions related to the movement of each job from a pick-up to a delivery point. Therefore, it would be interesting to extend the proposed approach to problems involving both routing and scheduling decisions.

## References

- [1] C. BONGIOVANNI, M. KASPI AND N. GEROLIMINIS (2019). *The electric autonomous dial-a-ride problem*, Transportation Research Part B: Methodological, 122, 436-456.
- [2] M. DE RYCK, M. VERSTHEYHE AND K. SHARIATMADAR (2020). *Resource management in decentralized industrial Automated Guided Vehicle systems*, Journal of Manufacturing Systems, 54, 204-214.
- [3] H. FAZLOLLAHTABAR, M. SAIDI-MEHRABAD (2015). *Methodologies to Optimize Automated Guided Vehicle Scheduling and Routing Problems: A Review Study*, J Intell Robot Syst, 77, 525-545.
- [4] G. FRAGAPANE, R. DE KOSTER, F. SGARBOSSA AND J. O. STRANDHAGEN (2021). *Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda*, European Journal of Operational Research, 294, 2, 405-426.
- [5] E. KAoud, M. A. EL-SHARIEF AND M.G. EL-SEBAIE (2017) *Scheduling problems of automated guided vehicles in job shop, flow shop, and container*

*terminals*, 2017 4th International Conference on Industrial Engineering and Applications, 60-65.

- [6] J. LUO, Y. WU AND A. B. MENDES (2016). *Modelling of integrated vehicle scheduling and container storage problems in unloading process at an automated container terminal*, Computers & Industrial Engineering, 94, 32-44.
- [7] A. MASONE, T. MURINO, C. STERLE AND M. STRAZZULLO (2021). *A MILP Formulation for an Automated Guided Vehicle Scheduling Problem with Battery Constraints*, APMS. Intelligence for Sustainable and Resilient Production Systems, Springer International Publishing, 15-23.
- [8] J. MEHAMI, M. NAWI AND R. Y. ZHONG (2018). *Smart automated guided vehicles for manufacturing in the context of Industry 4.0*, Procedia Manufacturing, 26, 1077-1086.
- [9] I. F. A. VIS (2006). *Survey of research in the design and control of automated guided vehicle systems*, European Journal of Operational Research, 170, 3, 677-709.
- [10] C. XIE AND T. T. ALLEN (2015). *Simulation and experimental design methods for job shop scheduling with material handling: a survey*, Int J Adv Manuf Technol, 80, 233-243.

# A hybrid local search algorithm for the Continuous Energy-Constrained Scheduling Problem

Roel Brouwer (Speaker) \*      Marjan van den Akker †  
Han Hoogeveen ‡

---

## 1 Introduction

We consider the Continuous Energy-Constrained Scheduling Problem (CECSP), introduced by Nattaf et al. [4]. A set  $J_1, \dots, J_n$  of jobs has to be processed on a continuous resource  $P$ . Each job  $j$  requires an amount of resource equal to  $E_j$ . We want to find a schedule such that: a job does not start before its release time  $r_j$ , is completed before its deadline  $\bar{d}_j$ , and respects its lower and upper bounds  $(P_j^-, P_j^+)$  on resource consumption during processing. Our objective is to minimize the total weighted completion time ( $\min \sum_j w_j C_j$ ). We look at the case where both the resource and time are continuous. We assume that there is no efficiency function influencing resource consumption.

The CECSP can be seen as a variant of the Resource Constrained Project Scheduling Problem (RCPSP). A survey by Hartmann & Briskorn [2, 3] provides a good overview. Baptiste et al. [1] defined the Cumulative Scheduling Problem (CuSP) as a subproblem of the RCPSP. More recent work by Nattaf et al. [4, 5, 6, 7], further defined the CECSP as a generalization of CuSP and provided a hybrid branch-and-bound algorithm to find exact solutions for small instances. Through its relation to CuSP, Nattaf et al. proved that the CECSP is NP-complete [6].

The CECSP is also closely related to the scheduling of malleable jobs (as introduced by Turek et al. [8]) on parallel machines, which involves the scheduling of jobs on  $P$  machines, while the number of machines assigned to a job can change during its execution.

In this work, we present a hybrid local search algorithm that exploits a decomposition of the problem, where start and completion times as well as resource consumption are determined by an event-based LP formulation. We perform computational experiments to compare the performance of our algorithm with exact

---

\* [r.j.j.brouwer@uu.nl](mailto:r.j.j.brouwer@uu.nl). Department of Information and Computing Sciences, Utrecht University

† [j.m.vandenakker@uu.nl](mailto:j.m.vandenakker@uu.nl). Department of Information and Computing Sciences, Utrecht University

‡ [j.a.hoogeveen@uu.nl](mailto:j.a.hoogeveen@uu.nl). Department of Information and Computing Sciences, Utrecht University

approaches and to show its ability to deal with larger problem sizes. Our approach can be extended to deal with piece-wise linear resource availability functions  $P(t)$ , explicit precedence relations and linear efficiency functions.

## 2 Hybrid local search algorithm

For each job  $J_j$ , we need to find a start time, a completion time, and a resource consumption profile within its execution window, such that all constraints are met.

We call the start and completion of a job *events*. If we know the order of events, we can split the total time window into  $2n - 1$  intervals, where each interval is bounded by two consecutive events. During these intervals, the set of jobs that are being processed does not change. Now let  $p_j(t)$  denote the resource consumption profile of job  $J_j$ . We can show that there always exist an optimal solution in which  $p_j(t)$  remains constant during each interval. We can therefore limit ourselves to such solutions.

With this, we can formulate an LP that, for a given order of events, determines the times at which these events take place ( $t_i$ ), and the amount of resource ( $p_{j,i}$ ) that each job  $j$  consumes in interval  $i$ , which corresponds to the time window  $[t_i, t_{i+1}]$ . Since we know the indices  $i_1$  and  $i_2$  corresponding to the start time  $S_j$  and completion time  $C_j$  of each job  $j$ , we put  $p_{j,i} = 0$  for  $i = 1, \dots, i_1 - 1$  and for  $i = i_2, \dots, 2n - 1$ . For the remaining intervals we require  $p_{j,i}$  to be between its lower and upper bound times the length of the interval, which is  $(t_{i+1} - t_i)$ . We further request that the total amount of resource assigned to job  $j$  equals  $E_j$ , that the total amount of resource used in interval  $i$  by all available jobs is no more than  $P(t_{i+1} - t_i)$ , that  $t_{i+1} \geq t_i$ , and finally that we obey the release dates and deadlines. The objective of minimizing the total weighted completion time can easily be formulated.

This LP is then used to evaluate the quality of (candidate) solutions considered during local search. *We use a simulated annealing algorithm to find a good order of events, and use the LP to find the corresponding optimal schedule.*

To evaluate the relative quality of infeasible event orders, we add slack variables that allow for the violation of lower bounds, upper bounds, and/or resource availability constraints, with appropriate cost. For example, the constraint for the energy consumption of job  $j$  in interval  $i$  then becomes

$$P_j^-(t_{i+1} - t_i) - s_{j,i}^- \leq p_{j,i} \leq P_j^+(t_{i+1} - t_i) + s_{j,i}^+$$

To determine an initial solution for our simulated annealing algorithm we use a greedy algorithm that may violate resource availability constraints. It defines intervals based on events (in this case, release dates and deadlines) and assigns resources during an interval to (available) jobs in order of deadline, while at least providing each job the minimum amount of resources they need to still be able to complete in time and respect their bounds. From the constructed schedule, an event order is derived.

The neighborhood is defined by a combination of operators: (1) swapping two events that are adjacent to each other in the order; (2) moving an event a random number of places up or down the order; (3) moving both events associated with a single job simultaneously.

### 3 Computational experiments

We generated a set of instances of different sizes ( $n = 5, 10, 15, 20, 50, 100, 200$ ). These are checked for feasibility by an LP, based on a max-flow on a graph representation of the problem that ignores lower bounds ( $P_j^-$ ).

The smaller instances are solved to optimality by an event-based MILP formulation. The local search algorithm is run on all instances.

Our computational experiments are ongoing and our current results suggest that the local search is able to find good solutions for larger instances.

### References

- [1] P. Baptiste, C. L. Pape, and W. Nuijten. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92:305–333, 1999.
- [2] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207:1–14, 2010.
- [3] S. Hartmann and D. Briskorn. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 297:1–14, 2022.
- [4] M. Nattaf, C. Artigues, and P. Lopez. A polynomial satisfiability test using energetic reasoning for energy-constraint scheduling. In *International Conference on Project Management and Scheduling (PMS 2014)*, pages 169–172, 2014.
- [5] M. Nattaf, C. Artigues, and P. Lopez. Cumulative scheduling with variable task profiles and concave piecewise linear processing rate functions. *Constraints*, 22:530–547, 10 2017.
- [6] 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*, 38:459–492, 3 2016.
- [7] M. Nattaf, M. Horváth, T. Kis, C. Artigues, and P. Lopez. Polyhedral results and valid inequalities for the continuous energy-constrained scheduling problem. *Discrete Applied Mathematics*, 258:188–203, 2019.

- [8] J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms for scheduling parallelizable tasks. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 323–332, 1992.

# Graph burning and non-uniform $k$ -centers for small treewidth

Matej Lieskovský \*

Jiří Sgall (Speaker) \*

---

## 1 Introduction

We give an algorithm that proves the non-uniform  $k$ -center problem to be in XP when parametrized by the number of different radii and the treewidth of the graph. This extends the known exactly solvable cases of the non-uniform  $k$ -center problem; in particular this also solves the  $k$ -center with outliers on graphs of small treewidth exactly. We then use this algorithm to give polynomial-time approximation scheme for the graph-burning problem for arbitrary graphs of small treewidth.

In an instance of the (uniform)  $k$ -center problem we are given a set of  $n$  nodes, a metric  $d$  defining distances between the nodes, and parameters  $k$  and  $r$  that tell us how many centers we are allowed to use and the radius of every center's reach, respectively. We must then decide if a set of  $k$  nodes can be selected as centers so that every node will be within distance  $r$  of some center. The  $k$ -center decision problem, first formulated in 1985 by Hakimi [7], is known to be strongly NP-hard. Minimizing  $r$  for a given  $k$  is the focus of a paper by Hochbaum and Shmoys [8] who provided a 2-approximation algorithm and proved it to be optimal unless  $P=NP$ .

The  $k$ -center problem with radius  $r$  rarely fits reality perfectly. A commonly needed generalization is the introduction of centers of varying reach. This is modelled by the non-uniform  $k$ -center (NUkC) problem, first defined in [5] although a special case had already been analyzed in [6]. Current techniques work best only with a limited number of different radii, while algorithms for the general case are lacking. The currently best results are from [5], where algorithms for two different radii are given. The special case known as  $k$ -center with outliers where one of the radii is zero is solved with an approximation factor of 2, which is optimal unless  $P=NP$ . An approximation factor of  $(1 + \sqrt{5})$  is given otherwise. They also prove that no constant-factor approximation can exist for the general case of the non-uniform  $k$ -center problem.

An entry point diametrically opposite to the uniform  $k$ -center problem is the graph burning problem, where each center gets a unique radius. Given an unori-

---

\*ml@iuuk.mff.cuni.cz, sgall@iuuk.mff.cuni.cz, Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic. Partially supported by the project 19-27871X of GA CR.

ented graph with unit-length edges and a single parameter  $g$ , our task is to cover the entire graph with  $g$  centers, each having a unique integer radius from 0 to  $g-1$ . A 3-approximation algorithm for  $g$  has been given in [2], but no better result for general graphs is known.

Graph burning was originally introduced in [3] to model the spread of a contagion through a network. In the original formulation, all nodes start unburned and in every time step fire spreads to all nodes that neighbour a burning node and then a single additional node is set on fire. The burning number  $g$  of a graph is defined as the number of time steps needed for all nodes to be on fire.

We approach the graph burning problem by restricting the graphs to be burned. For graph burning of linear forests (unions of disjoint paths) [2, 4] gave both a proof of NP-hardness and a PTAS. We shall consider a more general case of graphs of small treewidth.

## 2 Our results and techniques

We show that the non-uniform  $k$ -center problem is polynomial when parametrized by the number of different radii and treewidth. We do this by giving a recursive—or, equivalently, dynamic programming—algorithm for the problem.

We then use this XP algorithm to design a PTAS for burning graphs of a constant treewidth, with a slight generalization that allows edge lengths. This result significantly improves previous results, as a PTAS was known only for linear forests, not even for trees.

### 2.1 The NUKC problem with $\ell$ different radii

In an instance of this problem we are given an arbitrarily undirected graph  $G$  on  $n$  nodes with edge lengths from  $\mathbb{R}_0^+$  and distance  $d(u, v)$  defined as the length of the shortest path from  $u$  to  $v$ . We are also given a list of  $\ell$  radii  $r_1, r_2, \dots, r_\ell \in \mathbb{R}_0^+$ , and a vector  $(x_1, x_2, \dots, x_\ell)$  where  $x_i \in \mathbb{N}$  specifies how many centers of radius  $r_i$  we are allowed to use.

**Definition 1** *We define  $K \subseteq V(G) \times \{1, \dots, \ell\}$  as the set of all possible centers and  $K_v = \{(w, i) \in K \mid d(v, w) \leq r_i\}$  as the set of all centers that will cover node  $v$ .*

*A solution to an instance of the NUKC problem is an assignment (of centers)  $A : V(G) \rightarrow K$  such that every node  $v \in V(G)$  is assigned a center  $A(v) \in K_v$  and in addition, for each  $i$ , the number of centers of radius  $r_i$  in the image of  $A$  is at most  $x_i$ .*

### 2.2 Tidy assignments

We design a concept of “tidy” assignments of centers to vertices. In tidy assignments all nodes on any shortest path from a node to the assigned center are

assigned the same center. This allows us to process the graph and check partial solutions in the ordering given by the tree decomposition.

For the following definition and lemma we abuse notation so that  $A(u)$  also denotes the vertex where the center is located, i.e., the first component of  $A(u)$ . Thus, if  $A(u) = (w, i)$ , then  $d(u, A(u))$  denotes  $d(u, w)$  and a path to  $A(u)$  means a path to  $w$ .

**Definition 2** *An assignment  $A$  is called tidy if for any node  $v$  on any shortest path from  $u$  to  $A(u)$  we have  $A(v) = A(u)$ . In particular, if the assignment uses some center  $(v, i)$ , then  $A(v) = (v, i)$ .*

The following lemma shows that we can test locally whether our assignment is tidy.

**Lemma 3** *An assignment  $A$  is tidy if and only if for every edge  $uv \in E(G)$  one of the following holds:*

- $A(u) = A(v)$
- $d(u, A(u)) < d(u, v) + d(v, A(u))$  and  $d(v, A(v)) < d(v, u) + d(u, A(v))$

Once we restrict feasible solutions to tidy assignments, the algorithm is designed using standard dynamic programming techniques for graphs of bounded treewidth. The key property of tidy assignments, namely that they can be checked edge by edge, allows us to remember for a given bag in the tree decomposition only the assignment of the vertices in the bag.

Alternatively, instead of an explicit dynamic program, it is possible to use metatheorems. The extension of Courcelle's theorem by Arnborg, Lagergren and Seese [1] shows that the problem lies in XP. This requires a formulation of the problem in extended monadic second order logic which is then used to create a tree automaton that behaves similarly to the dynamic program.

## 2.3 PTAS for graph burning

To design PTAS for graph burning based on the NUKC problem with  $\ell$  different radii we need to use only a constant number of radii instead of  $g$  as in the definition. To do this, for a given constant  $\varepsilon > 0$ , we use radii  $\varepsilon \cdot g, 2\varepsilon \cdot g, \dots$ , with appropriate rounding and multiplicities. For positive instances of graph burning, we obtain a solution with  $g' \leq (1 + \varepsilon)g$ , which gives a PTAS.

## References

- [1] Stefan Arnborg, Jens Lagergren and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.
- [2] Stéphane Bessy, Anthony Bonato, Jeannette Janssen, Dieter Rautenbach, and Elham Roshanbin. Burning a graph is hard. *Discrete Applied Mathematics*, 232:73 – 87, 2017.

- [3] Anthony Bonato, Jeannette Janssen, and Elham Roshanbin. How to burn a graph. *Internet Mathematics*, 12, 2015.
- [4] Anthony Bonato and Shahin Kamali. Approximation Algorithms for Graph Burning. In Proc. of the 15th Ann. Conf. on Theory and Applications of Models of Computation (TAMC). Lecture Notes in Comp. Sci. 11436. Springer, 2019.
- [5] Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The non-uniform k-center problem. *ACM Trans. Algorithms*, 16(4), June 2020.
- [6] Moses Charikar, Samir Khuller, David Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. pages 642–651, 01 2001.
- [7] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450–459, 1964.
- [8] Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

# Methods for Planning the Search of Infected Nodes in Uncertain Graphs

Benjamín Rubio Orellana (Speaker) \*      José Baboun †  
Isabelle Beaudry ‡      Mauricio Castro §      Alejandro Jara ¶  
José Verschae ‖

---

## 1 Introduction

The efficient detection and localization of disease outbreaks has become a very important problem in the COVID-19 pandemic. Taking PCR tests directly in the water of sewage networks constitutes a promising method for the quick detection of new outbreaks [1]. For a sample taken in a given manhole, the result of a PCR test yields an estimate of the number of people infected whose wastewater passes through said manhole. This information, coupled with a careful planning of the sampled manholes, has the potential of playing a crucial role in the timely detection of new outbreaks.

We can model the wastewater network using a directed rooted in-tree  $T = (V, E)$ , where each manhole can be associated with a node in  $V$  and each edge in  $E$  reflects a pipe between manholes. The root corresponds to the water treatment plant. In its most basic form, our problem corresponds to the search of one infected node within this tree. Here, we assume that we can test for infections of a given subtree by taking a sample in its root. This and other variants of searches in trees (and more general directed acyclic graphs) have been studied extensively; see e.g. Onak and Parys [2] and Dereniowski [3].

---

\*berubio@uc.cl. Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Santiago, Chile.

†jibaboun@uc.cl. Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Santiago, Chile.

‡isabelle.beaudry@mat.uc.cl. Fac. de Matemáticas, Pontificia Universidad Católica de Chile, Santiago, Chile.

§mcastro@mat.uc.cl. Fac. de Matemáticas, Pontificia Universidad Católica de Chile, Santiago, Chile.

¶ajara@mat.uc.cl. Fac. de Matemáticas, Pontificia Universidad Católica de Chile, Santiago, Chile.

‖jverschae@uc.cl. Instituto de Ingeniería Matemática y Computacional, Pontificia Universidad Católica de Chile, Santiago, Chile.

Although the network carrying water is indeed a tree, one often finds in real data that the network information stored by the waste water treatment companies does not correspond to a tree. This comes from the fact that network databases additionally store pipes which are blocked and do not carry flow. Moreover, the extra pipes are not necessarily marked as unused. To model this situation, we assume that we are given a directed acyclic graph (DAG)  $G = (V, F)$  with a known root  $r$ . The actual tree  $T = (V, E) \subseteq G$ , containing the edges  $E$  of  $F$  that actually carry flow, is unknown.

## 2 Our Model

We propose a model to study policies to find a single infected node, taking into consideration the uncertainty of the graph. We assume that viral load is detected in a node  $u$  if and only if there is a path from the infected node to  $u$  in the underlying tree. Assume that every day we choose  $k$  nodes to sample, and that we know the results of each sample immediately. Knowing that there is a single infected node  $v^*$  in the whole network, the objective of our policy is to detect the infected node minimizing the makespan, that is, minimizing the number of days to find  $v^*$  in the worst-case. In order to formalize the analysis we first define the notion of *ideal* of an ideal generated by  $u \in V$ , which reflects the set of nodes that potentially affect samples in  $u$ .

**Definition 1** For a given node  $u \in V$  and a DAG  $H$ , we denote by  $I_u(H)$  the ideal generated by  $u$  in  $H$ , that is,  $I_u(H)$  is the set of all nodes  $w$  for which there exists a  $w - u$  path in  $H$ .

Notice that a sample taken in a node  $u \in V$  detects viral load if and only if the infected node  $v^* \in V$  belongs to  $I_u(T)$ , which is an unknown set.

More generally, given a set  $S$  of nodes to sample, let us denote by  $S_1$  (respectively  $S_0$ ) the set of sampled nodes where viral load was detected (respectively, not detected). We can reduce the candidates for infected node after each sample using the results.

**Lemma 2** Without uncertainty, that is, if  $G = T$ , after testing the nodes in a set of sample nodes  $S$  we deduce that the infected node must belong to the set

$$N(V, S, v^*) = \left( V \cap \bigcap_{u \in S_1} I_u(T) \right) \setminus \left( \bigcup_{u \in S_0} I_u(T) \right).$$

However, in the uncertain case things are more complicated, as we cannot simply change  $I_u(T)$  by  $I_u(G)$  in the expression above: we can never be sure if all nodes in an ideal  $I_u(G)$  effectively have paths to  $u$  in  $T$ , as  $G$  contains more edges than  $T$ . Taking this into account, we define a *robust* version of an ideal.

**Definition 3** The robust ideal of  $u \in V$ , denoted as  $\bar{I}_u(G)$ , is the set of all nodes  $v$  in  $G$  such that all paths from  $v$  to the root pass through  $u$ .

We can observe that for all nodes  $u \in V$  it holds that  $\bar{I}_u(G) \subseteq I_u(T) \subseteq I_u(G)$ . This follows from the fact that all nodes in the robust ideal of  $u$  have all paths passing through  $u$ , then in particular the real path does too (the one in  $T$ ). For the other inclusion, adding extra edges can only increase the size of the ideal. Consequently,  $\bar{I}_u(G)$  is an underestimation of the real ideal, and  $I_u(G)$  an overestimation. Using this we can adjust the previous lemma for the uncertain case.

**Lemma 4** *In the general case, after testing the nodes in a set of sample nodes  $S$  we can deduce that the infected node belongs to the set*

$$N(V, S, v^*) = \left( V \cap \bigcap_{u \in S_1} I_u(T) \right) \setminus \left( \bigcup_{u \in S_0} \bar{I}_u(T) \right).$$

Heuristically, to adequately determine a set of nodes  $S$  to test in a given day, we would like that to balance the sizes of all sets  $\{N(V, S, v^*) : v^* \in V\}$ . In this case we could avoid cases that are too unfavorable. Therefore, we consider the problem

$$\min_{S \subseteq V : |S| \leq k} \left\{ \max_{v^* \in V} |N(V, S, v^*)| \right\}.$$

Solving the previous problem optimally proves to be difficult, especially in the uncertain case. We propose an heuristic approach for searching for the answer. Given an upper bound for the answer to the problem, we greedily add nodes to an initially empty set  $S$  of nodes to sample. Each time we add a node whose size of ideal in the graph induced by the remaining nodes is maximal between all nodes with this size less or equal to the upper bound, and then we remove the nodes in this ideal from the remaining nodes. In the case of a directed rooted tree, the upper bound reflects directly the answer to the problem except for the case in which all sampled do not detect viral load. In the uncertain case this upper bound still serves as an approximation to the possible solution and we can try to avoid using nodes with highly uncertain *ideals* in the sampling process.

To properly adjust the value of the upper bound we can perform a binary search on it looking for the minimal value at which the size of  $V \setminus \bigcup_{u \in S} I_u(G)$  (that reflects the case all nodes do not detect disease) is below the upper bound. The final complexity of the proposed algorithm is  $O(N^3 \log(N))$  where  $N = |V|$ .

We tested the algorithm iteratively to find infected nodes in simulations based on real and synthetic data. We investigated different sizes of the graph, levels of uncertainty and amount of samples per day. With a 4000 node graph, five percent of uncertainty and five samples each day the maximum always came to six iterations in the tree and between 6 and 8 in the uncertain case. On the other hand, with 20 percent of uncertainty the maximum iterations in the uncertain case can go up to ten days. In general, less samples and more uncertainty increase the number of iterations required by our simulation.

Our empirical results leave several interesting theoretical open questions. In the case without uncertainty, it is easy to see that in a tree of bounded degree a makespan of  $O(\log(N))$  is attainable. In real wastewater network, the typical maximum degrees are around 5 due to spacial constraints, and this even holds for the larger graph  $G$ . Hence, it is natural to ask whether it is possible to obtain the same result in the uncertain case. We stress that for this results to be useful the policy should work for any underlying tree  $T$ . Similarly, we can ask about the complexity of finding a policy that minimizes the makespan, either exactly or approximately.

## References

- [1] T. BALDOVIN, I. AMORUSO, M. FONZO, A. BUJA, V. BALDO, S. COCCHIO, C. BERTONCELLO (2021) *SARS-CoV-2 RNA detection and persistence in wastewater samples: An experimental network for COVID-19 environmental surveillance in Padua, Veneto Region (NE Italy)*. Science of The Total Environment. 760:143329.
- [2] K. ONAK AND P. PARYS (2006) *Generalization of Binary Search: Searching in Trees and Forest-Like Partial Orders*. Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006, 379–388.
- [3] D. DERENIOWSKI (2008) *Edge ranking and searching in partial orders*. Discrete Applied Mathematics. 156:2493–2500.

# Robustification of Online Graph Exploration Methods

Franziska Eberle<sup>\*</sup>    Alexander Lindermayr<sup>†</sup>    Nicole Megow<sup>†</sup>  
Lukas Nölke<sup>†</sup>    Jens Schlöter<sup>†</sup>(Speaker)

---

## 1 Introduction

In an online exploration problem, a searcher has to create a complete map of an unknown environment without any *a priori* knowledge of its topology. The searcher makes all decisions based on local information and must move through the environment to obtain new data, e.g., via optical sensors. Online exploration problems emerge in several scenarios, such as the navigation of mobile robots, be it an autonomous vacuum cleaner or a scientific exploration robot in the deep sea or on Mars, and security maintenance of large networks [1, 2]. We initiate the study of a learning-augmented variant of the classical, notoriously hard online graph exploration problem by adding access to machine-learned predictions. We propose an algorithm that naturally integrates predictions into the well-known Nearest Neighbor (NN) algorithm and significantly outperforms any known online algorithm if the prediction is of high accuracy while maintaining good guarantees when the prediction is of poor quality. We provide theoretical worst-case bounds that gracefully degrade with the prediction error. Further, we extend our concept to a general framework to *robustify* algorithms. A full version [3] of this abstract is to appear in the proceedings of AAAI 2022.

We consider the *online graph exploration problem* on an undirected connected graph  $G = (V, E)$  with  $n$  vertices and non-negative edge costs  $c(e)$ ,  $e \in E$ . We assume that each vertex  $v \in V$  has a unique label, i.e., vertices are distinguishable. Starting in a given vertex  $s \in V$ , the task of the searcher is to find a tour that visits all vertices of  $G$  and returns to  $s$ . Initially, the graph is unknown to the searcher and she obtains local information when traversing the graph: Whenever she visits (*explores*) a vertex for the first time, all incident edges, as well as their costs, and the labels of their end points are revealed. This exploration model is also known as *fixed graph scenario* [4]. The searcher can explore a vertex  $v$  by traversing a path of known edges from her current location to  $v$ . When the searcher traverses an

---

<sup>\*</sup>f.eberle@lse.ac.uk. Department of Mathematics, London School of Economics, United Kingdom.

<sup>†</sup>{linderal,nmegow,noelke,jschloet}@uni-bremen.de. Faculty of Mathematics and Computer Science, University of Bremen, Germany.

edge  $e$ , she pays its cost  $c(e)$ . The goal is to minimize the total cost for exploring all vertices (and returning to the start vertex).

To measure the quality of algorithms for online graph exploration, we resort to standard *competitive analysis*, which compares the cost of an algorithm with an *offline optimal solution*, i.e., the optimal tour that can be found if the graph is known in advance. If the ratio between the costs of these two tours is bounded by  $\rho \geq 1$  for every instance, then the online algorithm is  $\rho$ -*competitive*. The *competitive ratio* of an algorithm is the minimum  $\rho$  for which it is  $\rho$ -competitive. For online graph exploration, the offline problem is a variant of the well-known, NP-hard Traveling Salesperson Problem (TSP) [5], in which we are allowed to visit vertices multiple times.

The currently best known ratio of  $\mathcal{O}(\log n)$  is achieved by NN [6], which greedily explores the unknown vertex closest to the current position, and the hierarchical Depth First Search algorithm (HDFS) [7]. For arbitrary input graphs there remains a large gap between the best known upper bound and the best known lower bound of  $10/3$  [8].

For rather special graph classes, there are constant-competitive algorithms. Examples are planar graphs with a competitive ratio of 16 [4, 7], graphs of bounded genus  $g$  with a ratio of  $16(1 + 2g)$ , and graphs with  $k$  distinct weights with a ratio of  $2k$  [7]. One drawback of such algorithms is that the topology of the graph is initially unknown, so we might not know if a certain special case actually applies. While some algorithms, e.g., HDFS, achieve improved ratios for their special case and at the same time maintain a worst-case guarantee on arbitrary inputs, this does not hold in general. E.g., the competitive ratio of the algorithm for planar graphs on arbitrary inputs is unknown.

## 2 Our results

As a main result, we present a *robustification framework* that allows us to *robustify* any online graph exploration algorithm  $\mathcal{A}$ . Given an algorithm  $\mathcal{A}$ , the framework achieves a competitive ratio that, asymptotically, matches the minimum of the competitive ratios of the given algorithm and the best known algorithm for arbitrary graphs. This allows us to exploit special case algorithms for an improved competitive ratio if the special case applies, while at the same time matching the best known ratio for arbitrary inputs.

**Theorem 1** *For any  $\lambda > 0$ , there is a robustification framework  $\mathcal{R}$  for the online graph exploration problem that, given an online algorithm  $\mathcal{A}$  and an instance  $\mathcal{I} = (G, s)$ , produces a solution of cost at most  $R_{\mathcal{I}} = \min\{(3 + 4\lambda) \cdot \mathcal{A}_{\mathcal{I}}, (1 + \frac{1}{2\lambda})(\lceil \log(n) \rceil + 1) \cdot \text{OPT}_{\mathcal{I}}\}$ , where  $\text{OPT}_{\mathcal{I}}$  and  $\mathcal{A}_{\mathcal{I}}$  denote the cost of an optimal solution and of the one obtained by  $\mathcal{A}$  on instance  $\mathcal{I}$ , respectively.*

Intuitively, the robustification framework  $\mathcal{R}$  of Theorem 1 carefully interpolates between the execution of algorithm  $\mathcal{A}$  and that of NN by executing the algorithms

in alternating phases. These phases are budgeted so that their costs are roughly proportional to each other, with the parameter  $\lambda > 0$  dictating the proportionality.

As a second contribution, we apply the robustification framework within the context of *learning-augmented algorithm design*. Given the tremendous success of artificial intelligence, the assumption of having no prior knowledge about the graph may be overly pessimistic. Instead, we might have access to machine-learned predictions about good exploration decisions. Such predictions are typically imperfect; they usually have a good quality but may be arbitrarily bad. A recent line of research is concerned with the design of online algorithms that exploit access to predictions of unknown quality (e.g. [9]) to achieve a better performance if the predictions are of high quality, while at the same time maintaining worst-case guarantees even if they are arbitrarily wrong. For online graph exploration, we consider predictions that suggest a known, but unexplored vertex as the next target to a learning-augmented algorithm. Predictions may be computed dynamically and use all data collected so far, which is what one would expect in practice. This rather abstract requirement allows the implementation of various prediction models. We consider two kinds of predictions, namely *tour predictions* and *tree predictions*, where the suggested vertex is the next unexplored vertex of a TSP tour or of a Depth First Search (DFS) tour corresponding to some predicted spanning tree, respectively. The prediction error  $\eta$  is the difference between the total exploration cost of following these per-step suggestions and that of following a perfect prediction w.r.t. the given prediction model (tour respectively tree predictions). Using the robustification framework  $\mathcal{R}$  of Theorem 1 with these predictions leads to the following theorem.

**Theorem 2** *For any  $\lambda > 0$  and instance  $\mathcal{I}$ , there is an algorithm for the online graph exploration problem that uses a predicted spanning tree or tour with a cost of at most  $\min\{(3 + 4\lambda)(\kappa \text{OPT}_{\mathcal{I}} + \eta), (1 + \frac{1}{2\lambda})(\lceil \log(n) \rceil + 1)\text{OPT}_{\mathcal{I}}\}$ , where  $\kappa = 1$ , for tour predictions,  $\kappa = 2$ , for tree predictions, and  $\text{OPT}_{\mathcal{I}}$  denotes the cost of the optimal solution.*

We show that our predictions are learnable in the sense of PAC learnability under the assumptions that the given graph is complete and its size known. Finally, we empirically confirm the power of using predictions and the effectivity of the robustification framework by empirically evaluating the performance of our algorithms on real-world instances.

## References

- [1] Piotr Berman. On-line searching and navigation. In *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 232–241. Springer, 1996.
- [2] Nageswara S. V. Rao, S. S. Iyengar, C. C. Jorgensen, and Charles R. Weisbin. Robot navigation in an unexplored terrain. *J. Field Robotics*, 3(4):389–407, 1986.

- [3] Franziska Eberle, Alexander Lindermayr, Lukas Nölke, Nicole Megow, and Jens Schlöter. Robustification of online graph exploration methods. *CoRR*, abs/2112.05422, 2021.
- [4] Bala Kalyanasundaram and Kirk Pruhs. Constructing competitive tours from local information. *Theor. Comput. Sci.*, 130(1):125–138, 1994.
- [5] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, and D.B. Shmoys. *The Traveling Salesman Problem – A Guided Tour of Combinatorial Optimization*. Wiley, 1985.
- [6] Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6(3):563–581, 1977.
- [7] Nicole Megow, Kurt Mehlhorn, and Pascal Schweitzer. Online graph exploration: New results on old and new algorithms. *Theor. Comput. Sci.*, 463:62–72, 2012.
- [8] Alexander Birx, Yann Disser, Alexander V. Hopp, and Christina Karousatou. An improved lower bound for competitive graph exploration. *Theor. Comput. Sci.*, 868:65–86, 2021.
- [9] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311. PMLR, 2018.